acebayes: An R Package for Bayesian Optimal Design of Experiments via Approximate Coordinate Exchange

Antony M. Overstall
University of Southampton

David C. Woods
University of Southampton

Maria Adamou
University of Southampton

Abstract

We describe the R package **acebayes** and demonstrate its use to find Bayesian optimal experimental designs. A decision-theoretic approach is adopted, with the optimal design maximising an expected utility. Finding Bayesian optimal designs for realistic problems is challenging, as the expected utility is typically intractable and the design space may be high-dimensional. The package implements the approximate coordinate exchange algorithm to optimise (an approximation to) the expected utility via a sequence of conditional one-dimensional optimisation steps. At each step, a Gaussian process regression model is used to approximate, and subsequently optimise, the expected utility as the function of a single design coordinate (the value taken by one controllable variable for one run of the experiment). In addition to functions for bespoke design problems with user-defined utility functions, **acebayes** provides functions tailored to finding designs for common generalised linear and nonlinear models. The package provides a step-change in the complexity of problems that can be addressed, enabling designs to be found for much larger numbers of variables and runs than previously possible. We provide tutorials on the application of the methodology for four illustrative examples of varying complexity where designs are found for the goals of parameter estimation, model selection and prediction. These examples demonstrate previously unseen functionality of **acebayes**.

Keywords: *A*-optimality, computer experiments, *D*-optimality, decision-theoretic design, Gaussian process regression, generalised linear models, high-dimensional design, model selection, nonlinear models, prediction, pseudo-Bayesian design.

1. Introduction

A well-planned and executed experiment is an efficient and effective way of learning the effect of an intervention on a process or system (Box *et al.* 2005), and design of experiments is a key contribution of statistics to the scientific method (Stigler 2016, ch. 6). Statistical design is an "a priori" activity, taking place before data is collected, and so fits naturally within a Bayesian framework. Before experimentation, current knowledge on models and parameters can be represented by prior probability distributions, and the experimental aim (e.g., parameter estimation, model selection, or prediction) can be incorporated into a decision-theoretic approach through the specification of a utility function (Berger 1985, ch. 2). A Bayesian optimal design is then found by maximising the expectation of this utility over the space of all possible designs, where expectation is with respect to the joint distribution of all unknown quantities including the, as yet, unobserved responses (Chaloner and Verdinelli 1995).

To formalise, suppose the aim of an experiment that varies k variables in n runs is to estimate or identify quantities $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)^\top$, which, for example, may be (functions of) parameters, model indicators or future responses. We perform this task using data $\boldsymbol{y} = (y_1, \ldots, y_n)^\top \in \mathcal{Y} \subset \mathbb{R}^n$ collected using a design $\boldsymbol{d} \in \mathcal{D} \subset \mathbb{R}^{n \times k}$, an $n \times k$ matrix with ith row $\boldsymbol{x}_i^\top = (x_{i1}, \ldots, x_{ik})$ holding the treatment, or combination of values of the controllable variables, assigned to the ith run of the experiment ($i = 1, \ldots, n$). We refer to nk as the dimensionality of the design and the x_{ij} as *coordinates* of the design ($j = 1, \ldots, k$).

A decision-theoretic Bayesian optimal design $\boldsymbol{d}^\star$ maximises the expected utility

$$\begin{aligned} U(\boldsymbol{d}) &= \mathrm{E}_{\boldsymbol{\gamma},\boldsymbol{y}|\boldsymbol{d}}\left[u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})\right] \\ &= \int u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) \pi(\boldsymbol{\gamma}, \boldsymbol{y}|\boldsymbol{d}) \, \mathrm{d}\boldsymbol{\gamma} \, \mathrm{d}\boldsymbol{y} \qquad (1) \\ &= \int u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) \pi(\boldsymbol{\gamma}|\boldsymbol{y}, \boldsymbol{d}) \pi(\boldsymbol{y}|\boldsymbol{d}) \, \mathrm{d}\boldsymbol{\gamma} \, \mathrm{d}\boldsymbol{y} \qquad (2) \\ &= \int u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) \pi(\boldsymbol{y}|\boldsymbol{\gamma}, \boldsymbol{d}) \pi(\boldsymbol{\gamma}|\boldsymbol{d}) \, \mathrm{d}\boldsymbol{\gamma} \, \mathrm{d}\boldsymbol{y} \,, \qquad (3) \end{aligned}$$

with utility function $u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})$ providing a measure of success of design $\boldsymbol{d}$ for quantities $\boldsymbol{\gamma}$ and data $\boldsymbol{y}$, and $\pi(\cdot|\cdot)$ denoting a conditional probability density or mass function. The density/mass $\pi(\boldsymbol{\gamma}|\boldsymbol{d})$ quantifies prior information about $\boldsymbol{\gamma}$ available before the experiment is run. The equivalence of Equations 2 and 3 follows from application of Bayes theorem; Equation 2 more clearly shows the dependency on the posterior distribution, whereas Equation 3 is often more useful for calculations and computation.

Although straightforward in principle, there are several hurdles to the practical evaluation and optimisation of Equations 2 or 3, highlighted in recent reviews by Ryan *et al.* (2016) and Woods *et al.* (2017).

1. The expected utility often results from an analytically intractable and, typically, high-dimensional integral. For such utilities, $U(\mathbf{d})$ may be approximated by a weighted sum

$$\tilde{U}(\boldsymbol{d}) = \sum_{b=1}^{B} w_b u(\boldsymbol{\gamma}_b, \boldsymbol{y}_b, \boldsymbol{d}) \,, \qquad (4)$$

for $w_b > 0$. For example, a Monte Carlo approximation would sample $\{\boldsymbol{\gamma}_b, \boldsymbol{y}_b\}_{b=1}^B$ from the joint distribution $\pi(\boldsymbol{\gamma}, \boldsymbol{y}|\boldsymbol{d})$ and set $w_b = 1/B$; maximisation of $\tilde{U}(\mathbf{d})$ would be a

stochastic optimisation problem. For simple utility functions not depending on data $\boldsymbol{y}$, a deterministic quadrature approximation may be applied, with $\boldsymbol{\gamma}_b$ and w_b being quadrature abscissae and weights, respectively.

2. The design space may be continuous and of high dimension.

3. The utility function itself may not be available in closed form, requiring an approximation $\tilde{u}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})$ to be substituted into Equation 4.

Most Bayesian optimal design methodology in the literature has been restricted to low-dimensional designs, i.e., small values of nk. See Müller and Parmigiani (1995), Müller (1999), Amzal *et al.* (2006), Long *et al.* (2013), and Drovandi *et al.* (2013, 2014), where the largest designs found had $nk = 4$. To find designs with larger n for $k = 1$ variable, Ryan *et al.* (2014) chose design points as quantiles of a parametric distribution. Although such an approach reduces the dimension of the optimisation problem (e.g., to finding the optimal values of a small number of parameters controlling the distribution), the original optimal design problem is not being directly addressed and hence usually sub-optimal designs will be found.

Overstall and Woods (2017) recently presented the first general methodology for finding high-dimensional Bayesian optimal designs using the approximate coordinate exchange (ACE) algorithm. As will be described in Section 2.1, the main feature of this approach is the combination of the low-dimensional smoothing methodology of Müller and Parmigiani (1995) with a coordinate exchange, or cyclic ascent, algorithm (Meyer and Nachtsheim 1995; Lange 2013, p. 171). In essence, the high-dimensional, computationally expensive and often stochastic optimisation problem is reduced to a sequence of one-dimensional, computationally cheaper, and deterministic optimisations. In this paper, we describe the R package **acebayes** (Overstall *et al.* 2018c) which implements the ACE algorithm, and introduce functionality that facilitates finding optimal designs for common classes of models, including nonlinear and generalised linear models. The package provides the first general-purpose software for finding fully-Bayesian optimal designs. In addition, the package implements methods to find pseudo-Bayesian designs (see Section 3.2) using coordinate exchange and quadrature approximations (see Gotwalt *et al.* 2009). The package has been demonstrated by finding Bayesian optimal designs for non-trivial statistical models, addressing design spaces with dimensionality approaching two orders of magnitude greater than existing methods.

There are only a few other R packages that attempt to find optimal designs, and none that tackle the general Bayesian design problem addressed by **acebayes**. The package **AlgDesign** (Wheeler 2014) implements exchange-type algorithms to find D-, A- and I-optimal designs for linear models. The package **OptimalDesign** (Harman and Filova 2016) tackles similar linear model design problems using various algorithms including integer quadratic programming. For nonlinear models, package **ICAOD** (Masoudi *et al.* 2018) can be used to find designs for quite general classes of models under "optimum-on-average" criteria, amongst others. Such criteria are mathematically equivalent to "pseudo-Bayesian" design criteria, see Section 3. This package uses the meta-heuristic imperialist competitive algorithm (Masoudi *et al.* 2017). For special classes of nonlinear models, locally optimal designs, with a point mass prior for $\boldsymbol{\gamma}$ and utility functions not depending on $\boldsymbol{y}$, can be found by packages **LDOD** (Masoudi *et al.* 2013), **designGLMM** (Bush and Ruggiero 2016) and **PopED** (Nyberg *et al.* 2012).

This paper is structured as follows. We briefly describe both the ACE algorithm and its implementation in **acebayes** in Section 2. Section 3 presents common utility functions em-

ployed in Bayesian design, and discusses their computational approximation. In Section 4 we demonstrate the use of various functions in the **acebayes** package to find optimal Bayesian designs for common nonlinear and generalised linear models, and bespoke model selection and prediction problems. We conclude in Section 5 with a short discussion.

2. Approximate coordinate exchange and acebayes

In this section we give a brief description of the ACE algorithm. Full details of the methodology can be found in Overstall and Woods (2017).

The algorithm has two phases, both of which are provided in full in Appendix A. In Phase I a smooth, and computationally inexpensive *emulator* for the approximation $\tilde{U}(\mathbf{d})$ in Equation 4 is maximised as a function of each design coordinate x_{ij} in turn, conditional on the values of the other $nk-1$ coordinates. In essence, the optimisation problem is solved via a sequence of computer experiments (see Santner *et al.* 2003).

For a stochastic approximation to the expected utility (e.g., Monte Carlo integration), the coordinate value that maximises each emulator is accepted with probability obtained from a Bayesian test of equality of the approximations under the proposed and current designs. For a deterministic approximation (e.g., quadrature), the design proposed by the emulator is accepted if the value of $\tilde{U}(\mathbf{d})$ for the proposed design is larger than for the current design.

As Phase I tends to produce clusters of design points, Phase II of the algorithm can be applied to attempt to amalgamate these clusters through use of a point exchange algorithm with a candidate set formed from the points in the final Phase I design.

2.1. ACE algorithm

Phase I of the algorithm (Appendix A.1) uses cyclic ascent to maximise approximation $\tilde{U}(\mathbf{d})$ to the expected utility. A one-dimensional emulator of $\tilde{U}(\mathbf{d})$ is built for each coordinate x_{ij} ($i = 1,\ldots,n$; $j = 1,\ldots,k$) in turn as the mean of the posterior predictive distribution conditioned on a small number of evaluations of $\tilde{U}(\mathbf{d})$ and assuming a Gaussian process (GP) prior (see Rasmussen and Williams 2006, ch. 2). For the ijth coordinate, an emulator is constructed by (i) selecting a one-dimensional space-filling design, $x_{ij}^1,\ldots,x_{ij}^Q$, with Q points; (ii) constructing the Q designs $\boldsymbol{d}_{ij}^q$, with the qth design having ith run $\boldsymbol{x}_i^q = (x_{i1},\ldots,x_{ij-1},x_{ij}^q,x_{ij+1},\ldots,x_{ik})^\top$ and all other runs equal to those from the current design; (iii) evaluating $\tilde{U}(\boldsymbol{d}_{ij}^q)$ for $q = 1,\ldots,Q$; and (iv) fitting a GP regression model to the "data" $\{x_{ij}^q,\tilde{U}(\boldsymbol{d}_{ij}^q)\}_{q=1}^Q$ and constructing an emulator $\hat{U}_{ij}(x)$ as the mean of the posterior predictive distribution. Maximisation of $\hat{U}_{ij}(x)$ to obtain $x_{ij}^\star$ is via evaluation of the emulator for a large discrete grid of values of x_{ij} to produce design $\boldsymbol{d}_{ij}^\star$ with ith row $(\boldsymbol{x}_i^\star)^\top = (x_{i1},\ldots,x_{ij-1},x_{ij}^\star,x_{ij+1},\ldots,x_{ik})$. Overstall and Woods (2017) found this approach to maximising $\hat{U}_{ij}(x)$ to be robust to multi-modal emulators and computationally efficient due to the negligible computational expense of evaluating the predictive mean.

If $\tilde{U}(\boldsymbol{d})$ is a Monte Carlo approximation, it is subject to two sources of potential errors: Monte Carlo error and emulator inadequacy. To separate these components and reduce the impact of a poor emulator, $\boldsymbol{d}_{ij}^\star$ is only accepted as the next design in the algorithm with probability $p^\star$ obtained from a Bayesian hypothesis test, independent of the GP emulator (see Step 2d in Appendix A.1). Here $p^\star$ is calculated as the posterior probability that the

expected utility for the proposed design $\boldsymbol{d}_{ij}^{\star}$ is greater than that for the current design, given independent Monte Carlo samples of the utility under each design and assuming normally distributed utility values. For cases where this latter assumption is violated, Overstall *et al.* (2018a) developed an alternative procedure derived from a one-sided test of a difference in proportions appropriate for use with, for example, a 0-1 utility function (see Equation 8 in Section 3.1). Larger Monte Carlo sample sizes are typically used for these tests than for the construction of the emulator to increase the precision of approximation $\tilde{U}(\mathbf{d})$. Both tests are implemented in **acebayes**.

For a deterministic $\tilde{U}(\boldsymbol{d})$, design $\boldsymbol{d}_{ij}^{\star}$ is accepted if its approximate expected utility, evaluated independently of the emulator, is greater than that of the current design.

Phase I can produce clusters of design points where, for example, design points $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ are separated by only a small Euclidean distance for some $i, i' = 1, \ldots, n$. Often, the design can be improved by consolidating such points into a single repeated design point (see also Gotwalt *et al.* 2009). Phase II of ACE (Appendix A.2) performs this consolidation step using a point exchange algorithm (e.g., Atkinson *et al.* 2007, ch. 12) with a candidate set given by the final design from Phase I. For Monte Carlo approximations to the expected utility, comparison of the approximate expected utility between two designs is again made on the basis of a Bayesian hypothesis test (see Step 6 of Appendix A.2).

In both phases, convergence is assessed informally using trace plots of the evaluations of approximate expected utility at each iteration. See Section 4.2 for an example of such a plot produced by **acebayes**.

Similar to all coordinate exchange algorithms (e.g., Goos and Jones 2011, pg. 36), ACE can be sensitive to the starting design. For this reason, it should be repeated from C different starting designs and `acebayes` provides methods to facilitate this repetition, potentially via simple parallel computing.

2.2. acebayes implementation of ACE

The main functions in the **acebayes** package are `ace` and `pace`, which implement both phases of the ACE algorithm and have mandatory and optional arguments as given in Tables 1 and 2, respectively. The `ace` function implements the ACE algorithm from a single starting design, whereas `pace` repeats ACE from C different starting designs. The argument `utility` gives the user complete flexibility to specify the design problem including the choice of statistical model, prior distribution, experimental aim and any necessary approximation to the utility function (see Section 3).

Much of the **acebayes** codebase is written in C++ and makes use of packages **Rcpp** (Eddelbuettel and Francois 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014). Space-filling designs to build the one-dimensional GP emulators are found using the R package **lhs** (Carnell 2016) which generates Latin hypercube samples.

To demonstrate the use of the `ace` and `pace` functions we use a simple Poisson response model for estimating a single parameter $\boldsymbol{\gamma} = (\theta)$. Consider an experiment where the ith run involves specifying the $k = 1$ variable $x_i \in [-1, 1]$ and measuring the count response y_i. Assume the following model;

$$y_i \sim \text{Poisson}(\mu_i),$$

independently, for $i = 1, \ldots, n$, with $\mu_i = \exp(x_i\theta)$. We assume a priori that $\theta \sim \text{N}(0, 1)$. We

Argument	Description
`utility`	A function with two arguments: `d` and `B`. For a Monte Carlo approximation (`deterministic = FALSE`), it should return a **vector** of length B where each element gives an evaluation of the (approximate) utility function $\tilde{u}(\boldsymbol{\gamma}_b, \boldsymbol{y}_b, \boldsymbol{d})$ for design `d` for each pair $(\boldsymbol{\gamma}_b, \boldsymbol{y}_b)$ generated from the joint distribution of $\boldsymbol{\gamma}$ and $\boldsymbol{y}$ for $b = 1, \ldots, B$. For a deterministic approximation (`deterministic = TRUE`), it should return a **scalar** giving the approximate value of the expected utility for design `d`. In this latter case, the argument `B` can be a list containing tuning parameters for the deterministic approximation. If `B` is not required, the utility function must still accept the argument, e.g., using the `...` notation.
`start.d`	For `ace`: an $n \times k$ matrix specifying the starting design for Phase I (see Step 1 in Appendix A.1). For `pace`: a list of C different starting designs.

Table 1: Mandatory arguments to the `ace` and `pace` functions.

find a design that maximises the expectation of the following utility function

$$u(\theta, \boldsymbol{y}, \boldsymbol{d}) = u(\theta, \boldsymbol{d}) = \mathcal{I}(\theta;\, \boldsymbol{d})\,,$$

where

$$\mathcal{I}(\theta;\, \boldsymbol{d}) = \sum_{i=1}^{n} x_i^2 \exp\left(\theta x_i\right)$$

is the Fisher information and $\boldsymbol{d} = (x_1, \ldots, x_n)^\top$. As u does not depend on $\boldsymbol{y}$, the expected utility reduces to

$$U(\boldsymbol{d}) = \int u(\theta, \boldsymbol{d}) \pi(\theta)\, \mathrm{d}\theta\,, \tag{5}$$

where $\pi(\theta)$ is the density of the standard normal distribution. It is straightforward to show that

$$U(\boldsymbol{d}) = \sum_{i=1}^{n} x_i^2 \exp\left(x_i^2/2\right),$$

and the optimal design is $\boldsymbol{d}^* = (\pm 1, \ldots, \pm 1)^\top$. However to demonstrate the use of the `ace` and `pace` functions, we employ a Monte Carlo approximation to $U(\boldsymbol{d})$ in Equation 5. This approximation is implemented in the R function below which takes two arguments: an $n \times 1$ matrix $\boldsymbol{d}$ and the Monte Carlo sample size B. It returns a vector of length B, where each element is an evaluation of $u(\theta, \boldsymbol{d})$ for a value of θ generated from the prior distribution.

```
R> utilfisher <- function(d, B) {
+   theta <- rnorm(B)
+   ui <- matrix(rep(d[, 1] ^ 2, B), ncol = B) * exp(outer(d[, 1], theta))
+   apply(ui, 2, sum)
+   }
```

Argument	Description
`B`	For a Monte Carlo approximation (`deterministic = FALSE`), a vector of length two specifying the size of the Monte Carlo samples generated from the joint distribution of unknown quantities and unobserved responses, to use when approximating the expected utility via Equation 4. The first element specifies the sample size to use in the comparison procedures (see Steps 2d and 6 in Appendices A.1 and A.2, respectively). The second element specifies the sample size to use for the evaluations of Monte Carlo integration that are used to fit the Gaussian process emulator (see Step 2b in Appendix A.1). If missing when `deterministic = FALSE`, the default value is `c(20000,1000)`. For a deterministic approximation (`deterministic = TRUE`), `B` may be a list of length two containing any necessary tuning parameters for the utility calculations for the comparison and emulation steps.
`Q`	The number, Q, of evaluations of the approximation to the expected utility function (1) used to construct the GP emulator. The default is `Q = 20`.
`N1`	The number, N_1, of iterations of Phase I. The default is `N1 = 20`.
`N2`	The number, N_2, of iterations of Phase II. The default is `N2 = 100`.
`lower`	Lower limits on the design space. It can be a scalar (all elements have the same lower limit) or an $n \times k$ matrix so that all elements can have unique lower limits. The default is `lower = -1`.
`upper`	Upper limits on the design space; see `lower`. The default is `upper = 1`.
`limits`	A function that can be used to define complex constraints on the design space. The default is `limits = NULL`, i.e., there are no constraints.
`progress`	For `ace` only. A Boolean indicating whether progress of the ACE algorithm is printed. The default is `progress = FALSE`.
`binary`	A Boolean indicating whether the Bayesian two sample t-test (`FALSE`; the default) or the test of proportions (`TRUE`) is carried out.
`deterministic`	A logical argument indicating use of a Monte Carlo (`FALSE`, default) or deterministic (`TRUE`) approximation to the expected utility.
`mc.cores`	For `pace` only. The number of cores to use, i.e. at most how many child processes will be run simultaneously. The default is `mc.cores = 1`.
`n.assess`	For `pace` only. If `deterministic = FALSE`, the approximate expected utility for the C final designs will be calculated as the mean of `n.assess` approximations to the expected utility for each design.

Table 2: Optional arguments to the `ace` and `pace` functions.

We now call the function `ace` to demonstrate finding a design with $n = 12$ runs from a single starting design. The first mandatory argument `utility` is set to be the function defined above. The second mandatory argument `start.d` specifies the starting design; note that although $k = 1$, the starting design still needs to be an `R matrix` object. Here we use a matrix of n zeros. We keep all other arguments as their default values and set a seed for reproducibility.

```
R> set.seed(1)
R> n <- 12
R> start.d <- matrix(0, nrow = n, ncol = 1)
R> ex22a <- ace(utility = utilfisher, start.d = start.d)
```

Printing the resulting `"ace"` object summarises the inputs and the computing resources required.

```
R> ex22a

User-defined model & utility
Number of runs = 12

Number of factors = 1

Number of Phase I iterations = 20

Number of Phase II iterations = 100

Computer time = 00:00:33
```

An `"ace"` object is a list which includes the final designs from Phase 1 (`phase1.d`) and Phase 2 (`phase2.d`) of the algorithm. If `N1 = 0` (i.e., there are no Phase I iterations), then `phase1.d` will be equal to the argument `start.d`. Correspondingly, if `N2 = 0` (i.e., there are no Phase II iterations), then `phase2.d` will be equal to `phase1.d`.

Consider now repeating the above implementation of ACE from $C = 10$ different randomly generated starting designs, where each element in the design is generated uniformly from $[-1, 1]$. The starting designs are organised into a list and we then call the function `pace` with the same `utility` argument as the call to `ace` above.

```
R> C <- 10
R> start.d <- list()
R> for(i in 1:C){
+ start.d[[i]] <- matrix(runif(n = n, min = -1, max = 1), ncol = 1)
+ }
R>
R> ex22b <- pace(utility = utilfisher, start.d = start.d)
```

Printing the resulting `pace` object summarises the inputs and the computing resources required.

```
R> ex22b

User-defined model & utility
Number of repetitions = 10

Number of runs = 12

Number of factors = 1

Number of Phase I iterations = 20

Number of Phase II iterations = 100

Computer time = 00:05:41
```

A `pace` object is a list which includes the Phase II design from each repetition (`final.d`) and, from those, the design (`d`) found with the largest approximate expected utility.

To compare two designs, `acebayes` provides the the `S3` method `assess`. It takes two mandatory arguments: `d1` and `d2`, specifying the two designs to be compared. The argument `d1` should be either a `ace` or `pace` object and the two designs will be compared on the basis of the expected utility used for `d1`. The argument `d2` should either be a `ace`, `pace` or `matrix` object. We use `assess` to compare the single starting design of n zeros, to the final designs from a single repetition of ACE and from C repetitions. In cases like this, where the approximation to the expected utility is not deterministic, `assess` will calculate `n.assess` approximations to the expected utility where `n.assess` is an optional argument. We set `n.assess` to be 100. The function `assess` will return an object of type `assess`. For a non-deterministic approximation to the expected utility, when printed, an `assess` object will show the mean and standard deviation of the `n.assess` evaluations of the approximate expected utility for each design.

```
R> assess(d1 = ex22a, d2 = matrix(rep(0, n), ncol = 1), n.assess = 100)

Mean (sd) approximate expected utility of d1 = 19.78256 (0.1546731)
Mean (sd) approximate expected utility of d2 = 0 (0)

R> assess(d1 = ex22b, d2 = ex22a, n.assess = 100)

Mean (sd) approximate expected utility of d1 = 19.78025 (0.1106841)
Mean (sd) approximate expected utility of d2 = 19.79534 (0.1633979)
```

We can clearly see the improvement in approximate expected utility from the design of n zeros and the designs found by ACE. It appears that, in this case, the C repetitions of ACE have not led to any improvement in the design performance. This can be seen by plotting the `assess` object providing side-by-side boxplots of the `n.assess` evaluations of the approximate expected utility for each design.

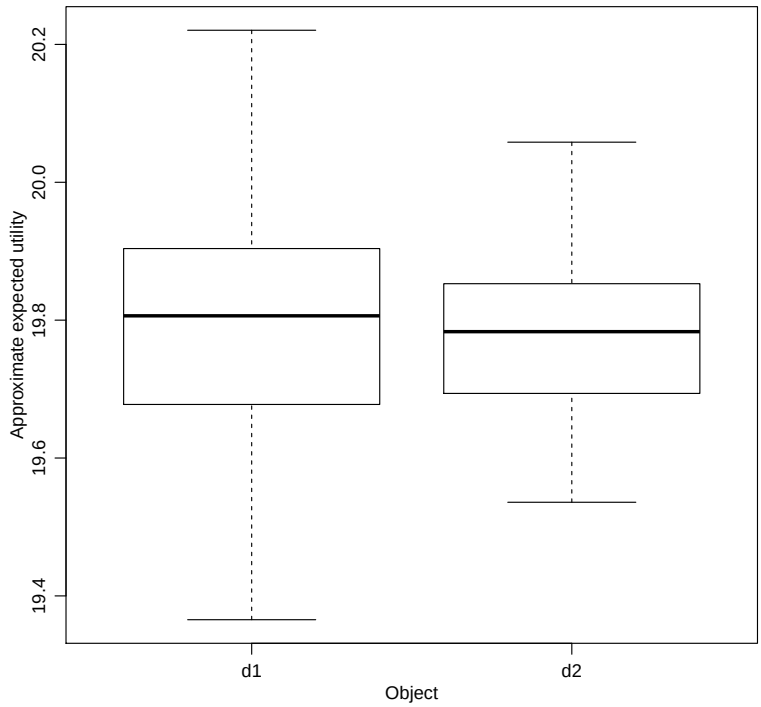

Figure 1: Boxplots of the evaluations of the approximate expected utility for the designs found from one (extttd1) and $C = 10$ (extttd2) repetitions of ACE.

```
R> assess22 <- assess(d1 = ex22a, d2 = ex22b, n.assess = 100)
R> plot(assess22)
```

The resulting plot is shown in Figure 1 from which we can see that the performance of the designs are very close. This can be confirmed by inspecting the two designs and noting that they both consist only of values $x \pm 1$, i.e. both are optimal.

```
R> t(ex22a$phase2.d)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]   -1   -1   -1    1   -1   -1   -1   -1   -1    -1    -1    -1
```

```
R> t(ex22b$d)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]   -1   -1   -1   -1   -1   -1   -1    1    1     1    -1     1
```

3. Utility functions and approximations

Prior to application of the ACE algorithm, a relevant, and perhaps pragmatic, choice of utility function must be made that encapsulates the aim of the experiment. In Section 4, we illustrate use of functions from **acebayes** by finding efficient designs under three common utility functions.

3.1. Common utility functions

1. **Shannon information gain** (SIG; Lindley 1956):

$$\begin{aligned} u_{SIG}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) &= \log \pi(\boldsymbol{\gamma}|\boldsymbol{y}, \boldsymbol{d}) - \log \pi(\boldsymbol{\gamma}|\boldsymbol{d}) \\ &= \log \pi(\boldsymbol{y}|\boldsymbol{\gamma}, \boldsymbol{d}) - \log \pi(\boldsymbol{y}|\boldsymbol{d}) \,. \end{aligned} \tag{6}$$

   A SIG-optimal design that maximises the expectation of u_{SIG} equivalently maximises the expected Kullback-Liebler divergence between the prior and posterior distributions (Chaloner and Verdinelli 1995).

2. **Negative squared error loss** (NSEL; e.g., Chaloner 1984):

$$u_{NSEL}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) = -\left[\boldsymbol{\gamma} - \mathrm{E}(\boldsymbol{\gamma}|\boldsymbol{y}, \boldsymbol{d})\right]^{\top} \left[\boldsymbol{\gamma} - \mathrm{E}(\boldsymbol{\gamma}|\boldsymbol{y}, \boldsymbol{d})\right] . \tag{7}$$

   A NSEL-optimal design that maximises the expectation of u_{NSEL} equivalently minimises the expected trace of the posterior variance matrix. Note that the use of this utility is not appropriate for nominal or ordinal $\boldsymbol{\gamma}$ (for example, if $\boldsymbol{\gamma}$ holds binary model indicator variables).

3. **0-1 utility** (e.g., Felsenstein 1992):

$$u_{01}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d}) = \prod_{l=1}^{p} I\left(M_l(\boldsymbol{y}, \boldsymbol{d}) - \delta_l < \gamma_l < M_l(\boldsymbol{y}, \boldsymbol{d}) + \delta_l\right) , \tag{8}$$

where $M_l(\boldsymbol{y}, \boldsymbol{d}) = \arg\max_{\gamma_l} \pi(\gamma_l | \boldsymbol{y}, \boldsymbol{d})$ is the marginal posterior mode of γ_l, $I(A)$ is the indicator function for event A, and $\delta_l \geq 0$ is a specified tolerance. This utility is only non-zero if the posterior mode is "close" to γ_l for all $l = 1, \ldots, p$. Setting $\delta_l = 0$ is typically only appropriate for discrete γ_l.

3.2. Approximating utility functions

Most utility functions, including those in Section 3.1, require approximation of posterior quantities, for example the marginal likelihood, $\pi(\boldsymbol{y}|\mathbf{d})$, or posterior mean, $\mathrm{E}(\boldsymbol{\gamma}|\boldsymbol{y}, \boldsymbol{d})$. Such quantities are analytically intractable for most models. Here we review those methods implemented in **acebayes** to produce approximate utilities $\tilde{u}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})$.

Overstall and Woods (2017) used Monte Carlo approximations, with a sample $\{\boldsymbol{\gamma}_b\}_{b=1}^{\tilde{B}}$ from $\pi(\boldsymbol{\gamma}|\boldsymbol{d})$, to approximate u_{SIG} and u_{NSEL} (in Equations 6 and 7, respectively) to enable design selection for parameter estimation, i.e., with $\boldsymbol{\gamma} = \boldsymbol{\theta}$, for a single model. When combined with a Monte Carlo approximation to the expected utility with sample size B, the resulting **nested Monte Carlo** (or double-loop Monte Carlo) approximation to $U(\boldsymbol{d})$ is subject to bias of order $\tilde{B}^{-1}$ (see Ryan 2003). Hence, large values of both B and $\tilde{B}$ are required to achieve suitable precision for design comparison and neglible bias, resulting in computationally expensive utility approximations.

Although the use of the one-dimensional emulators $\hat{U}_{ij}(x)$ helps to alleviate the computational cost associated with a nested Monte Carlo approximation, the adoption of alternative, cheaper, utility approximations can further increase the range and size of design problems that can be addressed. Several classes of analytical approximations have been proposed using normal approximations to the posterior distribution. Overstall *et al.* (2018a) applied ACE with a normal approximation to the posterior distribution with mean equal to the posterior mode and variance-covariance matrix equal to the inverse of the expected Fisher information, $\mathcal{I}(\boldsymbol{\theta}; \boldsymbol{d})$, minus the second derivative of the log prior density, both evaluated at the posterior mode. Such an approximation can lead to analytically tractable, if still potentially biased, **normal-based** approximations $\tilde{u}(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})$; for example, via a Laplace approximation to the marginal likelihood (see also Long *et al.* 2013).

Simpler approximations to some utilities can be obtained by using $\mathcal{I}(\boldsymbol{\theta}; \boldsymbol{d})^{-1}$ as an approximation to the posterior variance-covariance matrix (e.g., Chaloner and Verdinelli 1995). For example, for estimation of $\boldsymbol{\gamma} = \boldsymbol{\theta}$, approximations to the SIG and NSEL utility functions are given by

$$\tilde{u}_{SIGD}(\boldsymbol{\theta}, \boldsymbol{y}, \boldsymbol{d}) = \log\left|\mathcal{I}(\boldsymbol{\theta}; \boldsymbol{d})\right| , \tag{9}$$

$$\tilde{u}_{NSELA}(\boldsymbol{\theta}, \boldsymbol{y}, \boldsymbol{d}) = -\mathrm{tr}\left\{\mathcal{I}(\boldsymbol{\theta}; \boldsymbol{d})^{-1}\right\} . \tag{10}$$

Designs that maximise the expectation of $\tilde{u}_{SIGD}$ and $\tilde{u}_{NSELA}$ with respect to the prior distribution of $\boldsymbol{\theta}$ are referred to as **pseudo-Bayesian** D- and A-optimal, respectively.

3.3. Approximating the expected utility

The **acebayes** package uses expected utility approximations of the form given in Equation 4. For the nested Monte Carlo and normal-based approximations, $U(\boldsymbol{d})$ is approximated by the sample mean of $\tilde{u}(\boldsymbol{\gamma}_b, \boldsymbol{y}_b, \boldsymbol{d})$ for a sample $\{\boldsymbol{\gamma}_b, \boldsymbol{y}_b\}_{b=1}^{B}$ from $\pi(\boldsymbol{\gamma}, \boldsymbol{y}|\boldsymbol{d})$. Default values of B (and $\tilde{B}$ for nested Monte Carlo) are $B = 1000$ for constructing the one dimensional emulators $\hat{U}_{ij}(x)$ and $B = 20,000$ when calculating the probability of accepting the proposed design (see Section 2.1).

For pseudo-Bayesian D- and A-optimal design, where the approximations given by Equations 9 and 10 do not depend on $\boldsymbol{y}$, the p-dimensional integrals with respect to $\boldsymbol{\gamma} = \boldsymbol{\theta}$ can be approximated using quadrature methods. The **acebayes** package implements a radial-spherical integration rule (Monahan and Genz 1997), with $\boldsymbol{\gamma}_b$ and ω_b in Equation 4 being abscissas and (non-constant) weights, respectively. For small p, the value of B is typically of the order of several hundred, making this approach much less computationally intensive than either nested Monte Carlo or normal-based methods. Both multivariate normal and independent uniform prior densities are implemented for use with quadrature approximations in **acebayes**. See Gotwalt *et al.* (2009) for more details on using this quadrature scheme to find pseudo-Bayesian D-optimal designs.

4. Examples

In this section, we demonstrate the use of the **acebayes** package to find Bayesian optimal designs for four examples. Despite ACE being able to find efficient designs for larger and more complex problems than existing methods in the literature, it still requires significant computational resources. Hence we have chosen examples that illustrate the main features of the methodology and package but that do not require excessive computer time to complete. It should be clear how the examples can be extended to address more complex or realistic scenarios. In particular, the main arguments to the functions `ace` and `pace` are essentially identical. Therefore, to minimise the computer time taken to reproduce the examples, we only demonstrate the `pace` functionality in Sections 4.1 and 4.2 (in addition to that already demonstrated in Section 2.2).

A particular feature of this section is demonstration of the functions `aceglm` and `acenlm`, which simplify the process of finding Bayesian optimal designs for generalised linear models and nonlinear models, respectively. The `ace` function allows designs to be sought for very general problems. This flexibility comes at the price that non-expert users may feel uncomfortable with the level of additional coding required to use the function. To remove this potential barrier to the use of the package, `aceglm` and `acenlm` provide wrappers to `ace` that implement the ACE algorithm for these common model types. In both cases, the functions allow designs to be found for parameter estimation under a single model for a range of (approximated) utility functions. There are also corresponding functions, `paceglm` and `pacenlm`, which implement repetitions of ACE for generalised linear models and nonlinear models, respectively. Both `(p)aceglm` and `(p)acenlm` make use of the familiar `formula` and `family` R arguments and objects. In Sections 4.1 and 4.2 we demonstrate, in detail, the use of these functions.

In each case, unless otherwise stated, we use a randomly generated Latin hypercube space-filling design (see Santner *et al.* 2003, ch. 5) as the starting design for the ACE algorithm. These designs are generated using the `randomLHS` function in the **lhs** package (Carnell 2016).

Each element of the starting design is scaled to lie in the stated design space. Additionally, before we generate such a design, we set a random seed for full reproducibility of the results in this section.

4.1. Compartmental non-linear model

In this example, we demonstrate using `acenlm` to generate a pseudo-Bayesian D-optimal design for a compartmental model commonly used in pharmacokinetics (PK). The `acenlm` and `pacenlm` functions can find optimal designs for models of the form

$$y_i \sim \mathrm{N}\left(\mu(\boldsymbol{\theta};\, \boldsymbol{x}_i), \sigma^2\right) , \tag{11}$$

where $y_1, \ldots, y_n$ are assumed independently distributed and the user specifies a non-linear function, $\mu(\boldsymbol{\theta};\, \boldsymbol{x}_i)$, of parameters $\boldsymbol{\theta}$, and prior distributions for both $\boldsymbol{\theta}$ and $\sigma^2 > 0$, the unknown response variance.

A PK experiment typically involves introducing a fixed amount of drug to the body at time zero and measuring at times $t_1, \ldots, t_n$ the amount of drug remaining in the body. Hence the design consists of the n sampling times, i.e., $\boldsymbol{x}_i = (t_i)$, $k = 1$ and $\boldsymbol{d} = (t_1, \ldots, t_n)^\top$. Here we assume a sampling interval such that $t_i \in [0, 24]$ hours. We illustrate `(p)acenlm` on the compartmental model

$$\mu(\boldsymbol{\theta};\, t_i) = \theta_3 \left[\exp\left(-\theta_1 t_i\right) - \exp\left(-\theta_2 t_i\right)\right] ,$$

with $p = 3$ unknown parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^\top$.

For non-linear models of the form of Equation 11, the Fisher information for $\boldsymbol{\theta}$ is

$$\mathcal{I}(\boldsymbol{\theta};\, \boldsymbol{d}) = \frac{1}{\sigma^2} \sum_{i=1}^{n} \frac{\partial \mu(\boldsymbol{\theta};\, t_i)}{\partial \boldsymbol{\theta}} \frac{\partial \mu(\boldsymbol{\theta};\, t_i)}{\partial \boldsymbol{\theta}^\top}.$$

Following Gotwalt *et al.* (2009), we find designs with $n = 18$ sampling times and assume that elements of $\boldsymbol{\theta}$ have the following independent prior distributions:

$$\theta_1 \sim \mathrm{U}\left[0.01884, 0.9884\right] , \quad \theta_2 \sim \mathrm{U}\left[0.298, 8.298\right] , \tag{12}$$

with θ_3 having a prior point mass at 21.8.

To find a pseudo-Bayesian D-optimal design using `acenlm` we first specify one starting design having a single column named `"t"` and with elements scaled to the sampling interval $[0, 24]$.

```
R> set.seed(1)
R> n <- 18
R> k <- 1
R> p <- 3
R> start.d <- randomLHS(n = n, k = k) * 24
R> colnames(start.d) <- c("t")
```

We use quadrature to approximate the expected utility, and hence define the `prior` below as a list containing a single matrix `support`, where the rows specify the lower and upper limits of the uniform prior distribution for each parameter, respectively; see Equation 12. For use with the `acenlm` function, the columns of this matrix should be named. As $\mathcal{I}(\boldsymbol{\theta};\, \boldsymbol{d})$ only depends linearly on $1/\sigma^2$, the relative performance of designs under pseudo-Bayesian criteria do not depend on the unknown σ^2. Hence it is not required to specify a prior distribution for σ^2.

```
R> a1 <- c(0.01884, 0.298)
R> a2 <- c(0.09884, 8.298)
R> prior <- list(support = cbind(rbind(a1, a2), c(21.8, 21.8)))
R> colnames(prior[[1]]) <- c("theta1", "theta2", "theta3")
```

The `acenlm` function takes three mandatory arguments. The argument `formula` gives a symbolic description of the non-linear model (in this example, `~ theta3 * (exp( - theta1 * t) - exp( - theta2 * t))`). The argument `start.d` specifies the starting design, an $n \times k$ matrix with columns named as per the terms in the `formula` argument; and `prior` specifies the prior distribution for $\boldsymbol{\theta}$, with the input for this argument depending on the chosen `method` (see below). In addition, we set the lower and upper limits of each element of the design space to be 0 and 24 respectively.

```
R> ex411 <- acenlm(formula = ~ theta3 * (exp( - theta1 * t) -
+   exp( - theta2 * t)), start.d = start.d, prior = prior, lower = 0,
+   upper = 24)
```

For demonstration purposes, we compare the designs found from Phase I and II using the `S3` method `assess` (as introduced in Section 2.2). In cases of a deterministic approximation to the expected utility, `assess` will calculate one approximation to the expected utility for each of `d1` and `d2`. Furthermore, when `d1` is as a result of a call to `acenlm` or `aceglm` with the `criterion` argument being `"D"` or `"A"`, then `assess` will also approximate, again using quadrature, the pseudo-Bayesian relative D- or A-efficiency. The pseudo-Bayesian relative D-efficiency of design $\boldsymbol{d}_1$ relative to design $\boldsymbol{d}_2$, defined as

$$\text{Deff}(\boldsymbol{d}_1, \boldsymbol{d}_2) = 100 \times \exp\left\{\left[U_D(\boldsymbol{d}_1) - U_D(\boldsymbol{d}_2)\right]/p\right\},$$

provides a quantitative comparison of two designs. Here,

$$U_D(\boldsymbol{d}) = \int \log\left|\mathcal{I}(\boldsymbol{\theta};\, \boldsymbol{d})\right| \pi(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}\,.$$

Similar relative efficiency exists for pseudo-Bayesian A-optimal designs. The function `assess` calculates these efficiencies for pseudo-Bayesian D and A-optimal designs.

```
R> assess(d1 = ex411, d2 = ex411$phase1.d)
```

```
Approximate expected utility of d1 = 15.79695
Approximate expected utility of d2 = 15.70753
Approximate relative D-efficiency = 103.0255%
```

Here we see that Phase II led to a small increase in the expected utility. We can also compare the two designs in terms of the number of unique sampling times.

```
R> length(unique(ex411$phase1.d))
```

```
[1] 18
```

```
R> length(unique(ex411$phase2.d))

[1] 13
```

Therefore Phase II consolidated the design into 13 unique sampling times.

It is quite common in PK, and similar, experiments for there to be constraints on the minimum time between successive measurements. For such experiments, the ordered sampling times, $t_{(1)}, \ldots, t_{(n)}$, should satisfy the following constraint:

$$\min_{i=1,\ldots,n-1} |t_{(i)} - t_{(i+1)}| > c\,. \tag{13}$$

See Ryan *et al.* (2014) and Overstall and Woods (2017) for examples with similar constraints on the design for the compartmental model.

We can include such constraints in Phase I of the ACE algorithm using the `limits` argument. In Phase I, the candidate design is chosen by replacing the current coordinate (i.e., x_{ij}, $i = 1, \ldots, n$; $j = 1, \ldots, k$) by the value that maximises the predictive mean of the Gaussian process emulator. As discussed in Section 2.1, this maximisation is achieved by choosing the point with largest predicted mean from a grid of points, typically on an interval defined by the arguments `lower` and `upper`. We can also specify the argument `limits` as a function to incorporate (multivariate and dynamic) constraints on the design coordinates. The function should have three arguments: `d`, `i` and `j`. Here `d` specifies the design, and `i` and `j` define the current coordinate by specifying the row and column of `d`. The function should return a grid of points (in the form of a vector) from which the point with the largest predicted mean will be chosen.

The code below defines a `limits` function to incorporate the constraint given by Equation 13, with $c = 0.25$ (i.e., 15 minute intervals between sampling). A grid of 10,000 points from `lower = 0` to `upper = 24` is created. We then remove all points from this grid that are within 15 minutes of the sampling times in the current design excluding the ith point. Note that here the function does not depend on `j` as the design involves only $k = 1$ factor.

```
R> limits <- function(d, i, j) {
+   grid<-seq(from = 0, to = 24, length.out = 10000)
+   for(s in as.vector(d)[-i]) {
+     grid <- grid[(grid < (s - 0.25)) | (grid > (s + 0.25))]
+   }
+   grid
+ }
```

We find a design satisfying the constraint by including the specification of the `limits` argument in the `acenlm` function and setting the number of Phase II iterations to $N_2 = 0$ (as we do not want to consolidate clusters into repeated sampling times; see the constraint given by Equation 13).

```
R> ex412a <- acenlm(formula = ~ theta3 * (exp( - theta1 * t) -
+   exp( - theta2 * t)), start.d = start.d, prior = prior, lower = 0,
+   upper = 24, limits = limits, N2 = 0)
```

As described in Section 2.1, we should actually repeat ACE from C different starting designs. This can be achieved using the `pacenlm` function which takes the same arguments as `acenlm`. The only exception is that the argument `start.d` should be a list where each element is an $n \times k$ matrix. Below we specify such a list with $C = 10$.

```
R> C <- 10
R> start.d <- list()
R> for(i in 1:C){
+ start.d[[i]] <- randomLHS(n = n, k = k) * 24
+ colnames(start.d[[i]]) <- c("t")
+ }
R> ex412b <- pacenlm(formula = ~ theta3 * (exp( - theta1 * t) -
+   exp( - theta2 * t)), start.d = start.d, prior = prior, lower = 0,
+   upper = 24, limits = limits, N2 = 0)
```

We compare approximations to the expected utility for the design found from one repetition of ACE against the design found under $C = 10$ repetitions.

```
R> assess(d1 = ex412a, d2 = ex412b)

Approximate expected utility of d1 = 15.34813
Approximate expected utility of d2 = 15.36236
Approximate relative D-efficiency = 99.52679%
```

The design found under one repetition (`ex412a$phase2.d`) is about 99.5% D-efficient compared to the design found under $C = 10$ repetitions. In this case, the procedure was relatively robust to the starting design.

4.2. Logistic regression

In this section we consider a logistic regression model from Overstall and Woods (2017) to demonstrate the use of the `aceglm` and `paceglm` functions. We find designs that maximise the expected NSEL utility for estimation of parameters $\boldsymbol{\gamma} = \boldsymbol{\theta}$ using two different approximations to the utility function: 1) approximation given by Equation 10, resulting in a pseudo-Bayesian A-optimal design; and 2) the normal-based approximation of Overstall *et al.* (2018a).

A binary response is assumed to depend on $k = 4$ variables through the model

$$y_i \sim \text{Bernoulli}(\rho_i)\,,$$

for $i = 1, \ldots, n$ with $\rho_i = 1/[1 + \exp(-\eta_i)]$,

$$\eta_i = \theta_0 + \sum_{j=1}^{4} \theta_j x_{ij}\,,$$

and $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_2, \theta_4)^\top$ being the $p = 5$ unknown parameters that require estimation. We find designs with $n = 6$ runs, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{i4})^\top$, assuming $-1 \leq x_{ij} \leq 1$ $(i = 1, \ldots, 6; j = 1, \ldots, 4)$. The dimension of the design space is $nk = 24$.

Independent uniform prior distributions for each element of $\boldsymbol{\theta}$ are assumed,

$$\begin{array}{lll} \theta_0 \sim \mathrm{U}[-3, 3]\,, & \theta_1 \sim \mathrm{U}[4, 10]\,, & \theta_2 \sim \mathrm{U}[5, 11]\,, \\ \theta_3 \sim \mathrm{U}[-6, 0]\,, & \theta_4 \sim \mathrm{U}[-2.5, 3.5]\,. & \end{array} \tag{14}$$

The `aceglm` function has four mandatory arguments. The arguments `formula` and `family` are the well-known arguments we would supply to the `glm` function in the **stats** package (R Core Team 2018) and are used to define the logistic regression model, i.e., `formula = ~ x1 + x2 + x3 + x4` and `family = binomial`. The argument `start.d` specifies the starting design, an $n \times k$ matrix with columns named as per the terms in the `formula` argument; and `prior` specifies the prior distribution for $\boldsymbol{\theta}$, with the input for this argument depending on the chosen `method` (see below). Initially, we specify values for n, p and k, and generate a list of $C = 10$ starting designs.

```
R> set.seed(1)
R> n <- 6
R> p <- 5
R> k <- 4
R> C <- 10
R> start.d <- list()
R> for(i in 1:C){
+ start.d[[i]] <- randomLHS(n = n, k = k) * 2 - 1
+ colnames(start.d[[i]]) <- c("x1", "x2", "x3", "x4")
+ }
```

Pseudo-Bayesian A-optimal design

We start by finding a pseudo-Bayesian A-optimal design by specifying `criterion = "A"`. Note that the default value is `criterion = "D"` which would result in a pseudo-Bayesian D-optimal design using the approximation given in Equation 9. To approximate the expected utility we use the radial-spherical quadrature rule discussed in Section 3.3. This can be specified by setting the `method` argument in `paceglm` to `"quadrature"`, which is the default for pseudo-Bayesian criteria (i.e., `criterion` being `"A"`, `"D"` or `"E"`). Under this `method`, to specify the uniform prior distribution given by Equation 14, we define a list with a single matrix argument `support` with the first row specifying the lower limits for the prior for each parameter, and the second row specifying the upper limits. The other optional arguments to `paceglm` are shared with `ace` (see Table 2). The exception is the argument `deterministic`, which is not required for `(p)aceglm` and `(p)acenlm` as the `method` argument specifies if the approximation to the expected utility is deterministic. We leave these optional arguments set to default values but demonstrate their use in later examples.

```
R> a1 <- c(-3, 4, 5, -6, -2.5)
R> a2 <- c(3, 10, 11, 0, 3.5)
R> prior <- list(support = rbind(a1, a2))
R> ex411 <- paceglm(formula = ~ x1 + x2 + x3 + x4, family = binomial,
+   start.d = start.d, prior = prior, criterion = "A")
```

For demonstration purposes, using the `assess` function, we compare the design found with the highest expected utility from $C = 10$ repetitions (i.e. `ex411$d`) and the design found under the first repetition (i.e. `ex411$final.d[[1]]`).

```
R> assess(d1 = ex411, d2 = ex411$final.d[[1]])

Approximate expected utility of d1 = -225.6464
Approximate expected utility of d2 = -267.3872
Approximate relative A-efficiency = 118.4983%
```

This clearly demonstrates the advantage of repeating ACE from different starting designs.

Normal-based approximation to the NSEL utility function

To apply the normal-based approximation to the NSEL utility function we let `criterion = "NSEL-Norm"` in the `paceglm` function. This changes the default `method` argument to `"MC"` implementing Monte Carlo. Under this method, we require an R function that generates a sample from the prior distribution given by Equation 14.

```
R> prior <- function(B) {
+   theta <- matrix(0, nrow = B, ncol = p)
+   for(b in 1:B) {
+     theta[b, ] <- runif(n = p, min = a1, max = a2)
+     }
+   theta
+ }
```

Again, we set all other optional arguments to their default values.

```
R> ex412 <- paceglm(formula = ~ x1 + x2 + x3 + x4, family = binomial,
+   start.d = start.d, prior = prior, criterion = "NSEL-Norm")
```

We can check the approximate convergence of the ACE algorithm using the `S3` method `plot.ace`.

```
R> plot(ex412)
```

This function produces a trace plot (see Figure 2) of the Monte Carlo approximation to the expected utility (1) against iteration number for Phases I and II and the design that had the highest expected utility from the C repetitions. In Phase I, the algorithm makes very large initial improvements to the approximate expected utility, and appears to have converged after six or seven iterations. Phase II does not appear to lead to any improvements in the design, as also occurred when finding the pseudo-Bayesian A-optimal design.

We now compare the pseudo-Bayesian A-optimal design (`ex411$d`) to the design found under the normal-based approximation to the NSEL utility (`ex412$d`).

```
R> assess(d1 = ex412, d2 = ex411)
```

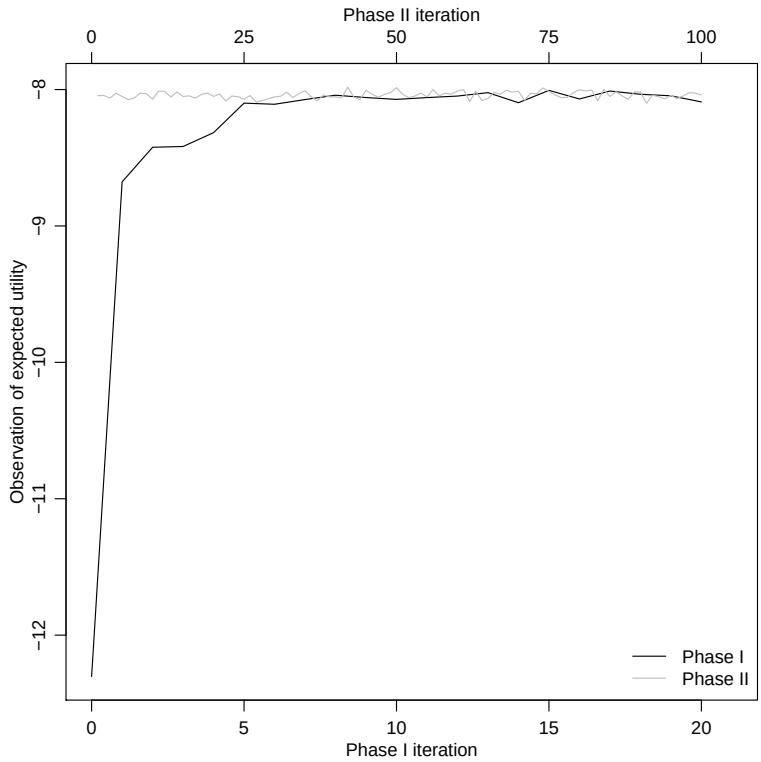

Figure 2: Convergence of phases I (lower x-axis) and II (upper x-axis) for the logistic regression model under the normal-based approximation to the NSEL utility function.

```
Mean (sd) approximate expected utility of d1 = -8.056569 (0.0367072)
Mean (sd) approximate expected utility of d2 = -10.61123 (0.03650658)
```

Notice how the optimal design under the normal-based approximation (`ex412$d`) achieves a substantially larger expected NSEL utility than the pseudo-Bayesian A-optimal design (`ex411$d`). Overstall and Woods (2017) and Overstall *et al.* (2018a) empirically investigated how the difference between NSEL and A-optimal designs decreases as n, the number of runs, increases, as the asymptotic approximation underpinning pseudo-Bayesian optimal designs improves. Overstall *et al.* (2018a) also found that, for this example, the difference between designs found using ACE with nested Monte Carlo and normal-based approximations to the expected NSEL utility were negligible, regardless of the value of n. However, designs under the normal-based approximation typically took around one-third of the computational time to find.

4.3. Model selection for chemical reactions

In this example, we demonstrate using the `ace` function for a problem that falls outside the capabilities of `aceglm` and `acenlm`, and illustrate construction of a bespoke `utility` function.

This example is adapted from Box and Hill (1967) and concerns mechanistic modelling of chemical reactions. We find designs with n runs where the ith run requires specifying the reaction time, $x_{i1} \in (0, 150)$, and temperature, $x_{i2} \in (450, 600)$, at which to measure reaction yield, y_i $(i = 1, \ldots, n)$. Therefore each run of the design is a two-vector $\boldsymbol{x}_i = (x_{i1}, x_{i2})^\top$. The following statistical model is posited:

$$y_i \sim \mathrm{N}\left(\mu(m, \boldsymbol{\theta}; \boldsymbol{x}_i), \sigma^2\right) ,$$

where

$$\mu(m, \boldsymbol{\theta}; \boldsymbol{x}_i) = \begin{cases} \exp\left(-\eta(\boldsymbol{\theta}; \boldsymbol{x}_i)\right) & \text{for } m = 0 \\ \\ \left[1 + m\eta(\boldsymbol{\theta}; \boldsymbol{x}_i)\right]^{-\frac{1}{m}} & \text{for } m = 1, 2, 3 \,. \end{cases}$$

Here

$$\eta(\boldsymbol{\theta}; \boldsymbol{x}_i) = \theta_1 x_{i1} \exp\left(-\frac{\theta_2}{x_{i2}}\right) ,$$

with unknown parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)^\top$ and $m \in \mathcal{M} = \{0, 1, 2, 3\}$ specifying the order of the reaction, with $m = 0, 1, 2, 3$ corresponding to first-, second-, third- and fourth-order reactions, respectively.

We create an R function to implement $\eta(\boldsymbol{\theta}; \boldsymbol{x}_i)$ with arguments `d`, an $n \times k$ matrix, and `theta`, a $B \times p$ matrix with bth row given by $\boldsymbol{\theta}_b = (\theta_{1b}, \theta_{2b})^\top$. It returns a $B \times n$ matrix with bith element given by $\theta_{1b} x_{i1} \exp\left(-\theta_{2b} x_{i2}\right)$. That is, it calculates the value of the function η for a given design $\boldsymbol{d}$ for every row of a matrix of parameter values `theta`.

```
R> etafunc <- function(d, theta) {
+   outer(theta[, 1], d[, 1]) * exp( - outer(theta[, 2], 1 / d[, 2]))
+ }
```

The aim of the experiment is to determine which order reaction is appropriate for the observed responses, i.e., a choice from the set $\mathcal{M} = \{0, 1, 2, 3\}$. That is, model selection with $\boldsymbol{\gamma} = (m)$

in $u(\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{d})$. Following Overstall *et al.* (2018a), identical prior distributions are assumed for the parameters under each model:

$$\theta_1 \sim \mathrm{N}\left(400, 25^2\right) , \quad \theta_2 \sim \mathrm{N}\left(5000, 250^2\right) . \tag{15}$$

We fix the response standard deviation as $\sigma = 0.1$, double the value assumed by Box and Hill (1967). We choose this larger value as use of too small a value of σ leads to the expected utility becoming less dependent on the design, i.e., the expected utility surface is quite flat. Equal prior probabilities are assumed for each model, i.e., $\pi(m) = 1/4$, for all $m \in \mathcal{M}$.

```
R> sig <- 0.1
R> prior <- function(B) {
+   theta1 <- rnorm(n = B, mean = 400, sd = 25)
+   theta2 <- rnorm(n = B, mean = 5000, sd = 250)
+   cbind(theta1, theta2)
+ }
```

We aim to find a design that maximises the expected 0-1 utility, i.e., the expectation of u_{01} given by Equation 8, and set $\delta_1 = \delta = 0$. For equal prior model probabilities, the posterior modal model, $M(\boldsymbol{y}, \boldsymbol{d})$, will maximise the marginal likelihood given by $\pi(\boldsymbol{y}|m, \boldsymbol{d}) = \int \pi(\boldsymbol{y}|m, \boldsymbol{d}, \boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}$. As the marginal likelihood is not available in closed form for these models, we use a Monte Carlo approximation implemented as a `utility` function which can then be passed to the `ace` function to find an optimal design.

We approximate u_{01} by

$$\tilde{u}_{01}(m, \boldsymbol{y}, \boldsymbol{d}) = I(m = \tilde{M}(\boldsymbol{y}, \boldsymbol{d})) , \tag{16}$$

where

$$\tilde{M}(\boldsymbol{y}, \boldsymbol{d}) = \arg\max \tilde{\pi}(\boldsymbol{y}|m, \boldsymbol{d}) ,$$

$$\tilde{\pi}(\boldsymbol{y}|m, \boldsymbol{d}) = \frac{1}{\tilde{B}} \sum_{b=1}^{\tilde{B}} \pi(\boldsymbol{y}|m, \tilde{\boldsymbol{\theta}}_b, \boldsymbol{d}) ,$$

with $\{\tilde{\boldsymbol{\theta}}_b\}_{b=1}^{\tilde{B}}$ a sample from the prior distribution given by Equation 15, likelihood $\pi(\boldsymbol{y}|m, \boldsymbol{\theta}, \boldsymbol{d}) = \prod_{i=1}^{n} \pi(y_i|m, \boldsymbol{\theta}, \boldsymbol{x}_i)$ and $\pi(y_i|m, \boldsymbol{\theta}, \boldsymbol{x}_i)$ being a normal density with mean $\mu(m, \boldsymbol{\theta}; \boldsymbol{x}_i)$ and variance σ^2.

The expected 0-1 utility can then be approximated as

$$\tilde{U}(\boldsymbol{d}) = \frac{1}{B} \sum_{b=1}^{B} \tilde{u}(m_b, \boldsymbol{y}_b, \boldsymbol{d}) ,$$

for a sample $\{m_b, \boldsymbol{\theta}_b, \boldsymbol{y}_b\}_{b=1}^{B}$ from the joint distribution of m, $\boldsymbol{\theta}$ and $\boldsymbol{y}$; such a sample can be easily generated by sampling a model indicator from $\mathcal{M}$ with probabilities $\pi(m)$, parameters from the prior distribution with density $\pi(\boldsymbol{\theta})$, and then responses from the conditional distribution with density $\pi(\boldsymbol{y}|m, \boldsymbol{\theta})$ (see the code below).

We can now create an R function, `util01`, to implement this approximate utility. Note that we set $\tilde{B} = 100$. The function takes as arguments a design `d` and Monte Carlo sample size `B`.

The main work is done within the nested `for` loop; for each data set generated in the outer loop, the posterior modal model is found by maximising a Monte Carlo approximation to the marginal likelihood, which is calculated in the inner loop. It returns a vector of `B` evaluations of the Monte Carlo approximated utility function $\tilde{u}(m, \boldsymbol{y}, \boldsymbol{d})$.

```
R> Btilde <- 100
R>
R> util01 <- function(d, B) {
+
+   theta <- prior(B)
+   mod <- sample(x = 0:3, size = B, replace = TRUE)
+
+   eta <- etafunc(d = d, theta = theta)
+   mu <- matrix(0, nrow = B, ncol = n)
+   mu[mod == 0, ] <- exp( - eta[mod == 0, ])
+   mu[mod == 1, ] <- (1 + eta[mod == 1, ]) ^ ( - 1)
+   mu[mod == 2, ] <- (1 + 2 * eta[mod == 2, ]) ^ ( - 1 / 2)
+   mu[mod == 3, ] <- (1 + 3 * eta[mod == 3, ]) ^ ( - 1 / 3)
+
+   Y <- mu + sig * matrix(rnorm(B * n), nrow = B)
+
+   thetatilde <- prior(Btilde)
+   etatilde <- etafunc(d = d, theta = thetatilde)
+   mutilde0 <- exp(-etatilde)
+   mutilde1 <- (1 + etatilde) ^ ( - 1)
+   mutilde2 <- (1 + 2 * etatilde) ^ ( - 1 / 2)
+   mutilde3 <- (1 + 3 * etatilde) ^ ( - 1 / 3)
+
+   modal <- rep(0, B)
+   for(b in 1:B) {
+     C <- matrix(0, nrow = Btilde, ncol = 4)
+     for(bt in 1:Btilde) {
+       C[bt, 1] <- sum(dnorm(x = Y[b, ], mean = mutilde0[bt, ],
+       sd = sig, log = TRUE))
+       C[bt, 2] <- sum(dnorm(x = Y[b, ], mean = mutilde1[bt, ],
+       sd = sig, log = TRUE))
+       C[bt, 3] <- sum(dnorm(x = Y[b, ], mean = mutilde2[bt, ],
+       sd = sig, log = TRUE))
+       C[bt, 4] <- sum(dnorm(x = Y[b, ], mean = mutilde3[bt, ],
+       sd = sig, log = TRUE))
+     }
+     logmarglik <- log(apply(exp(C), 2, mean))
+     modal[b] <- which.max(logmarglik) - 1
+   }
+   ifelse(modal == mod, 1, 0)
+ }
```

We use the `ace` function along with `util01` to find an optimal design. For illustration, we search for a design with $n = 20$ runs and set the arguments `B = c(1000, 100)`, `Q = 15`, `N2 = 0` (to skip Phase II), and using the arguments `lower` and `upper` to set the bounds for each factor (see Table 2). We specify a Bayesian hypothesis test of a difference in proportions by setting `binary = TRUE`.

```
R> set.seed(1)
R> n <- 20
R> q <- 2
R> lower <- cbind(rep(0, n), rep(450, n))
R> upper <- cbind(rep(150, n), rep(600, n))
R> start.d <- randomLHS(n = n, k = q)*(upper - lower) + lower
R> ex43 <- ace(utility = util01, start.d = start.d, B = c(1000, 100), Q = 15,
+   N2 = 0, binary = TRUE, lower = lower, upper = upper)
```

Note that C repetitions of the ACE algorithm could have been implemented using the `pace` function specifying a list of starting designs as the argument `start.d`. However we have only demonstrated `ace` here to minimise the computing time required to reproduce the example.

We compare the approximations to the expected 0-1 utility for the starting design and the final design from ACE as follows.

```
R> assess43 <- assess(d1 = ex43, d2 = start.d)
R> assess43

Mean (sd) approximate expected utility of d1 = 0.8789 (0.00611211)
Mean (sd) approximate expected utility of d2 = 0.80565 (0.0120973)
```

Assuming the data generating process is consistent with one of the models, the starting design identifies the true model about 81% of the time. By using an optimal design we can increase this to about 88%.

4.4. Optimal design for prediction

Prediction from a nonparametric regression model is a common aim of both spatial studies and computer experiments, often using Gaussian process (GP) regression (or Kriging). For example, optimal sensor placements, e.g., for pollution or environmental monitoring, may be sought within a geographical region of interest to provide accurate predictions at unsampled locations. See Diggle and Lophaven (2006), Zimmerman (2006) and Uciński and Patan (2007). As the cost of maintaining large monitoring networks can be high, costs are often also associated with the placement of each sensor. We demonstrate using **acebayes** to find an optimal design for such a problem.

Here, a design consists of n locations $\boldsymbol{x}_i = (x_{i1}, x_{i2})^\top$, within a specified two-dimensional region, at each of which a response y_i will be observed. The aim is to fit a Gaussian process model to the resulting responses, $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$, and to predict the n_0 unobserved responses, $\boldsymbol{y}_0 = (y_{01}, \ldots, y_{0n_0})^\top$, where y_{0i} is associated with pre-specified location $\boldsymbol{x}_{0i} = (x_{0i1}, x_{0i2})^\top$, for $i = 1, \ldots, n_0$. Let $\tilde{\boldsymbol{y}} = \left(\boldsymbol{y}^\top, \boldsymbol{y}_0^\top\right)^\top$ denote the $\tilde{n} \times 1$ vector of observed and unobserved responses where $\tilde{n} = n + n_0$.

The design problem is to specify the $n \times 2$ matrix $\boldsymbol{d}$ (with ith row $\boldsymbol{x}_i^\top$) to best predict $\boldsymbol{y}_0$ where the meaning of "best" is controlled by the choice of utility function (see later).

A zero-mean Gaussian process model results in the assumption of multivariate normal distribution for $\tilde{\boldsymbol{y}}$,

$$\tilde{\boldsymbol{y}}|\sigma^2, \phi, \tau^2 \sim N\left(\mathbf{0}, \sigma^2\tilde{\boldsymbol{\Sigma}}\right), \tag{17}$$

where $\sigma^2 > 0$ is a scale parameter and

$$\tilde{\boldsymbol{\Sigma}} = \tilde{\mathbf{C}} + \tau^2\mathbf{I}_{\tilde{n}}. \tag{18}$$

In Equation 18, $\mathbf{I}_{\tilde{n}}$ is the $\tilde{n} \times \tilde{n}$ identity matrix, $\tau^2 > 0$ is the nugget, and $\tilde{\mathbf{C}}$ is an $\tilde{n} \times \tilde{n}$ correlation matrix partitioned as follows

$$\tilde{\mathbf{C}} = \begin{pmatrix} \mathbf{C} & \mathbf{S} \\ \mathbf{S}^\top & \mathbf{C}_0 \end{pmatrix}. \tag{19}$$

In Equation 19, $\mathbf{C}$ is an $n \times n$ matrix with rtth element

$$C_{rt} = \rho(\boldsymbol{x}_r, \boldsymbol{x}_t, \phi), \quad \text{for } r, t = 1, \ldots, n,$$

$\mathbf{C}_0$ is an $n_0 \times n_0$ matrix with rtth element

$$C_{0rt} = \rho(\boldsymbol{x}_{0r}, \boldsymbol{x}_{0t}, \phi), \qquad \text{for } r, t = 1, \ldots, n_0,$$

$\mathbf{S}$ is an $n \times n_0$ matrix with rtth element

$$S_{rt} = \rho(\boldsymbol{x}_r, \boldsymbol{x}_{0t}, \phi), \qquad \text{for } r = 1, \ldots, n \text{ and } t = 1, \ldots, n_0,$$

and ρ is a known correlation function. In this example, we employ the squared exponential correlation function, $\rho(\boldsymbol{x}_k, \boldsymbol{x}_l; \phi) = \exp\{-\phi\sum_{j=1}^{2}(x_{kj} - x_{lj})^2\}$, as implemented by the following R function.

```
R> rho <- function(X1, X2, phi) {
+   k <- ncol(X1)
+   n1 <- nrow(X1)
+   n2 <- nrow(X2)
+   A <- matrix(0, nrow = n1, ncol = n2)
+   for(i in 1:k) {
+     A <- A - phi * (matrix(rep(X1[, i], n2), nrow =n1) -
+     matrix(rep(X2[, i], each = n1), nrow = n1)) ^ 2
+   }
+   exp(A)
+ }
```

We adopt a Bayesian approach with conjugate prior distribution assigned to the parameter σ^2:

$$\sigma^{-2} \sim \text{Gamma}\left(\frac{a}{2}, \frac{b}{2}\right),$$

where $a = 3$ and $b = 1$ are known shape and rate parameters, respectively. The correlation parameter $\phi = 0.5$ and nugget $\tau^2 = 1 \times 10^{-5}$ are assumed known and fixed.

```
R> phi <- 1
R> tau2 <- 0.00001
R> a <- 3
R> b <- 1
```

We assume that the two-dimensional region is such that $0 \leq x_{ij} \leq 1$. The n_0 pre-specified locations $\boldsymbol{x}_{01}, \ldots, \boldsymbol{x}_{0n_0}$ are given by the points on an evenly-spaced $r \times r$ grid where $r = 10$, i.e., $n_0 = 100$. We let $\boldsymbol{d}_0$ be the $n_0 \times 2$ matrix with ith row given by $\boldsymbol{x}_{0i}^\top$, for $i = 1, \ldots, n_0$.

```
R> k <- 2
R> r <- 10
R> n0 <- r ^ k
R> x0 <- seq(from = 0, to = 1, length.out = r)
R> d0 <- as.matrix(expand.grid(x0, x0))
```

We find optimal designs for prediction using a utility function adapted from Sansó and Müller (1997) and Müller *et al.* (2004) which compromises between the accuracy of the posterior predictive mean, $\mathrm{E}(\boldsymbol{y}_0|\boldsymbol{y})$, at the n_0 new locations $\boldsymbol{x}_{01}, \ldots, \boldsymbol{x}_{0n_0}$ against the cost of the n placed sensors:

$$u(\boldsymbol{y}_0, \boldsymbol{y}, \boldsymbol{d}) = \sum_{i=1}^{n_0} I(\mathrm{E}(y_{0i}|\boldsymbol{y}) - \delta < y_{0i} < \mathrm{E}(y_{0i}|\boldsymbol{y}) + \delta) - \sum_{i=1}^{n} c(\boldsymbol{x}_i)\,, \tag{20}$$

where $\delta > 0$ controls the desired accuracy and $c(\boldsymbol{x}_i)$ is the cost of taking an observation at $\boldsymbol{x}_i$. In this example, the cost $c(\boldsymbol{x}_i)$ depends on the location of the proposed sensor and is given by $c(\boldsymbol{x}_i) = x_{i1}^2 + x_{i2}^2$, i.e., the squared Euclidean distance from the origin. The total cost of the design is given by $\sum_{i=1}^{n} c(\boldsymbol{x}_i)$. Under the model given by Equation 17, the posterior predictive mean of $\boldsymbol{y}_0$ is given by

$$\mathrm{E}(\boldsymbol{y}_0|\boldsymbol{y}) = \mathbf{S}^\top \left(\mathbf{C} + \tau^2 \mathbf{I}_n\right)^{-1} \boldsymbol{y}\,, \tag{21}$$

with ith element $\mathrm{E}(y_{0i}|\boldsymbol{y})$, for $i = 1, \ldots, n_0$.

A Monte Carlo approximation to the expectation of $u(\boldsymbol{y}_0, \boldsymbol{y}, \boldsymbol{d})$ given by Equation 20 can be constructed as

$$\tilde{U}(\boldsymbol{d}) = \frac{1}{B} \sum_{b=1}^{B} u(\boldsymbol{y}_{0b}, \boldsymbol{y}_b, \boldsymbol{d})\,,$$

with $\tilde{\boldsymbol{y}}_b = (\boldsymbol{y}_b^\top, \boldsymbol{y}_{0b}^\top)^\top$ sampled from the marginal distribution of $\tilde{\mathbf{y}}$.

The function below implements this approximation, returning a vector of `B` evaluations of $u(\boldsymbol{y}_0, \boldsymbol{y}, \boldsymbol{d})$. We have specified $\delta = 0.25$.

```
R> delta <- 0.25
R>
R> utilpred <- function(d, B) {
+
+   n <- dim(d)[1]
+   C <- rho(d, d, phi)
```

```
+   S <- rho(d, d0, phi)
+   C0 <- rho(d0, d0, phi)
+
+   sig2 <- 1 / rgamma(n = B, shape = 0.5 * a, rate = 0.5 * b)
+   Sigmatilde <- rbind(cbind(C, S), cbind(t(S), C0)) + tau2 * diag(n + n0)
+   cSigmatilde <- chol(Sigmatilde)
+   ytilde <- matrix(0, nrow = n + n0, ncol = B)
+   for(b in 1:B) {
+     ytilde[, b] <- sqrt(sig2[b]) * t(rnorm(n + n0) %*% cSigmatilde)
+   }
+
+   y <- ytilde[1:n, ]
+   y0 <- ytilde[ - (1:n), ]
+
+   postpredmean <- t(S) %*% solve(C + tau2 * diag(n)) %*% y
+
+   accuracy <- rep(0,B)
+   for(b in 1:B) {
+     accuracy[b] <- sum(as.numeric(((postpredmean[, b] - delta) < y0[, b]) *
+       ((postpredmean[, b] + delta) > y0[,b])))
+   }
+
+   cost <- sum(apply(d^2, 1, sum))
+
+   accuracy - cost
+ }
```

We now illustrate the use of `ace` with these functions by finding a design with $n = 10$ sensors.

```
R> set.seed(1)
R> n <- 10
R> start.d <- randomLHS(n = n, k = k)
R> ex44 <- ace(utility = utilpred, start.d = start.d, lower = 0, upper = 1)
```

We can compare the value of the objective function for the starting design and the optimal design obtained from ACE.

```
R> assess(d1 = ex44, d2 = start.d)

Mean (sd) approximate expected utility of d1 = 95.86214 (0.02456676)
Mean (sd) approximate expected utility of d2 = 92.41169 (0.02434313)
```

We also compare the cost of each design.

```
R> sum(apply(start.d ^ 2, 1, sum))

[1] 6.653732
```

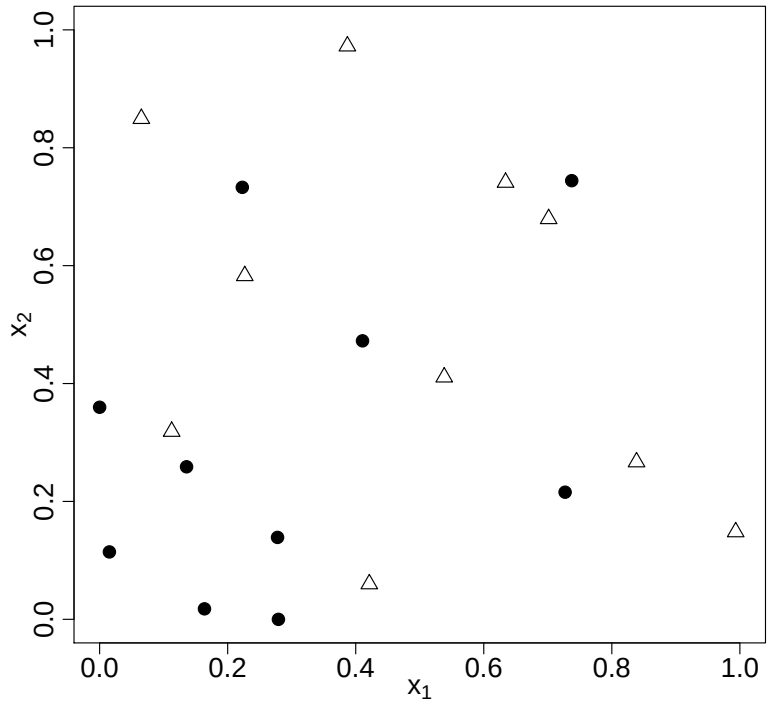

Figure 3: Bayesian optimal (•) and starting (△) designs for the prediction example.

```
R> sum(apply(ex44$phase2.d ^ 2, 1, sum))

[1] 3.081133
```

The difference in utility values between the Bayesian optimal design and the starting design are mostly due to the much lower cost of the sensor placements in the optimal design. That is, the optimal design has similar predictive accuracy as the starting design but at substantially reduced cost. Figure 3 shows the very different distribution of the points in the optimal design compared to the starting design, with the former having many more points in areas of low cost near the origin.

5. Discussion

Bayesian optimal design is conceptually straightforward but often difficult, and computationally expensive, to implement. The **acebayes** package provides a suite of functions that allow optimal designs to be found for complex and realistic problems, with dimensionality at least one order of magnitude greater than other current methods. The general purpose `ace` and `pace` functions can be used to solve very general, and bespoke, design problems. The functions `(p)aceglm` and `(p)acenlm` can find designs for common classes of statistical models.

Any set of examples can only be illustrative, and those in this paper are no different. To aid exposition, we have deliberately kept the problems relatively simple and the designs sought have been small. However, **acebayes** has been used to find designs for more complex

scenarios. These include high-dimensional design spaces (Overstall and Woods 2017) and ordinary differential equation models (Overstall *et al.* 2018d). It has also been used (Overstall and McGree 2018) to find design for intractable likelihood models (e.g. Drovandi and Pettitt 2013) and for synthesis optimisation of pharmaceutical products (Overstall *et al.* 2018b).

While **acebayes** allows much larger designs to be found than existing methods, for complex problems we would recommend coding the utility function in a low-level programming language (e.g., C/C++) and running the code on a computational cluster. The algorithm is heuristic, and so to overcome convergence to local optima, it should be run multiple times from different starting designs, for example using parallel computing and/or the `pace` function.

We have created a YouTube channel (`https://bit.ly/2SSC3ur`) hosting tutorial videos and a Twitter account (`@acebayes`) to update users on the latest developments to the **acebayes** package.

Acknowledgments

D.C. Woods was partially supported by a Fellowship EP/J018317/1 from the UK Engineering and Physical Sciences Research Council. The authors are grateful for constructive comments from two anonymous reviewers that improved the paper.

A. The approximate coordinate exchange algorithm

This appendix provides details on Phase I (Appendix A.1) and Phase II (Appendix A.2) of the ACE algorithm.

A.1. Phase I

1. Choose an initial design $\boldsymbol{d}^0 \in \mathcal{D}$ and set the current design to be $\boldsymbol{d}^C = \boldsymbol{d}^0$.
2. For $i = 1, \ldots, n$ and $j = 1, \ldots, k$, complete the following steps.
   (a) Let $\boldsymbol{d}^C(x_{ij}^q)$ equal $\boldsymbol{d}^C$ with ijth coordinate (entry) replaced by x_{ij}^q, where $x_{ij}^1, \ldots, x_{ij}^Q$ are the points from a one-dimensional space filling design in $\mathcal{D}_{ij}$, the design space for the ijth element of $\boldsymbol{d}$.
   (b) For $q = 1, \ldots, Q$, evaluate $\tilde{U}[\boldsymbol{d}^C(x_{ij}^q)]$, the approximation to the expected utility, e.g., Equation 4. Fit a Gaussian process emulator to "data" $\left\{x_{ij}^q, \tilde{U}[\boldsymbol{d}(x_{ij}^q)]\right\}_{q=1}^{Q}$, and set $\hat{U}_{ij}(x)$ to be the resulting predictive mean.
   (c) Find
   $$x_{ij}^\star = \arg\max_{x \in \mathcal{D}_{ij}} \hat{U}_{ij}(x)\,,$$
   and set $\boldsymbol{d}^\star = \boldsymbol{d}^C(x_{ij}^\star)$.
   (d) For a stochastic (e.g., Monte Carlo) approximation $\tilde{U}$, set $\boldsymbol{d}^C = \boldsymbol{d}^\star$ with probability p^* derived from a Bayesian hypothesis test. For a deterministic approximation (e.g., quadrature), set $\boldsymbol{d}^C = \boldsymbol{d}^\star$ if $\tilde{U}(\boldsymbol{d}^\star) > \tilde{U}(\boldsymbol{d}^C)$.

3. Repeat Step 2 N_1 times.

A.2. Phase II

1. Set the current design, $\boldsymbol{d}^C$, to be the final design from Phase I of the ACE algorithm.

2. For $i = 1, \ldots, n$, set
$$\boldsymbol{d}_i^{(1)} = \left[(\boldsymbol{d}^C)^\top, (\boldsymbol{x}_i^C)^\top\right]^\top ,$$
where $(\boldsymbol{x}_i^C)^\top$ is the ith row of $\boldsymbol{d}^C$; that is, form $\boldsymbol{d}_i^{(1)}$ by augmenting $\boldsymbol{d}^C$ with a repeat of the ith run.

3. Find $i^\star = \arg\max_{i=1,\ldots,n} \tilde{U}(\boldsymbol{d}_i^{(1)})$ and set $\boldsymbol{d}^{(2)} = \boldsymbol{d}_{i^\star}^{(1)}$.

4. For $h = 1, \ldots, n+1$, set
$$\boldsymbol{d}_h^{(3)} = \left[(\boldsymbol{x}_1^{(2)})^\top, \ldots, (\boldsymbol{x}_{h-1}^{(2)})^\top, (\boldsymbol{x}_{h+1}^{(2)})^\top, \ldots, (\boldsymbol{x}_{n+1}^{(2)})^\top\right]^\top ,$$
where $(\boldsymbol{x}_h^{(2)})^\top$ is the hth row of $\boldsymbol{d}^{(2)}$; that is, form $\boldsymbol{d}_h^{(3)}$ by removing the hth run.

5. Find $h^\star = \arg\max_{h=1,\ldots,n+1} \tilde{U}(\boldsymbol{d}_h^{(3)})$ and set $\boldsymbol{d}^* = \boldsymbol{d}_{h^\star}^{(3)}$.

6. For a stochastic (e.g., Monte Carlo) approximation $\tilde{U}$, set $\boldsymbol{d}^C = \boldsymbol{d}^\star$ with probability p^* derived from a Bayesian hypothesis test. For a deterministic approximation (e.g., quadrature), set $\boldsymbol{d}^C = \boldsymbol{d}^\star$ if $\tilde{U}(\boldsymbol{d}^\star) > \tilde{U}(\boldsymbol{d}^C)$.

7. Repeat steps 2 to 6 N_2 times.

References

Amzal B, Bois FY, Parent E, Robert CP (2006). "Bayesian-Optimal Design via Interacting Particle Systems." *Journal of the American Statistical Association*, **101**, 773–785.

Atkinson AC, Donev AN, Tobias RD (2007). *Optimum Experimental Designs, with SAS*. 2nd edition. Oxford University Press, Oxford.

Berger JO (1985). *Statistical Decision Theory and Bayesian Analysis*. 2nd edition. Springer-Verlag, New York.

Box GEP, Hill WJ (1967). "Discrimination among Mechanistic Models." *Technometrics*, **9**, 57–71.

Box GEP, Hunter JS, Hunter WG (2005). *Statistics for Experimenters: Design, Discovery and Innovation*. 2nd edition. John Wiley & Sons, Hoboken, New Jersey.

Bush S, Ruggiero K (2016). ***designGLMM**: Finding Optimal Block Designs for a Generalised Linear Mixed Model*. R package version 0.1.0, URL https://CRAN.R-project.org/package=designGLMM.

Carnell R (2016). ***lhs**: Latin Hypercube Samples*. R package version 0.14, URL https://CRAN.R-project.org/package=lhs.

Chaloner K (1984). "Optimal Bayesian Experimental Designs for Linear Models." *The Annals of Statistics*, **12**, 283–300.

Chaloner K, Verdinelli I (1995). "Bayesian Experimental Design: A Review." *Statistical Science*, **10**, 273–304.

Diggle P, Lophaven S (2006). "Bayesian Geostatistical Design." *Scandinavian Journal of Statistics*, **33**, 53–64.

Drovandi CC, McGree JM, Pettitt AN (2013). "Sequential Monte Carlo for Bayesian Sequentially Designed Experiments for Discrete Data." *Computational Statistics and Data Analysis*, **57**, 320–335.

Drovandi CC, McGree JM, Pettitt AN (2014). "A Sequential Monte Carlo Algorithm to Incorporate Model Uncertainty in Bayesian Sequential Design." *Journal of Computational and Graphical Statistics*, **23**, 3–24.

Drovandi CC, Pettitt AN (2013). "Bayesian experimental design for models with intractable likelihoods." *Biometrics*, **69(4)**, 937–948.

Eddelbuettel D, Francois R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**, 1–18.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063.

Felsenstein K (1992). "Optimal Bayesian Design for Discrimination Among Rival Models." *Computational Statistics and Data Analysis*, **14**, 427–436.

Goos P, Jones B (2011). *Optimal design of experiments: a case study approach*. John Wiley & Sons, Ltd.

Gotwalt CM, Jones BA, Steinberg DM (2009). "Fast Computation of Designs Robust to Parameter Uncertainty for Nonlinear Settings." *Technometrics*, **51**, 88–95.

Harman R, Filova L (2016). ***OptimalDesign**: Algorithms for D-, A-, and IV-Optimal Designs*. R package version 0.2, URL https://CRAN.R-project.org/package=OptimalDesign.

Lange K (2013). *Optimization*. 2nd edition. Springer-Verlag, New York.

Lindley DV (1956). "On a Measure of the Information Provided by an Experiment." *The Annals of Mathematical Statistics*, **27**, 986–1005.

Long Q, Scavino M, Tempone R, Wang S (2013). "Fast Estimation of Expected Information Gains for Bayesian Experimental Designs Based on Laplace Approximations." *Computer Methods in Applied Mechanics and Engineering*, **259**, 24–39.

Masoudi E, Holling H, Wong WK (2017). "Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs." *Computational Statistics and Data Analysis*, **113**, 330–345.

Masoudi E, Holling H, Wong WK (2018). ***ICAOD**: Imperialist Competitive Algorithm for Optimal Designs*. R package version 0.9.7, URL https://CRAN.R-project.org/package=ICAOD.

Masoudi E, Sarmad M, Talebi H (2013). ***LDOD**: Finding Locally D-optimal Designs for some Nonlinear and Generalized Linear Models*. R package version 1.0, URL https://CRAN.R-project.org/package=LDOD.

Meyer RK, Nachtsheim CJ (1995). "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs." *Technometerics*, **37**, 60–69.

Monahan J, Genz A (1997). "Spherical-Radial Integration Rules for Bayesian Computation." *Journal of the American Statistical Association*, **92**, 664–674.

Müller P (1999). "Simulation-Based Optimal Design." In JM Bernardo, JO Berger, AP Dawid, AFM Smith (eds.), *Bayesian Statistics 6*, pp. 459–474. Oxford University Press, Oxford.

Müller P, Parmigiani G (1995). "Optimal Design via Curve Fitting of Monte Carlo Experiments." *Journal of the American Statistical Association*, **90**, 1322–1330.

Müller P, Sansó B, De Iorio M (2004). "Optimal Bayesian Design by Inhomogeneous Markov Chain Simulation." *Journal of the American Statistical Association*, **99(467)**, 788–798.

Nyberg J, Ueckert S, Strömberg EA, Karlsson M, Hooker AC (2012). "PopED: An Extended, Parallelized, Nonlinear Mixed Effects Models Optimal Design Tool." *Computer Methods and Programs in Biomedicine*, **108**, 789–805.

Overstall A, McGree J, Drovandi C (2018a). "An Approach for Finding Fully Bayesian Optimal Designs using Normal-Based Approximations to Loss Functions." *Statistics and Computing*, **28**(2), 343–358.

Overstall AM, McGree JM (2018). "Bayesian design of experiments for intractable likelihood models using coupled auxiliary models and multivariate emulation." Available at: arXiv:1803.07018 [stat.ME].

Overstall AM, Woods D, Martin K (2018b). "Bayesian prediction for physical models with application to the optimization of the synthesis of pharmaceutical products using chemical kinetics." *Computational Statistics and Data Analysis*. To appear.

Overstall AM, Woods DC (2017). "Bayesian Design of Experiments using Approximate Coordinate Exchange." *Technometrics*, **59**, 458–470.

Overstall AM, Woods DC, Adamou M (2018c). ***acebayes**: Optimal Bayesian Experimental Design using the ACE Algorithm*. R package version 1.6.0, URL https://CRAN.r-project.org/package=acebayes.

Overstall AM, Woods DC, Parker BM (2018d). "Bayesian optimal design for ordinary differential equation models." Available at: arXiv:1509.04099v2 [stat.ME].

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rasmussen CE, Williams CKI (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, USA.

Ryan EG, Drovandi CC, McGree JM, Pettitt AN (2016). "A Review of Modern Computational Algorithms for Bayesian Optimal Design." *International Statistical Review*, **84**, 128–154.

Ryan EG, Drovandi CC, Thompson MH, Pettitt AN (2014). "Towards Bayesian Experimental Design for Nonlinear Models that Require a Large Number of Sampling Times." *Computational Statistics and Data Analysis*, **70**, 45–60.

Ryan K (2003). "Estimating Expected Information Gains for Experimental Designs with Application to the Random Fatigue-Limit Model." *Journal of Computational and Graphical Statistics*, **12**, 585–603.

Sansó B, Müller P (1997). "Redesigning a Network of Rainfall Stations." In C Gatsonis, RE Kass, B Carlin, A Carriquiry, A Gelman, I Verdinelli, M West (eds.), *Case Studies in Bayesian Statistics IV*, pp. 383–393. Springer-Verlag, New York.

Santner TJ, Williams BJ, Notz WI (2003). *The Design and Analysis of Computer Experiments*. Springer-Verlag, New York.

Stigler SM (2016). *The Seven Pillars of Statistical Wisdom*. Harvard University Press, Cambridge, Massachusetts.

Uciński D, Patan M (2007). "D-optimal Design of a Monitoring Network for Parameter Estimation of Distributed Systems." *Journal of Global Optimisation*, **39**(2), 291–322.

Wheeler R (2014). ***AlgDesign**: Algorithmic Experimental Design*. R package version 1.1-7.3, URL https://CRAN.R-project.org/package=AlgDesign.

Woods DC, Overstall AM, Adamou M, Waite TW (2017). "Bayesian Design of Experiments for Generalised Linear Models and Dimensional Analysis with Industrial and Scientific Application (with Discussion)." *Quality Engineering*, **29**, 91–118.

Zimmerman D (2006). "Optimal Network Design for Spatial Prediction, Covariance Parameter Estimation and Empirical Prediction." *Environmetrics*, **17**, 635–652.

Affiliation:

Antony M. Overstall, David C. Woods and Maria Adamou
Southampton Statistical Sciences Research Institute
University of Southampton
Southampton, SO17 1BJ, UK
E-mail: {A.M.Overstall,D.Woods,M.Adamou}@southampton.ac.uk