ORIGINAL ARTICLE
Journal Section

Active learning for efficiently training emulators of computationally expensive mathematical models

Alexandra G. Ellis[1,2] | Rowan Iskandar[1] | Christopher H. Schmid[1,3] | John B. Wong[4] | Thomas A. Trikalinos[1]

[1]Center for Evidence Synthesis in Health, Brown University School of Public Health, Providence, RI 02912, US
[2]Institute for Clinical and Economic Review, Boston, MA 02109, US
[3]Department of Biostatistics, Brown University School of Public Health, Providence, RI 02912, US
[4]Division of Clinical Decision Making, Tufts Medical Center, Boston, MA 02111, US

Correspondence
AG Ellis, Center for Evidence Synthesis in Health, Brown University School of Public Health, Providence, RI 02912, US
Email: alexandra_g_ellis@brown.edu

Funding information
AG Ellis was supported by a dissertation research grant from the Agency for Healthcare Research and Quality. Grant Number: R36HS24653-01

An emulator is a fast-to-evaluate statistical approximation of a detailed mathematical model (simulator). When used in lieu of simulators, emulators can expedite tasks that require many repeated evaluations, such as model calibration and value-of-information analyses. Emulators are developed using the output of simulators at specific input values (design points). Developing an emulator that closely approximates the simulator can require many design points, which becomes computationally expensive. We describe a self-terminating active learning algorithm to efficiently develop emulators tailored to a specific emulation task. Its postulated advantages over the prevalent approaches include (1) self-termination and (2) development of emulators with smaller mean squared errors. To explicate, we develop and compare Gaussian Process emulators of a prostate screening model using the adaptive algorithm versus standard approaches.

KEYWORDS
meta-model, surrogate model, kriging, adaptive design, sequential design, Gaussian Processes

Abbreviations: GP: Gaussian Process; LHS, Latin hypercube sampling; PSA, Prostate-specific Antigen.

1 | INTRODUCTION

Many decisions in health must be made under uncertainty or with incomplete understanding of the underlying phenomena. In such cases, mathematical modeling can help decision-makers synthesize evidence from different sources, estimate and aggregate anticipated outcomes while accounting for stakeholder preferences, understand trade-offs, and quantify the impact of uncertainty on decision making [1]. To be informative, a model should be detailed enough to capture salient aspects of the decisional problem at hand, but a highly detailed model can render routine analyses computationally expensive, hindering its usability.

As an example, consider the prostate-specific antigen (PSA) growth and prostate cancer progression (PSAPC) model, which is a microsimulation model that projects outcomes for PSA-based screening strategies defined by combinations of screening schedules and age-specific PSA positivity thresholds [2, 3]. Although evaluating the expected outcomes of one screening strategy takes just a few minutes on modern computer hardware, analyses that require many (on the order of 10^4 to 10^6) model evaluations, such as *calibration* of model parameters to external data [4, 5], *sensitivity analysis* [4, 6, 7], and *uncertainty analysis* [4, 8], become expensive if not impractical. For example, in part because of the computational burden, Gulati and colleagues evaluated only 35 screening strategies [3], out of more than 10^7 policy-relevant strategies that could have been explored [9].

A way to mitigate the computational cost of such analyses is to use *emulators* of the original mathematical models. Emulators, also called surrogate models or meta-models, are approximations of the original models, hereafter referred to as *simulators* for consistency with prior works [7, 8, 10, 11]. Once developed, emulators are orders of magnitude faster to evaluate than the simulators and, thus, can be used instead of, or in tandem with, the simulators to perform computationally intensive tasks [7, 12, 11]. Although uncommon in healthcare applications [13, 14, 15], using emulators in place of simulators is a well-established practice in mechanical, electrical, and industrial engineering, geology, meteorology, and military modeling [7, 16, 17, 4, 8].

Developing an emulator that approximates a simulator over an input domain is analogous to estimating a response surface, so it is an experimental-design problem [18]. In this context, the simulator is used to generate data (a *design*), and an emulator is fit to the design. In this work we describe an adaptive algorithm for developing designs that are tailored to a specific simulator and a particular emulation task. We compare the performance of emulators generated with the adaptive design versus with the standard approach of Latin Hypercube Sampling (LHS), which generates generic designs that are agnostic about the emulation task [19] and distribute points across intervals of the input domains.

This work is organized as follows. In Section 2, we describe desirable characteristics for emulators of simulators in health to motivate our choice of Gaussian Process emulators. In Section 3, we introduce an adaptive design for developing emulators, which is the focus of this work. Section 4 uses the PSAPC model to explicate the development of emulators that predict life expectancy over no screening for large sets of practically-implementable prostate cancer screening strategies. Although our adaptive design is meant for very expensive simulators (that, e.g., take days rather than minutes to evaluate), we use the PSAPC simulator as a model with which we can actually do full computations. We compare emulators developed with the sequential design versus those developed with LHS in Section 5, and conclude with key remarks (Section 6).

2 | EMULATION GOALS AND EMULATOR MODELS

2.1 | Simulators

For this exposition, a simulator is a deterministic smooth function $f : \mathbb{R}^K \rightarrow \mathbb{R}$ that maps from K input parameters to scalar outputs. K is typically in the many dozens or hundreds. We treat the simulator as a *black box* function that can be evaluated (expensively) at specific input values. Typically, a baseline set of values $\boldsymbol{z}^* \in \mathbb{R}^K$ has been specified for the simulator through data analysis, calibration, or expert knowledge.

While this setup is not the most general case for which we could present our algorithm, it is not as restrictive as it appears for the following reasons: (i) Many simulators in health that are not smooth are Lipschitz-continuous (i.e. have no “extreme” changes in the output given a marginal change in input) and, for the precision demanded in practice and for our purposes, can be treated as if they were smooth functions. (ii) Simulator outputs can be random variables, e.g., if some inputs are random variables, but we are usually interested in expectations of outcomes, which marginalize over the random input variables and result in deterministic mappings. (iii) Most simulators have multidimensional outputs, but, often, we are interested in one critical or composite outcome or are willing to consider one outcome at a time. (iv) Finally, we treat the simulator as a black box and do not attempt to exploit its analytical form. Often the analytical form is not available, or, when it is, its theoretical analysis may severely test our mathematical ability or be intractable.

With reference to more general setups for emulator development, in Section 6.2 we argue that extensions of our work to stochastic simulators and to simulators with multidimensional outputs require only minor modifications of the adaptive design algorithms that we present below.

2.2 | Goal of emulation

We aim to develop an emulator $f^*(\cdot)$ that statistically approximates the simulator’s input-output mapping over the domain $\mathcal{X} \subseteq \mathbb{R}^k$ of the simulators’ input parameters, where $1 \leq k \leq K$ [8]. We seek

$$f^*(\boldsymbol{x}) \cong f(\boldsymbol{x}, \boldsymbol{w}^*) \,\forall\, \boldsymbol{x} \in \mathcal{X}, \tag{1}$$

where $\boldsymbol{x}$ are values for the k inputs whose mapping we wish to emulate, $\boldsymbol{w}^* \in \mathbb{R}^{K-k}$ are values for the remaining inputs of the simulator that are kept equal to their corresponding elements in the baseline-values vector $\boldsymbol{z}^*$, and the symbol “$\cong$” means “either equal or close enough for the purpose of the application”. From now on we will be writing $f(\boldsymbol{x})$ instead of $f(\boldsymbol{x}, \boldsymbol{w}^*)$ to ease notation. We assume that $\mathcal{X}$ is a *polytope*, i.e., a convex polyhedron in $\mathbb{R}^k$, which is most often the case in applications.

Approximating the behavior of the emulator *over all vectors* in $\mathcal{X}$ is a reasonable goal when we wish to gain insights about the behavior of the simulator or for sensitivity and uncertainty analysis. For different tasks, e.g., for calibration of input variables to external data it may suffice to approximate the simulator in the *neighborhood of the set of optimal values* [20].

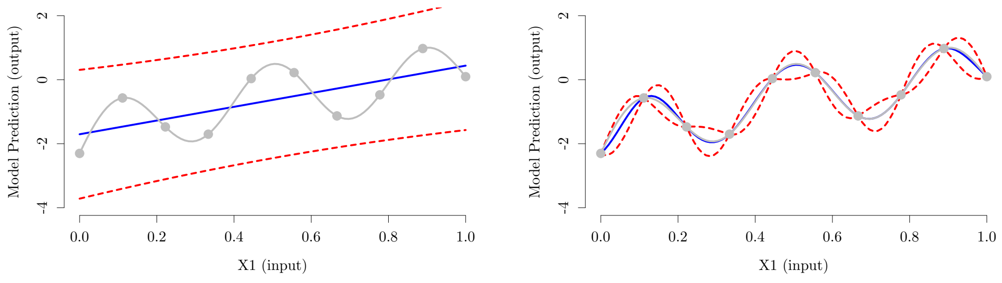
To fit $f^*(\cdot)$, we need a design that specifies the set of n points in $\mathcal{X}$ at which we will evaluate the simulator. Let $\mathcal{X}_n = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ be a set of distinct vectors in $\mathcal{X}$ that includes the extreme vectors of $\mathcal{X}$, so that all vectors in $\mathcal{X}$ are convex combinations of vectors in $\mathcal{X}_n$. Then, with some abuse of terminology, we will call the vectors in $\mathcal{D}_n = \{(\boldsymbol{x}^T, f(\boldsymbol{x}))^T : \boldsymbol{x} \in \mathcal{X}_n\}$ the *design vectors* or *design points*. We further simplify notation by writing $(\boldsymbol{x}, f(\boldsymbol{x}))$ instead of $(\boldsymbol{x}^T, f(\boldsymbol{x}))^T$ for each vector in $\mathcal{D}_n$.

2.3 | Choice of emulator model

We follow others in requiring that the emulator be an *exact interpolator* [8, 4, 11, 7, 10]. Specifically, we demand that

1. $f^*(\boldsymbol{x}) = f(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{X}_n$, that is, the emulator's prediction should agree with the simulator's output value at the design points because the simulator is deterministic.
2. At all other input vectors, the emulator must provide an interpolation of the simulator output value, whose uncertainty decreases when closer to a design point, so that it becomes 0 at the design points.
3. The emulator should be orders of magnitude faster to evaluate than the simulator.

Despite the fact that practically all statistical and machine learning models are fast to evaluate, i.e., they satisfy criterion 3, approaches such as linear regressions and neural networks do not satisfy the first two criteria, whereas kernel-based methods such as Gaussian Processes (GPs), do. Figure 1 helps build intuition. Therefore, despite its popularity in the literature in part due to its simple form and familiarity by researchers [21, 7], regression is not a preferred emulator type in this context. We use GP-based emulation, also referred to as *kriging* in other fields [8, 4, 11, 7, 10]. We review GPs briefly in Appendix A.



(a) Regression emulator.

(b) Gaussian Process emulator.

FIGURE 1 Two emulator models. Two emulator functions predict the output of a simplistic model (grey curve). The input is a single variable (X1, horizontal axis). The 10 grey points represent the data ("design points") used to fit the emulators. The solid blue curves are the emulators' predictions, and the areas between the red dashed curves are the associated 95% prediction intervals. The prediction from the linear regression emulator (a) passes near, but not through, all design points. Further, the prediction uncertainty is not zero at the design points. The Gaussian Process emulator (b) passes through all design points, and the uncertainty of its predictions has the desired pattern. Figure adapted from [4].

2.4 | Experimental designs for developing emulators

The most common experimental designs for emulators are space-filling designs and sequential designs. *Space-filling designs* such as LHS and its variants choose B input points from $\mathcal{X}$ ensuring that pre-specified intervals of each input variable are sampled. Space filling designs sample points regardless of an input variable's influence on the output variable [22, 23, 24]. Alternatively, *sequential designs* make use of the estimated relationship between inputs and outputs in the simulator [11, 7, 25]. Broadly, these approaches fit an emulator based on an initial set $\mathcal{D}_{n_0}$ of n_0 design points and then choose an additional new design point based on pre-established criteria related to the emulator's fit or uncertainty. When a new design point is chosen, it is added to the current set of design points and the emulator is refit with this new

BOX 1 DESIGN POINT ALGORITHM (ALGORITHM 1)

Start with a seeding set $\mathcal{X}_{n_0}$ of n_0 input vectors. At least one vector in the set is in the interior of the polytope of $\mathcal{X}_{n_0}$. Call $\mathcal{D}_{n_0}$ the set of corresponding design points.

```
For each iteration $t = 0, 1, \ldots$ until criteria are met:
  0. Fit an EMULATOR $f^*(\cdot)$ to all $n_0 + t$ design points in $\mathcal{D}_{n_0+t}$
  1. For the $i$-th interior input vector $\boldsymbol{x}_i$:
      1.1. Remove the design point $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))$
      1.2. Refit EMULATOR $f^*_{(-i)}(\cdot)$ to the remaining design points in $\mathcal{D}_{n_0+t} \setminus \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}$
  2. Obtain a set $C_t$ of candidate input vectors (see Appendix B)
  3. For each candidate input vector $\boldsymbol{c}_t \in C_t$:
      3.1. Estimate predictions $f^*(\boldsymbol{c}_t)$, and $f^*_{(-i)}(\boldsymbol{c}_t)$ for all $f^*_{(-i)}(\cdot)$ in Step 1.2. Get the range $R(\boldsymbol{c}_t)$ of predictions.
  4. Identify the candidate input vector $\boldsymbol{c}^*_t = \underset{\boldsymbol{c}_t \in C_t}{\text{argmax}}\ R(\boldsymbol{c}_t)$
  5. IF $R(\boldsymbol{c}^*_t) \geq T_{resample}$ , where $T_{resample}$ is a predefined threshold:
      - Add design point $(\boldsymbol{c}^*_t, f(\boldsymbol{c}^*_t))$ to $\mathcal{D}_{n_0+t}$ and return to Step 0.
    ELSE:
      5.1. Estimate the standard error of predictions $\hat{SE}(f^*(\boldsymbol{c}_t))$ at all $\boldsymbol{c}_t \in C_t$
      5.2. Identify the candidate point $\boldsymbol{c}^{**}_t$ with $\underset{\boldsymbol{c}_t \in C_t}{\text{argmax}}[\hat{SE}(f^*(\boldsymbol{c}_t))]$
      5.3. IF $\hat{SE}(f^*(\boldsymbol{c}^{**}_t)) \geq T_{SE}$ , where $T_{SE}$ is a predefined threshold:
          - Add design point $(\boldsymbol{c}^{**}_t, f(\boldsymbol{c}^{**}_t))$ to $\mathcal{D}_{n_0+t}$ and return to Step 0.
        ELSE: END
```

set of $n_0 + 1$ design points. The process repeats until pre-established stopping criteria are met.

LHS and other space-filling designs are easy to implement and are available in many software packages (e.g., the `lhs` package [26] in `R`). The number B of design points is fixed in advance in the form of a computational budget; however, there is little guidance on pre-determining the adequate number of points. Furthermore, although LHS performs well when the simulator behaves similarly across the input domain, it may approximate poorly when the simulator behaves qualitatively differently in an input subdomain that is not sampled [27]. By contrast, sequential designs do not require the user to specify the number of design points in advance, but rather allow them to continue to add design points until the stopping criteria are met. They may be more efficient in that they may require fewer design points than fixed designs like LHS [7, 28]. However, sequential designs may be computationally expensive due to the re-estimation of the emulator's parameters after each additional design point [29].

3 | AN ACTIVE LEARNING ALGORITHM

The proposed active learning (AL) algorithm (Box 1: ALGORITHM 1) adopts aspects of sequential designs from the current literature [7]. The algorithm seeks to select design points in the regions where (i) the simulator output changes quickly and (ii) the emulator has high predictive uncertainty. The first goal is pursued through a resampling procedure which successively removes existing design points from the complete set of design points, refits emulators using each of the resulting subsets of design points, and then identifies the "candidate" input vectors with the largest range in predictions obtained using this series of emulators. Candidate input vectors are those input vectors that have not yet been evaluated with the simulator. With this resampling procedure, the regions where the simulator is "fast-changing" are likely to have larger ranges in predictions than in regions where the simulator is relatively flat [18]. For example, removing a design point from a region where small changes in a PSA threshold value produce large changes in life-years saved will likely change the prediction of a nearby PSA threshold value. If the resampling procedure does not identify any "fast-changing" regions (towards satisfying the first goal above), then the second goal is pursued by examining the regions with large variance in the emulator's prediction. Choosing additional design points in these regions will reduce the emulator's overall predictive uncertainty. Details of the algorithm's process are described below, and Figure 2 illustrates the steps of the algorithm using a simplistic 1-dimensional example.

To start, the algorithm requires an initial set $\mathcal{X}_{n_0}$ of n_0 input vectors, which includes the n_* extreme vertices of the input space $\mathcal{X}$ and a non-empty set of $n_0 - n_*$ input vectors which are in the interior of the polytope of $\mathcal{X}$ (Figure 2 (a)). The initial input vectors are paired with their output values evaluated in the simulator $f(\cdot)$ to obtain the set $\mathcal{D}_{n_0} = \{(\boldsymbol{x}, f(\boldsymbol{x})) : \boldsymbol{x} \in \mathcal{X}_{n_0}\}$ of initial design points (Figure 2 (b)). An emulator $f^*(\cdot)$ is then fit to $\mathcal{D}_{n_0}$ (Step 0, Figure 2 (c)). Next, for each interior input vector $\boldsymbol{x}_i$, the corresponding design point $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))$ is removed from the complete set of design points (Step 1.1) and a new emulator $f^*_{(-i)}(\cdot)$ is fit to the remaining set $\mathcal{D}_{n_0} \setminus \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}$ of $n_0 - 1$ design points (Step 1.2, Figure 2 (d)). A set $\mathcal{C}$ of candidate input vectors with "large" prediction errors from the current emulator are then obtained (Step 2). We describe a simple algorithm for selecting candidate points in Appendix B. For each candidate input vector $\boldsymbol{c} \in \mathcal{C}$, predictions are estimated using the emulator $f^*(\cdot)$ and each re-fit emulator $f^*_{(-i)}(\cdot)$ from Step 1.2. (Step 3.1, Figure 2 (e)); the range $R(\boldsymbol{c})$ of these predictions is obtained (Step 3.2, Figure 2 (f)). The "winning" candidate input vector $\boldsymbol{c^*} \in \mathcal{C}$ is the one with the largest range in predictions (Step 4). If the range from this candidate input vector is above a pre-specified threshold (i.e., $R(\boldsymbol{c}^*) \geq T_{resample}$), the new design point $(\boldsymbol{c}^*, f(\boldsymbol{c}^*))$ is added to the set of existing design points and we return to Step 0 (Step 5, Figure 2 (g)). Step 0 to Step 5 are repeated until the range of predictions for $\boldsymbol{c^*}$ is below the threshold $T_{resample}$ (Figure 2 (h)). Then, the standard error of the predictions $\hat{SE}(f^*(\boldsymbol{c}))$ with emulator $f^*(\cdot)$ is estimated for each candidate input vector (Step 5.1, Figure 2 (i)), and the candidate input vector $\boldsymbol{c}^{**} \in \mathcal{C}$ with the largest prediction error is identified (Step 5.2). If the prediction error for this candidate input vector is above a pre-specified threshold (i.e., $\hat{SE}(f^*(\boldsymbol{c}^{**})) \geq T_{SE}$), then the new design point $(\boldsymbol{c}^{**}, f(\boldsymbol{c}^{**})))$ is added to the set of existing design points and we return to Step 0 (Step 5.3). The process repeats until both criteria are satisfied.

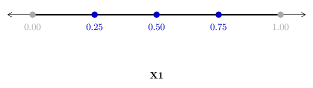
3.1 | Implementation

The AL algorithm is implemented in `R` and uses a modified version of the `GPfit` package [30] for fitting GP emulators. When refitting emulators during iterations of the active learning algorithm, we used the initial values of previously fit emulators as initial values. We use the `geometry` package [31] to obtain the tesselation of $\mathcal{X}_n$ that is required for the algorithm in Appendix B that identifies the candidate points at each iteration of the active learning algorithm, and the package `lhs` to generate LHS designs [26].

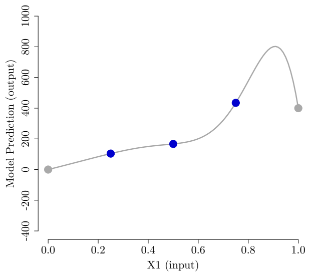
4 | APPLICATION

4.1 | The PSAPC model

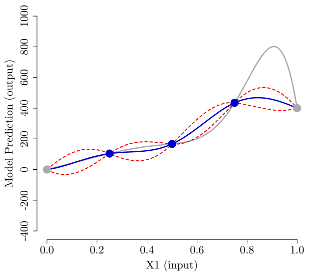
The PSAPC microsimulation model accounts for the relationship between PSA levels, prostate cancer disease progression, and clinical detection [32, 2]. The model, its estimation approach, its calibration, and its comparison with other



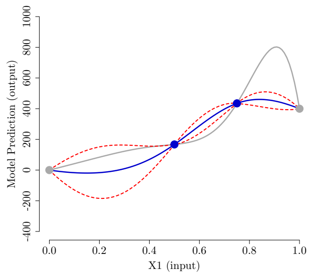
(a) Initial set-up: Input vectors.



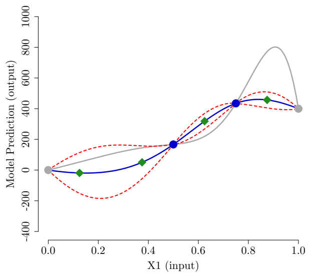
(b) Initial set-up: Design points.



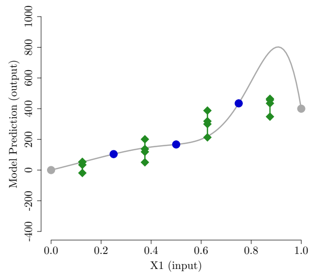
(c) Fit an emulator (Step 0).



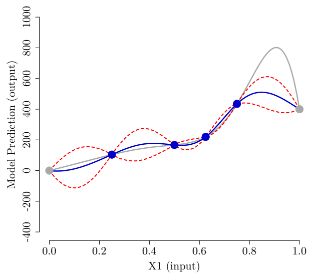
(d) Remove i-th interior point (Step 1.1) and re-fit emulator $f^*_{(-i)}(\boldsymbol{x})$ (Step 1.2).



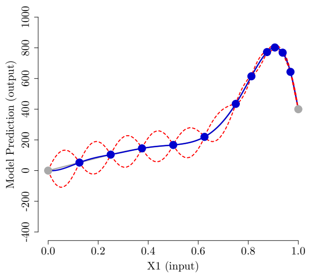
(e) For each candidate input (Step 2), obtain predictions using re-fit emulators (Step 3.1).



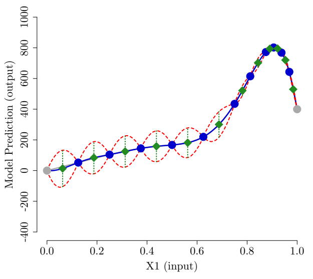
(f) Obtain range of predictions at each candidate input over $f^*(\boldsymbol{x})$ and all $f^*_{(-i)}(\boldsymbol{x})$ (Step 3.2).



(g) With the candidate input with the largest range in predictions (Step 4), fit a new emulator (Step 5).



(h) Repeat Steps 0-5 until range of predictions is below threshold.



(i) Estimate standard error of predictions (Step 5.1) and identify input with largest prediction error (Step 5.2).

FIGURE 2 1-dimensional example of ALGORITHM 1. Blue points: interior design points; Grey points: design points corresponding to extreme vertices; Grey curve: simulator; Blue curve: emulator prediction. Red dashed curve: emulator's 95% prediction intervals.

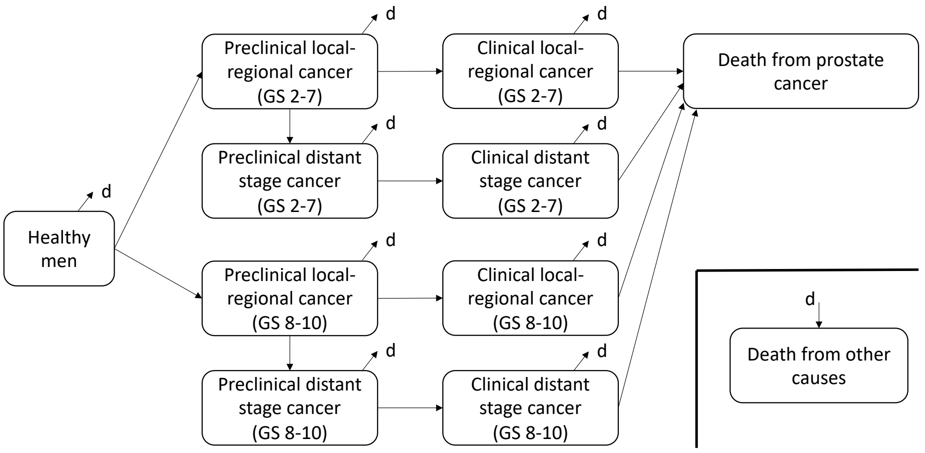

Healthy men
Preclinical local-regional cancer (GS 2-7)
Preclinical distant stage cancer (GS 2-7)
Preclinical local-regional cancer (GS 8-10)
Preclinical distant stage cancer (GS 8-10)
Clinical local-regional cancer (GS 2-7)
Clinical distant stage cancer (GS 2-7)
Clinical local-regional cancer (GS 8-10)
Clinical distant stage cancer (GS 8-10)
Death from prostate cancer
Death from other causes
d

FIGURE 3 PSAPC model natural history of disease progression. Healthy, preclinical, clinical, prostate cancer mortality, and other-cause mortality states in the absence of screening. Rounded rectangle represent the various health states. Arrows between rectangles represent allowable transitions between health states. People develop preclinical local-regional or distal disease, which may manifest clinically. Patients can die of prostate cancer only after they have developed clinical local regional or distal disease. People can die of other causes from any "alive" health state. For simplicity, transitions from the "alive" health states to "death from other causes" are not drawn explicitly, but are depicted by the broken arrows and the letter "d". GS: Gleason Score.

prostate cancer models have been described in detail elsewhere [2, 3, 33, 34]. Here, we treat the PSAPC model as a "black box".

Figure 3 outlines the PSAPC model. Briefly, simulated healthy men may develop preclinical, local-regional cancer (disease onset). The PSAPC version we are using incorporates disease grade (low=Gleason scores 2-7; high=Gleason scores 8-10) which is determined and fixed upon disease onset. Patients with low- or high-grade, local-regional cancer may progress to distant sites (metastatic spread). Patients with either local-regional or metastatic disease may present with symptoms (clinical detection). Those with a clinically-detectable form of disease may die from prostate cancer (prostate cancer mortality). At any time and any health state in the model, patients may die from other causes (other-cause mortality). Disease progression is related to age or PSA levels. PSA levels are modeled as a function of age and age of disease onset, such that there is a linear changepoint in (log) PSA after disease onset. PSA levels after disease onset differ for those with low versus high-grade disease. Parameters for the age of disease onset, metastatic spread, and clinical detection are estimated from calibration.

In the presence of screening, the simulated individuals with cancer may be identified and treated earlier (i.e., during preclinical state) than without screening. The model assigns each simulated individual a schedule of PSA-based screening tests and biopsies, as determined by the simulated screening strategy. Every time screening occurs, men with PSA levels above the PSA positivity threshold are referred to biopsy, and those with a positive biopsy result are managed with radical prostatectomy, radiation therapy, or active surveillance (i.e., no treatment but continued monitoring) [32, 2].

The PSAPC model projects several outcomes, including the number of screenings, false-positive results, prostate cancer diagnoses, prostate cancer deaths, and life-years gained with PSA-screening versus no screening (i.e., clinical detection only). The model results are presented as the mean number of events or the lifetime probability of each outcome based on the simulated cohorts of men (e.g., 100 million men) [3].

4.2 | Example goal: Life expectancy with different PSA positivity thresholds for a set of screening policies

Using the PSAPC model described above, we aimed to estimate the expected life-days saved with PSA-based screening versus no screening for policy-relevant screening strategies that used age-specific PSA thresholds. We considered all annual PSA screening strategies that used PSA positivity thresholds between 0.5 and 6.0 ng/mL and used four age-specific PSA positivity thresholds ($k = 4$): one for men aged 40-44 years (PSA_{40-44}), a second for men aged 45-49 years (PSA_{45-49}), a third for men aged 50-54 (PSA_{50-54}), and a fourth for men aged 55-74 (PSA_{55-74}). Because PSA levels increase with increasing age [35], we assumed "policy-relevant" strategies consist of those with age-specific PSA positivity thresholds being constant in each age-group, where the PSA positivity threshold value for a given age group was at least as high as the value in the preceding age group. In sensitivity analyses, we also considered each of $k = 1, 2, 3$ age-specific PSA positivity thresholds.

To obtain results with the PSAPC model with negligible Monte Carlo error, we simulated cohorts of 100 million men aged 40 in the year 2000 who were screened annually from age 40 to age 74.

4.3 | The PSAPC emulators

A series of GP emulators of the PSAPC model were developed to predict the expected life-days saved with screening versus no screening using as predictors the PSA-positivity thresholds in each age-group. The parameter values for the emulators' initial set-up and the selection of subsequent design points are outlined in Table 1. Briefly, an initial seeding set of $n_0 = 3k$ or $n_0 = 10k$ PSA positivity threshold values was sampled over the k-dimensional input space, which included the n_* extreme vertices of the input space and additional $n_0 - n_*$ randomly sampled interior vectors. Because the location of the seeding design points can affect the fit of the initial emulator, one emulator was fit to each of 10 initial seeding sets. The mean function $m(\cdot)$ for the GP-based emulators was a constant. Overall, there were 8 cases governing the initial set-up of the emulators (Table 1).

For each of the 8 initial cases, the GP emulators were developed using the proposed active learning algorithm (AL approach) and using a LHS space-filling design (LHS approach). The AL algorithm in Section 3 was applied for each of the 10 initial seeding sets of design points. The stopping criteria were 0.2 or 0.5 life-days gained for the resampling threshold ($T_{resample}$) and 0.5 life-days gained for the predictive uncertainty threshold (T_{SE}), because we judged that a precision in the order of 1 day was good enough for policy making in this application. The stopping criteria needed to be met for up to 5 consecutive iterations. With the LHS approach, sequential batches of 5 points ($P = 5$) were repeatedly sampled across the input space, evaluated in the PSAPC simulator, and added as design points to the previous set until 100 design points were obtained. Because LHS is a randomized routine, 10 LHS-based emulators were developed using different random seeds for each of the 10 initial seeding sets resulting in a total of 100 LHS-based emulators (10 replications of 10 random initial sets) for each of the 8 initial cases.

TABLE 1 **Set-up for emulators developed with the AL and LHS approaches.**

Parameter for emulator development	Values
Initial set-up (both approaches)	
Number of input variables (k)	$\{1, 2, 3, 4\}$
Number of initial design points (n_0)	$\{3k, 10k\}$
Number of initial seeding sets of design points	$\{10\}$
Emulator mean function ($m(\cdot)$)	{constant}
AL approach	
Resampling threshold in life-days (T_{resamp})	$\{0.2, 0.5\}$
SE threshold in life-days (T_{SE})	$\{0.5\}$
Number of consecutive iterations meeting threshold criteria	$\{1, 2, 3, 4, 5\}$
LHS approach	
Number of seeds per initial set of design points	$\{10\}$
Number of LHS samples per iteration (P)	$\{5\}$
Final number of design points* (B)	$\{100\}$

AL: active learning algorithm; LHS: Latin hypercube sampling.
*The LHS approach stopped selecting design points when 100 points had been selected. For $n_0 = 3k$, the final number of design points with LHS was 103 ($k = 1$), 101 ($k = 2$), 104 ($k = 3$), and 102 ($k = 4$).

4.4 | Performance metrics

To assess each emulator's "closeness" to the PSAPC simulator, the following metrics were estimated by evaluating the PSAPC simulator over a fine grid of M_k PSA positivity threshold values: (i) the square-root mean squared error $RMSE_k = \sqrt{\frac{1}{M_k}\sum_{m=1}^{M_k}(f^*(\boldsymbol{x}_m) - f(\boldsymbol{x}_m))^2}$, where $m = 1, \ldots, M_k$, which averages discrepancies over the whole input domain; and (ii) the maximum difference between the emulator prediction and the PSAPC simulator output, $MAX_k = \max_m |f^*(\boldsymbol{x}_m) - f(\boldsymbol{x}_m)|$. For both metrics, a lower value indicates better performance (i.e., closer to the PSAPC simulator). PSA positivity threshold values ranged from 0.5 to 6 ng/mL, with the PSA positivity threshold value for a given age group at least as high as the value in the preceding age group as described above. For $k = 4$, the number of evaluations was $M_4 = 11,616$.[1]

[1]The number of evaluations is as follows. For $k = 1$, there were 101 evaluations so that each PSA_{40-74} interval had length 0.01, yielding $M_1 = 101$. For $k = 2$, a fine grid with intervals of 0.02 was obtained, then restricted to the subset where the values for $PSA_{45-74} \geq PSA_{40-44}$. The 101 values from $k = 1$ were also included for a total of $M_2 = 1,376$ evaluations. For $k = 3$, a fine grid with intervals of 0.04 was obtained, then restricted to the subset where the values for $PSA_{50-74} \geq PSA_{45-49} \geq PSA_{40-44}$. The 1,376 values from $k = 2$ were also included for a total of $M_3 = 4,301$ evaluations. For $k = 4$, a fine grid with intervals of 0.05 was obtained, then restricted to the subset where the values for $PSA_{55-74} \geq PSA_{50-54} \geq PSA_{45-49} \geq PSA_{40-44}$. The 4,301 values from $k = 3$ were also included for a total of $M_4 = 11,616$.

5 | RESULTS

Figure 4 illustrates the performance of emulators fit with the AL approach using a resampling threshold of 0.5 life-days (solid black lines) and the LHS approach (dashed grey lines) for the case where $k = 4$ and $n_0 = 3k$. In particular, Figures 4 (a, b) show the RMSE and Figures 4 (c, d) show the MAX, as assessed at each iteration of the design point selection process. For both approaches, the addition of design point(s) may improve or worsen an emulator's performance, although the performance typically improves with more design points. The improvement tends to be greater in the beginning, when there are few design points, as indicated by the initial steep slopes of the curves. The curves level off as more design points were selected; the AL approach terminates shortly after the curves level (e.g., 40-50 design points) whereas the LHS approach exhausted the budget of 100 design points. At any number of design points, all AL-trained emulators performed at least as well as the LHS-trained emulators. A more extensive set of emulator result comparisons is available elsewhere [20].

Table 2 present the emulators' performance results for all scenarios. For the AL-trained emulators, the results are the emulators fit to the final set of design points (i.e., when the AL terminated). To facilitate comparisons with the LHS-based emulators, the tables present performance results of LHS-emulators fit to $n_0 + 5$, $n_0 + 10$, $n_0 + 50$, and the full budget of design points. The AL-trained emulators typically achieved a better performance more quickly (i.e., with fewer design points/simulator evaluations) than the LHS-trained emulators. For example, with $n_0 = 3k$ and stopping criteria of $T_{resample} = 0.5$ for 5 consecutive iterations, the median [range] MAX for emulators trained with the AL approach was 0.78 [0.37, 1.30] with between 26 and 50 design points whereas the MAX for LHS-based emulators was 0.86 [0.50, 1.44] with 62 design points. However, in practice, one would train the LHS emulators on the whole budget, where the respective results are 0.71 [0.47, 1.22]. Then, the AL-trained emulators would have attained comparable MAX metrics with 2 to 4 times fewer design points.

Across all analyses, there were gains in the performance of the AL-trained emulators versus the LHS-trained emulators with comparable numbers of design points when using more stringent stopping criteria (e.g, a lower resampling threshold or higher number of consecutive iterations).

6 | DISCUSSION

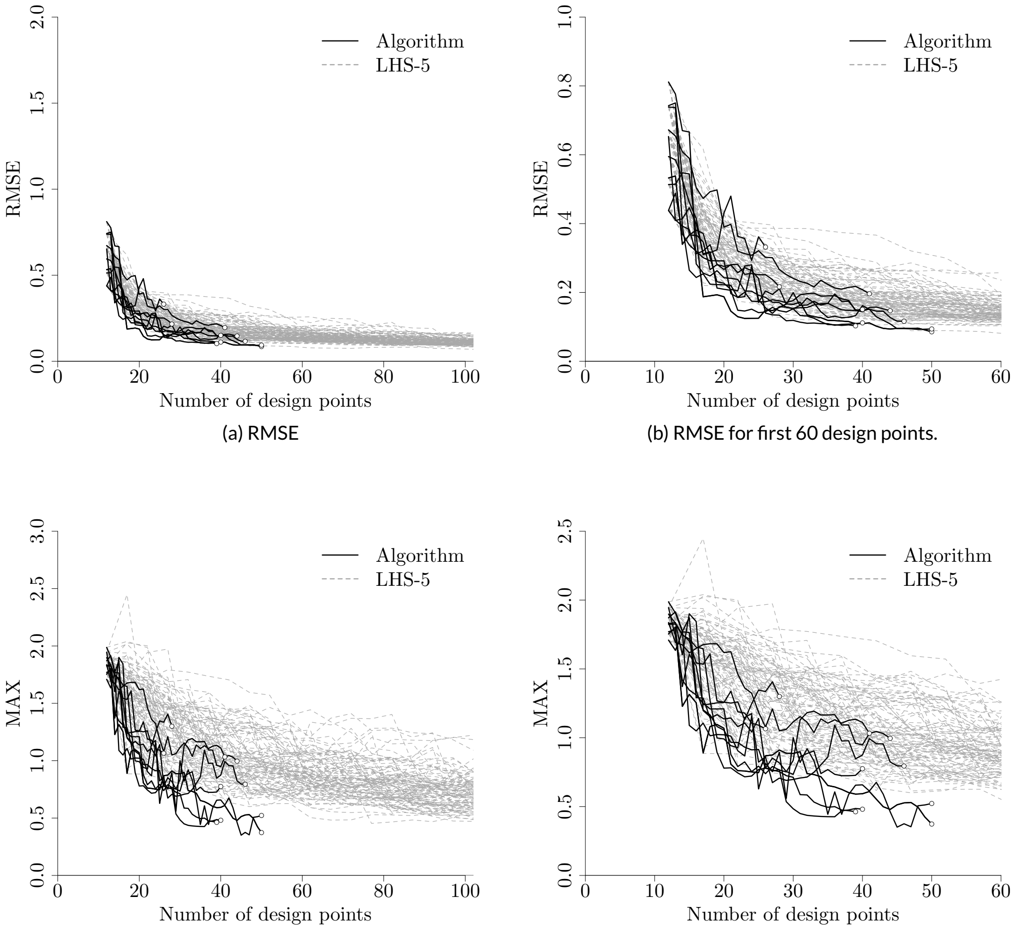
Our AL algorithm appears to be a reasonable choice for many emulation problems, because (i) it is self-stopping and (ii) it is efficient, in that it requires only a limited number of simulator evaluations to achieve a close-enough approximation to the simulator. The self-stopping attribute is important, because there is little guidance in selecting the total number of design points for GP emulators. "Rules of thumb" based on simulation testing suggest that trying about 100 points may be sufficient for a few input variables [36, 27]. Others have needed up to 400 points for emulators with 28 to 30 input variables [37]. By contrast, in all analyses, the AL algorithm self-terminated to yield well-performing emulators. We believe (but have not proven) that for smooth simulators, the AL algorithm will always terminate, because GPs converge to the underlying simulator with increasing number of design points [38]. Therefore, we believe that this work responds to the call by the Second Panel for Cost-Effectiveness in Health and Medicine for future research on "practical approaches to efficiently develop well-performing emulators" of mathematical models in healthcare [15, 13, 39].

Our results are consistent with the literature, which also suggests that sequential designs can approximate simulators with predictive accuracy that is at least comparable to or better than that of conventional space-filling designs [18, 40]. The efficiency of the AL algorithm is an important consideration for simulators that require substantially more computational resources compared to the PSAPC, such as the physiology-based chronic disease

TABLE 2 Overview of performance results for $k = 4$ and a constant mean function. n is the number of design points used to fit the emulators. For the active learning algorithm approach, the n in brackets is the range of design points when stopping criteria were met for 1, 2, 3, 4, or 5 consecutive iterations. Results for the MAX and RMSE are the median [range] for emulators developed with 10 different initial seeds (AL approach) and 10 replications of the 10 random initial seeds (LHS approach)

	$n_0 = 12$			$n_0 = 40$		
	n	MAX	RMSE	n	MAX	RMSE
Initial emulators	12	1.83 [1.71, 1.99]	0.62 [0.44, 0.81]	40	1.15 [0.82, 1.55]	0.25 [0.19, 0.40]
AL ($T_{resample} = 0.5$)						
# consecutive iterations						
1	[14-36]	0.98 [0.43, 1.75]	0.19 [0.10, 0.47]	[43-55]	0.78 [0.44, 1.24]	0.14 [0.11, 0.28]
2	[15-41]	0.86 [0.44, 1.70]	0.16 [0.11, 0.49]	[45-59]	0.71 [0.29, 0.86]	0.12 [0.07, 0.16]
3	[19-44]	0.77 [0.48, 1.68]	0.14 [0.10, 0.50]	[49-63]	0.67 [0.31, 0.88]	0.12 [0.07, 0.17]
4	[24-49]	0.76 [0.42, 1.41]	0.14 [0.09, 0.36]	[50-64]	0.67 [0.31, 0.96]	0.11 [0.07, 0.19]
5	[26-50]	0.78 [0.37, 1.30]	0.13 [0.09, 0.33]	[53-65]	0.68 [0.32, 0.87]	0.11 [0.07, 0.17]
AL ($T_{resample} = 0.2$)						
# consecutive iterations						
1	[14-39]	1.25 [0.86, 1.75]	0.31 [0.13, 0.51]	[43-63]	0.76 [0.27, 1.38]	0.13 [0.06, 0.28]
2	[19-54]	0.99 [0.47, 1.38]	0.21 [0.09, 0.38]	[49-70]	0.69 [0.27, 1.13]	0.11 [0.06, 0.22]
3	[25-55]	1.03 [0.46, 1.32]	0.21 [0.09, 0.34]	[50-71]	0.70 [0.27, 1.14]	0.11 [0.06, 0.22]
4	[26-56]	0.99 [0.46, 1.32]	0.17 [0.09, 0.34]	[56-72]	0.54 [0.27, 0.96]	0.09 [0.06, 0.27]
5	[27-57]	0.99 [0.45, 1.33]	0.16 [0.09, 0.32]	[57-73]	0.54 [0.27, 1.04]	0.09 [0.06, 0.31]
LHS						
$n_0 + 5$	17	1.59 [0.89, 2.45]	0.38 [0.22, 0.62]	45	0.97 [0.64, 1.85]	0.19 [0.12, 0.35]
$n_0 + 10$	22	1.47 [0.80, 2.00]	0.28 [0.17, 0.41]	50	0.94 [0.62, 1.74]	0.17 [0.10, 0.32]
$n_0 + 50$	62	0.86 [0.50, 1.44]	0.14 [0.08, 0.26]	90	0.74 [0.41, 1.38]	0.11 [0.07, 0.21]
last	102	0.71 [0.47, 1.22]	0.11 [0.07, 0.16]	100	0.70 [0.39, 1.32]	0.10 [0.07, 0.20]

AL: Active Learning algorithm; LHS: Latin Hypercube Sampling; MAX: Maximum deviation between emulator and simulator; RMSE: Square-Root Mean Squared Error.



(a) RMSE (b) RMSE for first 60 design points. (c) MAX (d) MAX for first 60 design points.

FIGURE 4 Performance of emulators with $k = 4$, $n_0 = 12$, $T_{resample} = 0.5$, and a constant mean function. From 10 initial seeding sets of design points, 10 emulators were fit using AL (black solid curve) and 100 emulators (10 per initial seed) were fit using LHS (grey dashed curves). (a) Square-root mean squared error (RMSE). (b) Magnification of the RMSE from (a) of the first 60 design points. (c) Maximum difference between emulator prediction and simulator results (MAX). (d) Magnification of the MAX from (c) of the first 60 design points.

model *Archimedes* [41, 42] or detailed modeling of the biomechanics of red blood cells [43]. In our case, we chose a computationally tractable simulator so as to quantify the evolution of RMSE and MAX metrics for emulators developed with different designs and to demonstrate computational efficiencies. Such calculations would be impractical with mathematical models that take hours or days, rather than minutes, to evaluate.

6.1 | Limitations

Several of the limitations pertain to the fact that we use GPs for emulation, which do not scale well to many inputs, do not extrapolate well, and can themselves be expensive to fit [38]. O'Hagan suggests that that GPs may be practical for up to 50 input dimensions [8]. If a large number of inputs must be emulated, one should explore whether developing several lower-dimensional emulators suffices. Knowing the analytic specification of the underlying model could offer insights about the feasibility of this simplification. GPs are interpolators and should not be used to extrapolate outside the input polytope. If such extrapolations are required, other types of emulators should be used, as the emulation problem has different goals that those described in Section 2.2.

The computational cost of learning GPs is about $O(n^3)$ [38] in the number n of design points, and for large n, fitting GPs becomes computationally expensive. The AL algorithm requires successively fitting a large number of GPs. However, it fits GPs that have one less or one more design point than an already fitted GP. Re-fitting in a minimally reduced or augmented dataset can be sped up by using previously-estimated parameter values as starting values for the fitting algorithm (as was done here), and with other approximations [44].

GPs trained with the AL algorithm are not guaranteed to converge to the simulator, whereas GPs trained with LHS will eventually do so [23]. Asymptotic convergence would probably be achieved by modifying step 5 in the AL algorithm to also allow choosing, with some probability, a set of random design points in an "exploration" step. However, this might undercut the AL algorithms' efficiency gains.

6.2 | Extensions

In some applications, we may be interested in both the expectation and the variance of the simulator output. The AL algorithm could be used by substituting stochastic GPs for the deterministic GPs that we have used here. Stochastic GPs model the error in the observation [38, 8, 7]. If a stochastic GP is used, some care is needed to avoid setting too stringent a resampling or SE threshold in the stopping criteria. For example, if the T_{SE} threshold is smaller than the simulator's prediction uncertainty, the AL algorithm will never terminate.

For emulating multivariate simulator outputs a multivariate GP can be used, which can also model correlations between the multiple outputs [38, 16, 45]. If multivariate emulators are to be used, the stopping criteria for the AL algorithm would also need to be modified, e.g., $T_{resample}$ and T_{SE} should be be met for each output, or for some function of the outputs. However, in practice, for a large class of models, there appears to be little gain attempting to learn multivariate emulators compared to several univariate ones [46, 47, 48].

The AL algorithm can be used with different types of emulators, including regression or neural network learners, although we have not examined this case in an example. We have also not explicitly discussed the case of *linked emulators*, that are used to approximate systems of computer simulators, or equivalently, distinct modules of a modular simulator, where the results of one module are used by other modules to yield the output of the overall simulator [48, 49]. This result suggests that we should be able to fit separate emulators for each simulator module with our AL algorithm. Finally, elsewhere, we build on this work to develop AL algorithms that train emulators to *a target accuracy*, albeit at an increased computational cost [20].

ACKNOWLEDGEMENTS

We thank Dr. Roman Gulati for providing the PSAPC model source code and for answering technical questions about the implementation.

CONFLICT OF INTEREST

We have no conflicts to declare.

REFERENCES

[1] Owens DK, Whitlock EP, Henderson J, Pignone MP, Krist AH, Bibbins-Domingo K, et al. Use of Decision Models in the Development of Evidence-Based Clinical Preventive Services Recommendations: Methods of the US Preventive Services Task Force Decision Modeling for USPSTF Recommendations. Annals of Internal Medicine 2016;165(7):501–508.

[2] Gulati R, Inoue L, Katcher J, Hazelton W, Etzioni R. Calibrating disease progression models using population data: a critical precursor to policy development in cancer control. Biostatistics 2010;11(4):707–719.

[3] Gulati R, Gore J, Etzioni R. Comparative effectiveness of alternative prostate-specific antigen–based prostate cancer screening strategies: Model estimates of potential benefits and harms. Annals of Internal Medicine 2013;158(3):145–153. +http://dx.doi.org/10.7326/0003-4819-158-3-201302050-00003.

[4] National Research Council. Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification. Washington, D.C.: NRC; 2012.

[5] Rutter CM, Miglioretti DL, Savarino JE. Bayesian calibration of microsimulation models. Journal of the American Statistical Association 2009;104(488):1338–1350.

[6] Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, et al. Global sensitivity analysis: the primer. John Wiley & Sons; 2008.

[7] Kleijnen J. Design and analysis of simulation experiments, vol. 20. Springer; 2008.

[8] O'Hagan A. Bayesian analysis of computer code outputs: A tutorial. Reliability Engineering & System Safety 2006;91(10–11):1290 – 1300. http://www.sciencedirect.com/science/article/pii/S0951832005002383, the Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004).

[9] Bertsimas D, Silberholz J, Trikalinos TA. Optimal healthcare decision making under multiple mathematical models: application in prostate cancer screening. Health care management science 2018;21(1):105–118.

[10] Santner TJ, Williams BJ, Notz WI. The design and analysis of computer experiments. Springer Science & Business Media; 2013.

[11] Sacks J, Welch W, Mitchell T, Wynn H. Design and analysis of computer experiments. Statistical science 1989;p. 409–423.

[12] Blanning RW. The construction and implementation of metamodels. simulation 1975;24(6):177–184.

[13] Neumann PJ, Kim DD, Trikalinos TA, Sculpher MJ, Salomon JA, Prosser LA, et al. Future Directions for Cost-effectiveness Analyses in Health and Medicine. Med Decis Making 2018 Oct;38(7):767–777.

[14] Jalal H, Goldhaber-Fiebert JD, Kuntz KM. Computing expected value of partial sample information from probabilistic sensitivity analysis using linear regression metamodeling. Medical Decision Making 2015;35(5):584–595.

[15] Neumann PJ, Sanders GD, Russell LB, Siegel JE, Ganiats TG. Cost-effectiveness in health and medicine. Oxford University Press; 2016.

[16] Conti S, O'Hagan A. Bayesian emulation of complex multi-output and dynamic computer models. Journal of Statistical Planning and Inference 2010;140(3):640 – 651. http://www.sciencedirect.com/science/article/pii/S0378375809002559.

[17] Kennedy M, Anderson C, Conti S, O'Hagan A. Case studies in Gaussian process modelling of computer codes. Reliability Engineering & System Safety 2006;91(10–11):1301 – 1309. http://www.sciencedirect.com/science/article/pii/S0951832005002395, the Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004).

[18] Kleijnen J, Van Beers W. Application-driven sequential designs for simulation experiments: Kriging metamodelling. Journal of the Operational Research Society 2004;55(8):876–883.

[19] McKay MD, Beckman RJ, Conover WJ. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 1979;21(2):239–245.

[20] Ellis AG. Emulators to facilitate the development and analysis of computationally expensive decision analytic models in health. PhD thesis, Brown University; 2018.

[21] İnci Batmaz, Tunali S. Small response surface designs for metamodel estimation. European Journal of Operational Research 2003;145(2):455 – 470. http://www.sciencedirect.com/science/article/pii/S0377221702002072.

[22] Lin CD, Bingham D, Sitter RR, Tang B. A new and flexible method for constructing designs for computer experiments. The Annals of Statistics 2010;p. 1460–1477.

[23] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 2000;42(1):55–61.

[24] Salagame RR, Barton RR. Factorial hypercube designs for spatial correlation regression. Journal of Applied Statistics 1997;24(4):453–474.

[25] Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. Journal of Global optimization 1998;13(4):455–492.

[26] Carnell R. lhs: Latin Hypercube Samples; 2016, https://CRAN.R-project.org/package=lhs, r package version 0.14.

[27] Meckesheimer M, Barton RR, Simpson T, Limayem F, Yannou B. Metamodeling of combined discrete/continuous responses. AIAA journal 2001;39(10):1950–1959.

[28] Park S, Fowler JW, Mackulak GT, Keats JB, Carlyle WM. D-optimal sequential experiments for generating a simulation-based cycle time-throughput curve. Operations Research 2002;50(6):981–990.

[29] Gano SE, Renaud JE, Martin JD, Simpson TW. Update strategies for kriging models used in variable fidelity optimization. Structural and Multidisciplinary Optimization 2006;32(4):287–298.

[30] MacDonald B, Ranjan P, Chipman H, et al. GPfit: An R package for fitting a gaussian process model to deterministic simulator outputs. Journal of Statistical Software 2015;64(i12).

[31] Barber C, Habel K, Grasman R, Gramacy RB, Stahel A, Sterratt DC. geometry: Mesh generation and surface tesselation. R Package Version 03-1, URL http://cran r-project org/web/packages/geometry/index html 2012;.

[32] CISNET, Cancer Intervention and Surveillance Modeling Network. Prostate Cancer Modeling;. Accessed: July 2015. http://cisnet.cancer.gov/prostate/.

[33] Gulati R, Tsodikov A, Etzioni R, Hunter-Merrill RA, Gore JL, Mariotto AB, et al. Expected population impacts of discontinued prostate-specific antigen screening. Cancer 2014;120(22):3519–3526.

[34] FHCRC, Fred Hutchinson Cancer Research Center (PSAPC) model profile; 2009 (accessed April 26, 2018). http://cisnet.cancer.gov/prostate/profiles.html.

[35] Oesterling JE, Jacobsen SJ, Chute CG, Guess HA, Girman CJ, Panser LA, et al. Serum prostate-specific antigen in a community-based population of healthy men: establishment of age-specific reference ranges. Jama 1993;270(7):860–864.

[36] Helton JC, Johnson JD, Oberkampf W, Sallaberry CJ. Sensitivity analysis in conjunction with evidence theory representations of epistemic uncertainty. Reliability Engineering & System Safety 2006;91(10):1414–1434.

[37] Rojnik K, Naveršnik K. Gaussian Process Metamodeling in Bayesian Value of Information Analysis: A Case of the Complex Health Economic Model for Breast Cancer Screening. Value in Health 2008;11(2):240 – 250. http://www.sciencedirect.com/science/article/pii/S1098301510605177.

[38] Rasmussen CE, Williams CK. Gaussian processes for machine learning, vol. 2. the MIT Press; 2006.

[39] Sanders GD, Neumann PJ, Basu A, Brock DW, Feeny D, Krahn M, et al. Recommendations for conduct, methodological practices, and reporting of cost-effectiveness analyses: second panel on cost-effectiveness in health and medicine. JAMA 2016;316(10):1093–1103.

[40] Jin R, Chen W, Simpson TW. Comparative studies of metamodelling techniques under multiple modelling criteria. Structural and Multidisciplinary Optimization 2001;23(1):1–13. http://dx.doi.org/10.1007/s00158-001-0160-4.

[41] Schlessinger L, Eddy DM. Archimedes: a new model for simulating health care systems—the mathematical formulation. Journal of biomedical informatics 2002;35(1):37–50.

[42] Eddy DM, Schlessinger L. Archimedes: a trial-validated model of diabetes. Diabetes care 2003;26(11):3093–3101.

[43] Tang YH, Lu L, Li H, Evangelinos C, Grinberg L, Sachdeva V, et al. OpenRBC: a fast simulator of red blood cells at protein resolution. Biophysical journal 2017;112(10):2030–2037.

[44] Shen Y, Seeger M, Ng AY. Fast gaussian process regression using kd-trees. In: Advances in neural information processing systems; 2006. p. 1225–1232.

[45] Genton MG, Kleiber W, et al. Cross-covariance functions for multivariate geostatistics. Statistical Science 2015;30(2):147–163.

[46] Kleijnen JP, Mehdad E. Multivariate versus univariate Kriging metamodels for multi-response simulation models. European Journal of Operational Research 2014;236(2):573–582.

[47] Gu M, Berger JO, et al. Parallel partial Gaussian process emulation for computer models with massive output. The Annals of Applied Statistics 2016;10(3):1317–1347.

[48] Kyzyurova K. On Uncertainty Quantification for Systems of Computer Models. PhD thesis, Duke University; 2017.

[49] Kyzyurova KN, Berger JO, Wolpert RL. Coupling Computer Models through Linking Their Statistical Emulators. SIAM/ASA Journal on Uncertainty Quantification 2018;6(3):1151–1171.

[50] Kleijnen J, Sargent R. A methodology for fitting and validating metamodels in simulation. European Journal of Operational Research 2000;120(1):14 – 29. http://www.sciencedirect.com/science/article/pii/S0377221798003920.

[51] Delaunay B. Sur la sphere vide. Bulletin of Academy of Sciences of the USSR 1934;7:793–800.

APPENDICES

A | Review of Gaussian Processes

A Gaussian Process is defined as *a collection of random variables, any finite number of which have a joint normal distribution* [38]. A Gaussian Process $f^*(\boldsymbol{x})$ is specified by its mean function

$$m(\boldsymbol{x}) = \mathbb{E}[f^*(\boldsymbol{x})]$$

and covariance function

$$\text{cov}\big(f^*(\boldsymbol{x}_i), f^*(\boldsymbol{x}_j)\big) = k(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)) = \mathbb{E}\Big[\big(f^*(\boldsymbol{x}_i) - m(\boldsymbol{x}_i)\big)\big(f^*(\boldsymbol{x}_j) - m(\boldsymbol{x}_j)\big)\Big].$$

We write the notation $f^*(\cdot) \sim \mathcal{GP}\big(m(\cdot), k(\cdot,\cdot)\big)$.

In practice, a simulator $f(\cdot)$ is evaluated n times to obtain a set of design points, $\mathcal{D}_n = \{(\boldsymbol{x}, f(\boldsymbol{x})) : \boldsymbol{x} \in \mathcal{X}_n\}$. Typically, the inputs $\boldsymbol{x}$ are transformed so that their domain is the unit cube. A Gaussian Process emulator model approximates the simulator $f(\cdot)$ with

$$f^*(\cdot) = m(\cdot) + \delta(\cdot),$$

where $\delta(\cdot) \sim \mathcal{GP}\big(\mathbf{0}, k(\cdot,\cdot)\big)$ is a zero-mean Gaussian Process and $\mathbf{0} = (0, \ldots, 0)^T$.

Commonly, the mean function is a constant model $m(\boldsymbol{x}) = \mu$, but it may also be a function of the input vectors, e.g., a linear component $m(\cdot) = \boldsymbol{h}(\cdot)^T\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ is a vector of p coefficients and $\boldsymbol{h}(\cdot)'$ is a vector of p known functions of the input vectors. Under this specification, the residuals from the linear regression are modeled by a zero-mean Gaussian Process. Although specifying a non-constant $m(\cdot)$ is not necessary, doing so may increase the smoothness of the fitted Gaussian Process and hence may require fewer design points for a "good" model fit [8].

The "behavior" of the zero-mean Gaussian Process $\delta(\cdot)$ is determined by the covariance function $k(\cdot,\cdot)$. Many choices for the covariance function exist; see [38] for a discussion. A common class of functions assumes a stationary covariance process, i.e., that the relationship between any two points depends only on their distance and not their location. Within this class, a typical choice for the covariance function is the squared exponential or radial basis function

$$k(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)) = \sigma^2 \exp\big(-(\boldsymbol{x}_i - \boldsymbol{x}_j)'\boldsymbol{\Theta}(\boldsymbol{x}_i - \boldsymbol{x}_j)\big) = \sigma^2 \prod_{d=1}^{k} \exp\big(-\theta_d(x_{di} - x_{dj})^2\big) = \sigma^2\, c(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)),$$

where σ^2 is the variance of the process, $\boldsymbol{\Theta} = \mathrm{diag}(\theta_1, \ldots, \theta_k)$ a diagonal matrix of k non-negative roughness parameters, and $c(\cdot,\cdot)$ the implied correlation function. Note that $c(f(\boldsymbol{x}_i), f(\boldsymbol{x}_i)) = 1$, and that the correlation between $\boldsymbol{x}_i, \boldsymbol{x}_j$ is positive and decreases with the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ [50, 38]. The roughness parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)'$ determine how quickly the correlation decreases in each dimension. A large value suggests a low correlation for a given dimension, even when two points are close.

Let $\boldsymbol{y} = (y_1, \ldots, y_n)' = (f^*(\boldsymbol{x}_1), \ldots, f^*(\boldsymbol{x}_n))'$ be a vector of observations modeled with a Gaussian Process. From the definition of the Gaussian Process, these observations are modeled as following an n-dimensional multivariate normal distribution. The parameters $\boldsymbol{\beta}, \sigma^2$, and $\boldsymbol{\theta}$ of the mean function and the covariance function can be obtained by optimizing the likelihood of the observations [7, 38, 30]. For example, for a constant mean function $m(\boldsymbol{x}) = \mu$, the mean μ and the variance σ^2 of the Gaussian Process can be expressed in closed form as a function of the roughness parameters

$$\mu = \frac{\mathbf{1}'\boldsymbol{R}^{-1}\boldsymbol{y}}{\mathbf{1}'\boldsymbol{R}^{-1}\mathbf{1}}, \text{ and}$$

$$\sigma^2 = \frac{(\boldsymbol{y} - \mathbf{1}\mu)'\boldsymbol{R}^{-1}(\boldsymbol{y} - \mathbf{1}\mu)}{n},$$

where $\mathbf{1} = (1, \ldots, 1)'$ and $\boldsymbol{R}$ is a $n \times n$ symmetric positive definite correlation matrix in which the correlation between the i-th and j-th observations is $c(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j))$. The negative profile log likelihood $L_{\boldsymbol{\theta}}$

$$-2\log(L_{\boldsymbol{\theta}}) \propto \log(|\boldsymbol{R}|) + n\log[(\boldsymbol{y} - \mathbf{1}\mu)'\boldsymbol{R}^{-1}(\boldsymbol{y} - \mathbf{1}\mu)]$$

is a function of the roughness parameters $\boldsymbol{\theta}$ because μ, σ^2, and $\boldsymbol{R}$ are functions of $\boldsymbol{\theta}$. In the equation above, $|\boldsymbol{R}|$ is the determinant of $\boldsymbol{R}$. Optimizing the likelihood yields estimates $\hat{\boldsymbol{\theta}}$ for $\boldsymbol{\theta}$, and thus estimates for μ, σ^2 and the correlation matrix $\boldsymbol{R}$.

The best linear unbiased predictor of the output value at a new input vector $\tilde{\boldsymbol{x}}$ is [7, 38]

$$f^*(\tilde{\boldsymbol{x}}) = \mu + \boldsymbol{r}'\boldsymbol{R}^{-1}(\boldsymbol{y} - \mathbf{1}\mu)$$

where $\boldsymbol{r}$ is a n-vector of correlations between $\tilde{\boldsymbol{x}}$ and the n input vectors from the design points, such that the i-th element of $\boldsymbol{r}$ is $c(f(\tilde{\boldsymbol{x}}), f(\boldsymbol{x}_i))$. The prediction error is [7, 38]

$$s^2(\tilde{\boldsymbol{x}}) = \sigma^2 \left[1 - \boldsymbol{r}'\boldsymbol{R}^{-1}\boldsymbol{r} + \frac{(1 - \mathbf{1}'\boldsymbol{R}^{-1}\boldsymbol{r})^2}{\mathbf{1}'\boldsymbol{R}^{-1}\mathbf{1}}\right].$$

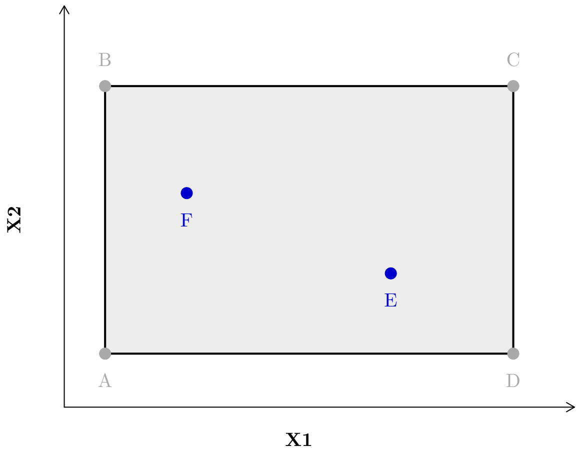
The maximum likelihood estimate $\hat{\boldsymbol{\theta}}$ is used to obtain $\boldsymbol{r}$ and $\boldsymbol{R}$ in the two prediction formulas above.

B | Algorithm for candidate points

The identification of the set $\mathcal{C}$ of candidate input vectors noted in ALGORITHM 1 is outlined in ALGORITHM 2 (Box 2). A second example is provided in Figure B.1 to illustrate this algorithm with a more complex 2-dimensional example, where the steps can be illustrated more clearly than in the 1-dimensional case. To start, the algorithm requires a set $\mathcal{D}_n$ of design points (Figure B.1 (a)), with $\mathcal{X}_n$ denoting the corresponding set of input vectors, and an emulator $f^*(\cdot)$ fit with $\mathcal{D}_n$.

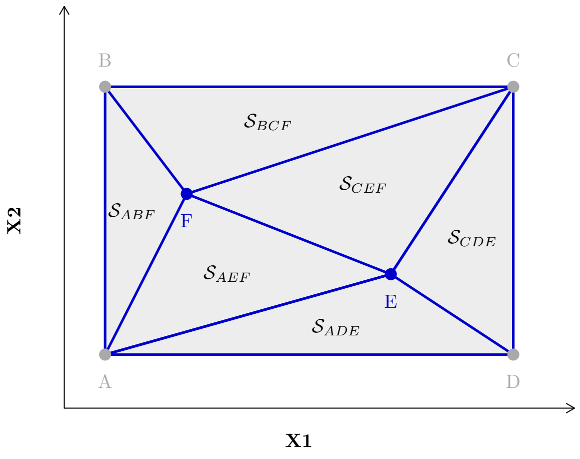
First, a triangulation of $\mathcal{X}_n$ is obtained (Step 1, Figure B.1 (b)). The triangulation $T(\mathcal{X}_n) = \cup \mathcal{S}_j$ partitions $\mathcal{X}_n$ in $k-$dimensional simplexes $\mathcal{S}_j$ using all $\boldsymbol{x} \in \mathcal{X}_n$ as vertices. The partitioning is such that $\mathcal{S}_j \cap \mathcal{S}_l$ for $j \neq l$ is either the empty set or a lower-dimensional simplex (a shared extreme point, line, or facet).

Any triangulation would do. For relatively small numbers of points (in the few hundreds), and few dimensions (say, less than 10), it is practical to use a Delaunay triangulation, for which many routines exist [51]. Within each returned simplex, we identify the vector $\boldsymbol{s}_j^*$ that maximizes the standard error of the predictions $\hat{SE}(f^*(\boldsymbol{c}))$ with emulator $f^*(\cdot)$, for all $\boldsymbol{s}_j \in \mathcal{S}_j$ (Step 1.1 in ALGORITHM 1, Figure B.1 (c)). The set $\mathcal{C}$ of unique $\boldsymbol{s}_j^*$ are selected as the candidate input vectors to be exploited in ALGORITHM 1.



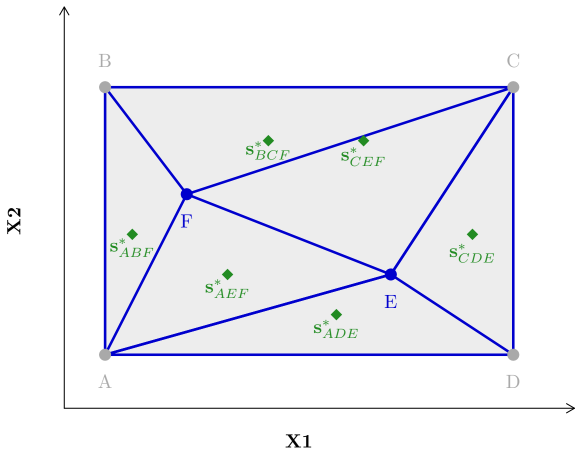
(a) **Initial set-up: Input vectors.**

The polytope of $\mathcal{X}_6$ is the grey shaded rectangle, which includes the extreme vertices A, B, C, D (grey points). E, F are interior vectors (blue points).



(b) **Step 1.**

The triangulation gives the 6 simplexes $T(\mathcal{X}_6) = \{\mathcal{S}_{ADE}, \mathcal{S}_{CDE}, \mathcal{S}_{CEF}, \mathcal{S}_{BCF}, \mathcal{S}_{AEF}, \mathcal{S}_{ABF}\}$ defined by the blue line segments.



(c) **Step 1.1.**

The vectors $\{\boldsymbol{s}^*_{ADE}, \boldsymbol{s}^*_{CDE}, \boldsymbol{s}^*_{CEF}, \boldsymbol{s}^*_{BCF}, \boldsymbol{s}^*_{AEF}, \boldsymbol{s}^*_{ABF}\}$ (green diamonds) that maximize the standard error of the predictions within each of the 6 simplexes are identified.

FIGURE B.1 **Illustrative example of ALGORITHM 2.**

BOX 2 CANDIDATE DESIGN POINT ALGORITHM (ALGORITHM 2)

For a given set $\mathcal{D}_n$ of design vectors, let $\mathcal{X}_n$ be the set of the corresponding input vectors, and $f^*(\cdot)$ an emulator fit with $\mathcal{D}_n$.

```
1. Obtain a triangulation $T(\mathcal{X}_n)$. For each simplex $\mathcal{S}_j$ in $T(\mathcal{X}_n)$:
   1.1 Find the input vector $\mathbf{s}_j^*$ that maximizes the standard error of the prediction $\hat{SE}(f^*(\mathbf{s}_j))$ for all $\mathbf{s}_j \in \mathcal{S}_j$.
2. Return the candidate input vectors $\mathcal{C} = \mathrm{unique}[\{\mathbf{s}_j^* \text{for all } j\}]$ from the input vectors identified in Step 1.
```