Efficient construction of Bayes optimal designs for stochastic process models

Colin S. Gillespie* Richard J. Boys

School of Mathematics, Statistics and Physics, Newcastle University,
Newcastle upon Tyne, NE1 7RU, UK

Abstract

Stochastic process models are now commonly used to analyse complex biological, ecological and industrial systems. Increasingly there is a need to deliver accurate estimates of model parameters and assess model fit by optimizing the timing of measurement of these processes. Standard methods to construct Bayes optimal designs, such the well known Müller algorithm, are computationally intensive even for relatively simple models. A key issue is that, in determining the merit of a design, the utility function typically requires summaries of many parameter posterior distributions, each determined via a computer-intensive scheme such as MCMC. This paper describes a fast and computationally efficient scheme to determine optimal designs for stochastic process models. The algorithm compares favourably with other methods for determining optimal designs and can require up to an order of magnitude fewer utility function evaluations for the same accuracy in the optimal design solution. It benefits from being embarrassingly parallel and is ideal for running on multi-core computers. The method is illustrated by determining different sized optimal designs for three problems of increasing complexity.

Keywords: observation times, particle representation, prior predictive distribution, utility function.

1 Introduction

Stochastic process models are increasingly used to describe the dynamic evolution of a complex system containing different interacting species. Applications appear in many areas such as biology, ecology, pharmacokinetics and industry; see, for example, Henderson et al. (2009), Cook et al. (2008), Ryan et al. (2015) and Khatab et al. (2017). Designed experiments can be very useful to the practitioner as they allow them to learn about models and their parameters in an efficient way. For example, in systems biology, an experimenter might build a stochastic kinetic model for their biological system; these models are typically described through a series of reactions between the species, with each reaction depending on an unknown stochastic rate constant. Data are then collected with the aim of estimating these constants and assessing model fit. Clearly scheduling the timing of say k observations within a $(0, T)$ experimental time period has the potential to yield much more accurate inferences than say just observing the process at k times on a regular grid.

We consider designs in which the stochastic process is observed on k occasions, at times $\boldsymbol{d} = (t_1, \ldots, t_k)$. In general, the merit of a particular design $\boldsymbol{d}$ is captured through a utility function $u(\boldsymbol{d}, \boldsymbol{y}, \boldsymbol{\theta})$, where $\boldsymbol{y}$ are data that might be observed at times $\boldsymbol{d}$ when the model parameter is $\boldsymbol{\theta}$. In this paper we focus on utility functions based on the posterior distribution of $\boldsymbol{\theta}$, namely, the posterior generalised precision

$$u(\boldsymbol{d}, \boldsymbol{y}) = 1/\det\{\mathrm{Var}(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{d})\} \tag{1}$$

*email: colin.gillespie@newcastle.ac.uk

Algorithm 1 MCMC algorithm by Müller (1999)

Initialise $\boldsymbol{d}^0$, simulate $\boldsymbol{y}_1^0, \ldots, \boldsymbol{y}_J^0 \overset{indep}{\sim} \pi(\boldsymbol{y}|\boldsymbol{d}^0)$
Calculate $u^0 = \prod_{j=1}^{J} u(\boldsymbol{d}^0, \boldsymbol{y}_j^0)$
for $i = 1$ to N **do**
  Propose $\boldsymbol{d}^* \sim q(\boldsymbol{d}|\boldsymbol{d}^{i-1})$
    and simulate $\boldsymbol{y}_1^*, \ldots, \boldsymbol{y}_J^* \overset{indep}{\sim} \pi(\boldsymbol{y}|\boldsymbol{d}^*)$
  Calculate $u^* = \prod_{j=1}^{J} u(\boldsymbol{d}^*, \boldsymbol{y}_j^*)$
  if $U(0,1) < \min\left(1, u^* q(\boldsymbol{d}^{i-1}|\boldsymbol{d}^*)/\{u^{i-1} q(\boldsymbol{d}^*|\boldsymbol{d}^{i-1})\}\right)$
    then $u^i = u^*$, $\boldsymbol{d}^i = \boldsymbol{d}^*$ **else** $u^i = u^{i-1}$, $\boldsymbol{d}^i = \boldsymbol{d}^{i-1}$
end

and its logarithm. These utilities have an intuitive motivation and are appropriate if the posterior is unimodal and without substantial skewness or kurtosis. Note that these utility functions do not depend on the model parameter $\boldsymbol{\theta}$. Other popular utility functions (not used here) that do depend on the model parameter $\boldsymbol{\theta}$ are those based on the self-information loss, absolute error loss and squared error loss (Overstall et al., 2018).

As the choice of design must be made before observing data, designs should be assessed by their expected utility

$$u(\boldsymbol{d}) = E_{\boldsymbol{y}}\{u(\boldsymbol{d}, \boldsymbol{y})\} = \int_{\boldsymbol{y}} u(\boldsymbol{d}, \boldsymbol{y}) \pi(\boldsymbol{y}|\boldsymbol{d})\, d\boldsymbol{y},$$

where $\pi(\boldsymbol{y}|\boldsymbol{d}) = \int_{\boldsymbol{\theta}} \pi(\boldsymbol{y}|\boldsymbol{d}, \boldsymbol{\theta}) \pi(\boldsymbol{\theta}) d\boldsymbol{\theta}$ is the prior predictive density of the unobserved data, $\pi(\boldsymbol{y}|\boldsymbol{d}, \boldsymbol{\theta})$ is the density of the unobserved data $\boldsymbol{y}$ when using design $\boldsymbol{d}$ and the model parameter is $\boldsymbol{\theta}$, and $\pi(\boldsymbol{\theta})$ is the prior density describing uncertainty in the model parameter. Therefore the optimal design $\boldsymbol{d}^*$ over a design space $\mathcal{D}$ is given by

$$\boldsymbol{d}^* = \arg\max_{\boldsymbol{d} \in \mathcal{D}} u(\boldsymbol{d}).$$

Unfortunately the expected utility $u(\boldsymbol{d})$ is rarely analytically tractable and so computational schemes are needed. As standard Monte Carlo integration methods are also not feasible for non-trivial problems, Müller (1999) proposed a Monte Carlo Markov chain (MCMC) approach. Although the general Müller scheme allows for the utility function to depend on $\boldsymbol{\theta}$, here we focus on utility functions which depend only on the design $\boldsymbol{d}$ and potential observations $\boldsymbol{y}$. In this scenario the Müller scheme targets the density

$$h(\boldsymbol{d}, \boldsymbol{y}) \propto u(\boldsymbol{d}, \boldsymbol{y}) \pi(\boldsymbol{y}|\boldsymbol{d})$$

using the MCMC scheme in Algorithm 1 (with $J = 1$). The key feature of this scheme is that the marginal distribution over $\boldsymbol{y}$ is proportional to the expected utility $u(\boldsymbol{d})$. Therefore the optimal design can be estimated as the mode of the empirical marginal for $\boldsymbol{d}$, obtained from the MCMC sample. Note that this scheme mixes over design space by proposing moves using a proposal distribution q for designs. Also realisations from the prior predictive distribution of the unobserved data $\pi(\boldsymbol{y}|\boldsymbol{d})$ are obtained straightforwardly by first simulating a parameter value $\boldsymbol{\theta}$ from the prior distribution and then data $\boldsymbol{y}$ from the stochastic model. Müller suggested that this mode might be identified more easily by using $J > 1$ replicates from the prior predictive distribution. Here the MCMC scheme target becomes

$$h_J(\boldsymbol{d}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_J) \propto \prod_{j=1}^{J} u(\boldsymbol{d}, \boldsymbol{y}_j) \pi(\boldsymbol{y}_j|\boldsymbol{d}).$$

Algorithm 2 Step m of the Resampling-Markov algorithm by Amzal et al. (2006)

for $i = 1$ to N **do**
  Simulate $\boldsymbol{y}_{ij}^{m-1} \overset{indep}{\sim} \pi(\boldsymbol{y}|\boldsymbol{d}_i^{m-1})$,
    $j = J_{m-1}+1, \ldots, J_m$
  Calculate $w_i^m = w_i^{m-1} \prod_{j=J_{m-1}+1}^{J_m} u(\boldsymbol{d}_i^{m-1}, \boldsymbol{y}_{ij}^{m-1})$
end
Simulate $(\ell_1, \ldots, \ell_N) \sim M(N, \boldsymbol{w}^m)$
for $i = 1$ to N Set $\widehat{\boldsymbol{d}}_i = \boldsymbol{d}_{\ell_i}$ and $\widehat{u}_i^m = \widehat{u}_{\ell_i}^{m-1} \times w_{\ell_i}^m$
for $i = 1$ to N **do**
  Simulate $\overline{\boldsymbol{d}}_i^m \sim q_{MH}(\boldsymbol{d}|\widehat{\boldsymbol{d}}_i^m)$
    and $\overline{\boldsymbol{y}}_{ij}^m \overset{indep}{\sim} \pi(\boldsymbol{y}|\overline{\boldsymbol{d}}_i^m)$, $j = 1, \ldots, J_m$
  Calculate $\overline{u}_i^m = \prod_{j=1}^{J_m} u(\overline{\boldsymbol{d}}_i^m, \overline{\boldsymbol{y}}_{ij}^m)$
  Calculate
  $\alpha_i = \min[1, \{\overline{u}_i^m \, q_{MH}(\widehat{\boldsymbol{d}}_i^m|\overline{\boldsymbol{d}}_i^m)\}/\{\widehat{u}_i^m \, q_{MH}(\overline{\boldsymbol{d}}_i^m|\widehat{\boldsymbol{d}}_i^m)\}]$
  if $U(0,1) < \alpha_i$ **then** $\boldsymbol{d}_i^m = \overline{\boldsymbol{d}}_i^m$ **else** $\boldsymbol{d}_i^m = \widehat{\boldsymbol{d}}_i^m$
end

The marginal for $\boldsymbol{d}$ is now proportional to $u(\boldsymbol{d})^J$ and so, as J increases, the variance of this marginal is reduced and the mode is identified more easily. However, a significant problem with this algorithm is that, for large J, the computational burden becomes prohibitive and the algorithm risks getting stuck in a local mode.

Amzal et al. (2006) have developed a particle-based approach to target $h_J(\boldsymbol{d})$. Their method begins by first constructing a list of increasing values of J: $J_1 < J_2 < \cdots < J_M$. The main loop in their Resampling-Markov algorithm starts with a sample drawn approximately from $h_{J_{m-1}}(\boldsymbol{d})$ that is then resampled and enriched by a Markov step (with proposal distribution q_{MH}) to become an approximated sample from $h_{J_m}(\boldsymbol{d})$. The algorithm for a particular choice of J is given in Algorithm 2. It has many strengths over the standard Müller algorithm, particularly in the way it adaptively searches for the optimal design.

More recently, algorithms have been developed which are aimed at determining high-dimensional designs: the approximate coordinate exchange (ACE) algorithm (Overstall and Woods, 2017) and the induced natural selection heuristic (INSH) algorithm (Price et al., 2018). The ACE algorithm can be implemented using the R package `acebayes` (Overstall et al., 2017). R code for the INSH algorithm has been provided to us by the authors. All code for the examples in this paper can be found at `https://github.com/csgillespie/expt_design`.

2 Efficiency improvements to the algorithm

We now describe a new algorithm which also uses a particle-based approach but is one that is much more straightforward, makes more efficient use of evaluations of the (expensive) utility function and also more efficiently identifies near-optimal designs. Essentially the approach bases particle weights on current estimates of expected utility and thereby focuses sampling effort around near-optimal designs.

Consider the general case where we need an optimal k-timepoint design $\boldsymbol{d} = (t_1, \ldots, t_k)$, where the times t_i lie on a grid rather than in a continuous interval (as in the Müller, Amzal and ACE algorithms). This restriction reflects the practical nature of experimentation but, of course, near continuous designs may be found by using a grid with a fine mesh. The algorithm uses refinements

of a particle distribution to increasingly focus on designs around the optimal design. Instead of basing the weights on increasing powers of the utility function, in step m we focus on designs with (current estimated) expected utility values in the upper $100\alpha_m\%$ of their distribution. The algorithm targets near optimal designs by working through a series of steps in which the α_m–values decrease as the step number m increases. In this paper we take $\alpha_m = 2^{-m}$. The powering-up technique used by Müller (1999) and Amzal et al. (2006) is their way of focusing on designs which are near optimal. Although we could use a similar technique which calculates weights by powering-up current estimates of the expected utility, we feel that it is more natural and intuitive to deal directly with the size of the upper tail of the distribution of the expected utility (over designs).

Suppose the location of a design on the discretised mesh of timepoints is defined by the coordinate system $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_k)$ for $\boldsymbol{\ell} \in \text{Ł}$ so that the design space is $\mathcal{D} = \{\boldsymbol{d}_{\boldsymbol{\ell}} : \ \boldsymbol{\ell} \in \text{Ł}\}$. The algorithm works with a (discrete) k-dimensional categorical distribution in which the design at location $\boldsymbol{\ell}$ has probability $p(\boldsymbol{\ell}) = w_{\boldsymbol{\ell}}$. This distribution is initialised to be a discrete uniform $\text{Cat}_k(\boldsymbol{w}^0)$ distribution, with un-normalised weights $w^0_{\boldsymbol{\ell}} = 1$, and mass function $p(\boldsymbol{\ell}) \propto w^0_{\boldsymbol{\ell}}, \ \boldsymbol{\ell} \in \text{Ł}$. This choice is an exchangeable one and reflects the inability to choose between designs initially. The initialisation continues by taking a random sample of design locations from the $\text{Cat}_k(\boldsymbol{w}^0)$ distribution, simulating datasets from the prior predictive distribution at these designs, and then determining the utility function at these design/data choices. These utility calculations are then used to initialise the estimate of the expected utility $\hat{u}(\boldsymbol{d}_{\boldsymbol{\ell}}) = \text{mean}(u_{\boldsymbol{\ell}_k} : \ \boldsymbol{\ell}_k = \boldsymbol{\ell})$ and the number of utility calculations contributing to these means $n_{\boldsymbol{\ell}} = \#(\boldsymbol{\ell}_k = \boldsymbol{\ell})$. Note that any designs not visited during the initialisation are given zero expected utility. These estimates of expected utilities are then used to construct the particle distribution for the first $m = 1$ step, $\text{Cat}_k(\boldsymbol{w}^1)$, by taking $\boldsymbol{w}^1_{\boldsymbol{\ell}} = \hat{u}(\boldsymbol{d}_{\boldsymbol{\ell}})$. Note that this choice is only sensible if all utility estimates are positive. However, if this is not the case, as is likely when using say the logged generalised precision utility function, then we have found using un-normalised weights $w_{\boldsymbol{\ell}} = u_{\boldsymbol{\ell}} - \min_{\boldsymbol{\ell}' \in \text{Ł}} u_{\boldsymbol{\ell}'}$ works well.

The algorithm then goes through a sequence of steps $m = 1, 2, \ldots, M$ which operate in a similar way to the initialisation. One potential issue is that, given the size of the design space, it is possible that many near-optimal designs are not selected and so it is not prudent to give these designs zero weight in the particle distribution of the next step. We circumvent this issue by perturbing the sampled locations using a distribution q which, for example, might move a design to one of its 2^k neighbouring locations, and thereby reach local near-optimal designs. More generally we can use a random walk proposal such as $q(\boldsymbol{d}_{\boldsymbol{\ell}^*}|\boldsymbol{d}_{\boldsymbol{\ell}}) = \prod_{i=1}^k q_i(\ell_i^*|\ell_i)$ over design locations, where each $q_i(\ell_i^*|\ell_i)$ is a symmetric univariate random walk proposal, to tailor the size of the local search and help to reach and stay fairly close to near-optimal designs. One proposal we have found to work well is to take $\ell_i^* = \ell_i + \nu_i$ where the ν_i are independent and have a distribution which is the difference between two independent Poisson random variables, each with mean λ. Utilities are then calculated by first simulating datasets from the prior predictive distribution at these design locations and these utilities used to update the particle weights $\boldsymbol{w}^m$ to contain the (estimated) expected utility for those designs $\boldsymbol{d}_{\boldsymbol{\ell}}$ in the top $100\alpha_m\%$ of the expected utility distribution, with $\boldsymbol{w}^m_{\boldsymbol{\ell}} = 0$ for all other designs. In the final step, we have found it useful to increase the accuracy of estimated expected utility at designs currently thought to be near-optimal, that is, at designs $\boldsymbol{d}_{\boldsymbol{\ell}}$ with $w^M_{\boldsymbol{\ell}} \neq 0$, through more utility evaluations. In other words, the final step proceeds essentially the same as in the previous steps but without any perturbation around the selected designs. Finally, after completing all M steps, we take the optimal design as $\boldsymbol{d}^* = \boldsymbol{d}_{\boldsymbol{\ell}^*}$, where $\boldsymbol{\ell}^* = \arg\max_{\boldsymbol{\ell}} w^M_{\boldsymbol{\ell}}$, or conduct a final more intensive search around this putative optimal design.

The Müller, Amzal and ACE algorithms do not make use of utility calculations made at design locations in previous steps. Thus a simple but productive efficiency improvement can be made by using all utility calculations made in previous steps of the algorithm. This is easily done by keeping a running average of the utility calculations at each design location. Additional improvements can be made by using all computer cores available to the user. For example, if C cores are available,

Algorithm 3 Step m of new algorithm.

1: **for** $i = 1$ to N_m **do (in parallel)**
2: Update un-normalised weights $\boldsymbol{w}^m$ to contain
top $100\alpha_m\%$ values of $\{\hat{u}(\boldsymbol{d}_{\boldsymbol{\ell}}) : n_{\boldsymbol{\ell}} > 0\}$
3: Simulate $\boldsymbol{\ell}^* \sim \text{Cat}(\boldsymbol{w}^m)$
4: Simulate $\boldsymbol{\ell} \sim q(\boldsymbol{\ell}|\boldsymbol{\ell}^*)$ and $\boldsymbol{y} \sim \pi(\boldsymbol{y}|\boldsymbol{d}_{\boldsymbol{\ell}})$
5: Calculate $u(\boldsymbol{d}_{\boldsymbol{\ell}}, \boldsymbol{y})$
6: Update expected utility
$\hat{u}(\boldsymbol{d}_{\boldsymbol{\ell}}) \leftarrow \{u(\boldsymbol{d}_{\boldsymbol{\ell}}, \boldsymbol{y}) + n_{\boldsymbol{\ell}}\hat{u}(\boldsymbol{d}_{\boldsymbol{\ell}})\}/\{n_{\boldsymbol{\ell}} + 1\}$
7: Update count $n_{\boldsymbol{\ell}} \leftarrow n_{\boldsymbol{\ell}} + 1$
8: **end**
9: $\boldsymbol{w}^{m+1} = \boldsymbol{w}^m$

there is little additional time penalty in calculating C utilities $u(\boldsymbol{d}_{\boldsymbol{\ell}_k}, \boldsymbol{y}_k)$ rather than just one. Finally, another efficiency gain may be achieved by using a different run length N_m in each step, with the earlier steps perhaps using longer runs. However, the rate at which the N_m decrease should depend on the extent to which reducing α_m identifies a clear optimal design. A summary of the new algorithm is given in Algorithm 3. Although updating the weights could be left until the next step of the algorithm, we have found it beneficial to update the current weights $\boldsymbol{w}^m$ regularly, leading to line 2 being within the main loop.

In general, the initial uniform $Cat_k(\boldsymbol{w}^0)$ distribution over design locations will work reasonably well so long as the number of possible designs $|\mathcal{D}|$ is not too large. However, if $|\mathcal{D}|$ is large, for example when using a fairly fine time-grid and seeking a design with a moderate number of timepoints, making sure that the algorithm visits all near-optimal designs in step 1 could become problematic. That said, we have found that expanding the reach of the local random walk proposals deals with this issue quite well, though this inevitably leads to needing a larger number of iterations N_m.

3 Examples

We demonstrate the efficiency of our method by determining optimal designs of different sizes in four scenarios. We begin by studying the death model considered by Cook et al. (2008) and Drovandi and Pettitt (2013). This simple model has a tractable likelihood and so calculation of the posterior variance, and hence the utility function, is straightforward. We consider in detail the case of determining an optimal single timepoint and compare the accuracy of our new method with those of other popular methods. We then look at finding an optimal two timepoint design for an oscillatory system typical of those commonly found when modelling biological systems. The oscillations in this system induce multiple modes in the expected utility and this complicates the search for optimal designs. We compare the performance of our algorithm with the ACE algorithm in determining this optimal two timepoint design. We then compare performances in finding 15-dimensional optimal designs using a toy utility function which has features typical of those in real design problems. Finally we consider optimal design for a more complex stochastic model of aphid growth (Matis et al., 2007). Here we calculate the posterior variance using a moment closure approximation to the stochastic model and determine optimal designs of different sizes using multi-core parallel computing, enabling a six-fold speed-up.

In the following examples, our algorithm uses threshold $\alpha_m = 2^{-m}$ in step m and (except where stated otherwise) spreads the number of utility evaluations equally between the initialisation and the steps, that is, uses $N_m = N/(M + 1)$ particles in the initialisation and in each step. Also selected designs are perturbed by (independent) random numbers of grid points (in each dimension), each

calculated as the difference between two independent Poisson random variates with mean $\lambda = 4$ (except where stated otherwise). Runs of the ACE algorithm are made using the `acebayes` package with (default) parameters close to $B = (200, 19)$, $N_1 = 20$ and $N_2 = 0$ (suggested by the authors) – the actual values used were modified slightly to match the computational budget in each example. The INSH algorithm depends on many more parameters and we use those provided by the authors and found in their R code at `https://github.com/csgillespie/expt_design`.

3.1 Death model

Cook et al. (2008) describe a death model in which the size $Y(t)$ of the population at time t obeys the probabilistic law: in the small time period $(t, t + \delta t]$, the probability of a death is $Pr\{Y(t + \delta t) = i - 1 | Y(t) = y(t)\} = \beta y(t)\, \delta t + o(\delta t)$, otherwise no death occurs. If the population is initialised with size $n = Y(t_0 = 0)$ and then observed at times $t_1, \ldots, t_k$, the likelihood is formed from terms $Y(t_k)|Y(t_{k-1}) = y_{k-1} \sim Bin\{y_{k-1}, \exp[-\beta(t_k - t_{k-1})]\}$.

Suppose interest lies in determining the optimal k-timepoint design $\boldsymbol{d} = (t_1, \ldots, t_k)$ using the posterior precision of β as our utility function, that is, $u(\boldsymbol{d}, \boldsymbol{y}) = 1/\mathrm{Var}(\beta|\boldsymbol{y})$. One considerable benefit of studying this model is that values of the expected utility can be calculated and there is no need to employ a stochastic algorithm such as an MCMC algorithm or importance sampling. It is fairly quick to calculate $K_i(\boldsymbol{y}) = \int \beta^i \pi(\boldsymbol{y}|\beta)\pi(\beta)d\beta$, $i = 0, 1, 2$ over all possible $\boldsymbol{y} = (y_{t_1}, \ldots, y_{t_k})$ using the GSL library (Galassi et al., 1996). Thus we can calculate the expected utility over all possible designs $\boldsymbol{d}$ using

$$\begin{aligned} u(\boldsymbol{d}) &= \sum_{y_{t_1} \geq \cdots \geq y_{t_k} = 0}^{n} \pi(\boldsymbol{y}) u(\boldsymbol{d}, \boldsymbol{y}) \\ &= \sum_{y_{t_1} \geq \cdots \geq y_{t_k} = 0}^{n} \frac{K_0(\boldsymbol{y})^3}{K_2(\boldsymbol{y})K_0(\boldsymbol{y}) - K_1(\boldsymbol{y})^2}, \end{aligned} \tag{2}$$

where n is the initial population size, and thereby determine the optimal k-timepoint design $\boldsymbol{d}^*$.

We follow Drovandi and Pettitt (2013) by considering an experimental period over $(0, T = 10)$ and restrict design timepoints to be on the grid $t = 0.01(0.01)10$. The initial population size is fixed to be $n = 50$. We also take their log-normal $\mathrm{LN}(-0.005, 0.01)$ prior distribution for β and focus on determining the optimal single ($k = 1$) timepoint design. We also use their utility function (1). This problem is sufficiently simple that it is possible to calculate the expected utility for each possible single timepoint design and determine that the optimal single timepoint design is $\boldsymbol{d}^* = t_1^* = 1.61$.

Fig. 1 shows how the new algorithm increasingly focuses on getting ever more accurate estimates of near optimal expected utilities (by averaging over more realisations from the prior predictive distribution) over the initialisation and steps $m = 1, 2, 3, 4$ of the algorithm, with $\alpha_m = 2^{-m}$. The plot for the initialisation shows the (near) uniform coverage over all possible timepoints and then, as m increases, more and more realisations are simulated at near-optimal timepoints. After the $m = 4$ step, the estimate of expected utility at $t_1^* = 1.61$ is an average over 160 realisations of $u(t_1^* = 1.61, y)$.

Fig. 2 gives a comparison of the performance of the Müller, Amzal, ACE and our new algorithms by showing the sampling distribution of the optimal designs they return over 500 independent runs of each algorithm. Note that we give results for two sets of ACE parameters, one being the default choice and the other a special choice for this model given to us by the authors of ACE. The actual values used can be found in our code at `https://github.com/csgillespie/expt_design`.
Each run of each algorithm uses the same computational budget of 24K utility evaluations and takes approximately the same run time. Also each algorithm was run a single CPU core. However, as mentioned previously, if more cores are available then our new algorithm scales trivially with the number of cores.

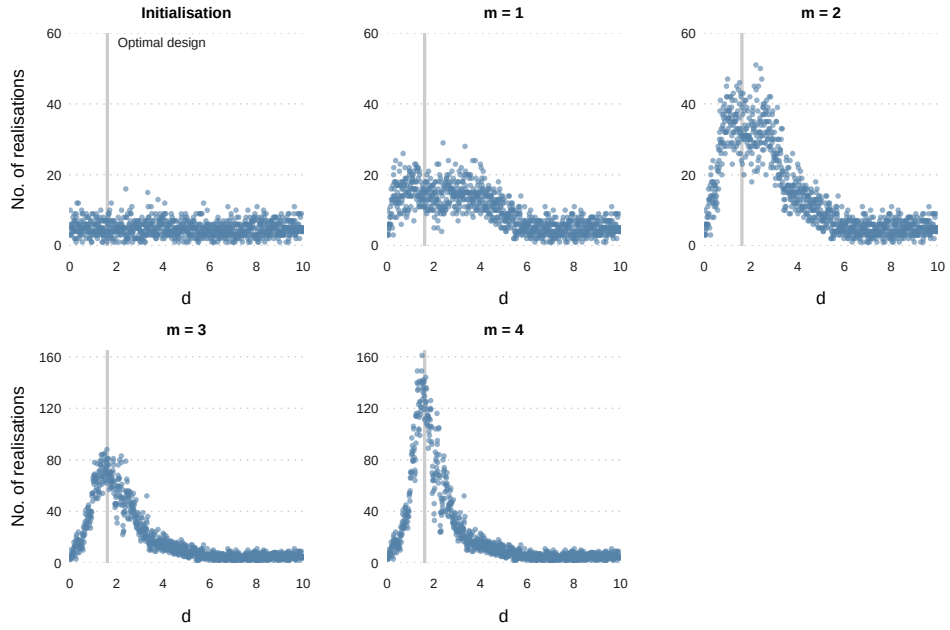

Figure 1: Graphs showing the number of realisations $u(d, y_i)$ contributing to the estimate of $u(d)$ at each d as the number of steps m increases. The vertical grey line shows the optimal design ($d^* = 1.61$).

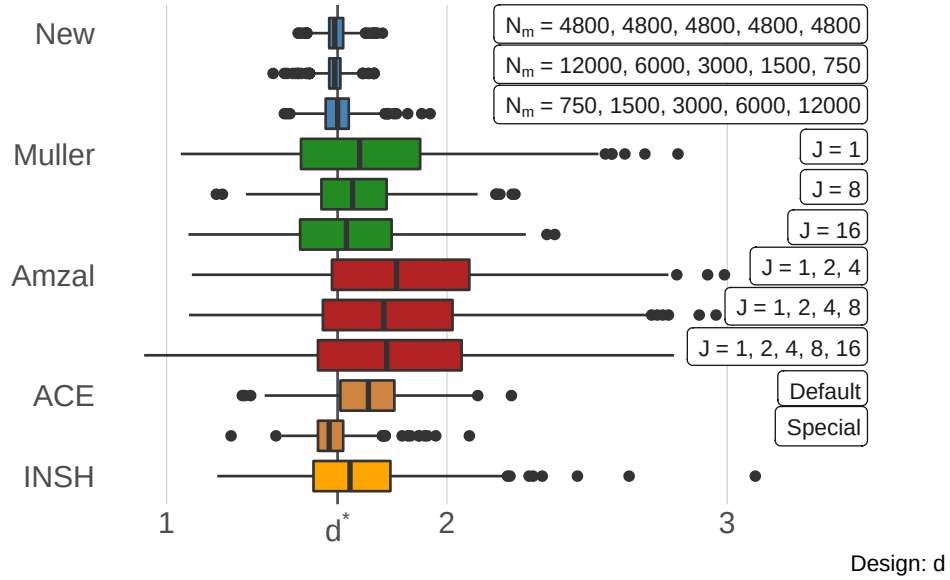

Figure 2: Box plots of the optimal designs returned by each algorithm from 500 independent runs. The correct optimal design is $d^* = 1.61$. The different cases are (a) New algorithm with $\alpha_m = 2^{-m}$ for (i) $N_m = 4800, 4800, 4800, 4800, 4800$; (ii) $N_m = 12000, 6000, 3000, 1500, 750$; (iii) $N_m = 750, 1500, 3000, 6000, 12000$. (b) Müller algorithm for $J = 1, 8, 16$. (c) Amzal algorithm for (i) $N_m = 2400$, $J_m = 1, 2, 4$; (ii) $N_m = 1090$, $J_m = 1, 2, 4, 8$; (iii) $N_m = 522$, $J_m = 1, 2, 4, 8, 16$. (d) Approximate coordinate exchange (ACE) algorithm using default and special settings. (e) Induced natural selection heuristic (INSH) algorithm.

The top three boxplots in Fig. 2 summarise the results for the new algorithm but with different a breakdown of the 24K utility evaluations in the initialisation and steps $m = 1, 2, 3, 4$. Overall the results show that the algorithm is fairly insensitive to the number (N_m) of evaluations in

each step, though having the N_m decreasing in m appears to work best. Also the boxplots are tightly centered around the optimal design ($t_1^* = 1.61$). The next three boxplots are for the Müller algorithm with $J = 1, 8, 16$. It is clear that the spread of the Müller solutions is much larger than for the new algorithm. Another issue is that choosing an appropriate value for J would require tuning and therefore additional utility evaluations. The next three boxplots are for the Amzal algorithm using different powering up schemes with different numbers of steps (but still a total of 24K utility evaluations). Given the sophistication of the Amzal scheme it is surprising to see that its performance is similar to the Müller algorithm with $J = 1$. Inspection of the Amzal algorithm reveals that this is mainly due to the number of utility evaluations needed within the main loop (Algorithm 2, line 9). The bottom three boxplots show the results from the ACE and INSH algorithms. The variation in solutions from ACE is smaller than that of Müller, Amzal and INSH, and the INSH solutions are slightly worse than the Müller solutions using $J = 8$. Perhaps a better overall objective measure of algorithm performance is their square root mean squared error (RMSE) about the correct solution $t_1^* = 1.61$. The values for these various implementations (in the order top-bottom in Fig. 2) is New: 0.04, 0.07, 0.04, Müller: 0.33, 0.18, 0.24, Amzal: 0.43, 0.41, 0.44, ACE: 0.08, 0.17, INSH: 0.23 and show that the new algorithm can perform between 2.5 and 11 times more efficiently. An additional and powerful attribute of the new algorithm is that it is a much more simple algorithm to implement than the others and is embarrassingly parallel.

There is a indirect relationship between decreasing α_m in the new algorithm and Müller's powering-up approach. Consider the normalised utility values $w_i^J = u(d_i)/\sum_{i'=1}^{|\mathcal{D}|} u^J(d_{i'})$, with order statistics $w_{(1)}^J \leq \cdots \leq w_{(|\mathcal{D}|)}^J$ and empirical distribution function $\hat{F}_J(\cdot)$. A measure of the correspondence between the new algorithm and its Müller equivalent is the value of k, where

$$\arg\min_k \hat{F}_J\{w_{(k)}^J\} > 1 - \alpha_m, \tag{3}$$

and α_m is the threshold used in step m of the new algorithm. The distribution of expected utility becomes increasingly peaked as $J \to \infty$, in which case $w_{(|\mathcal{D}|)}^J \to 1$, that is, $k \to |\mathcal{D}|$. Also during the $m = 1$ step of the new algorithm, designs are sampled from a distribution with weights proportional to (an estimate of) $u(d)$ and so this corresponds to the Müller algorithm with $J = 1$. In the Müller algorithm, as the expected utility is powered-up (by increasing J) the algorithm preferentially sample designs near the optimal design, that is, in the upper tail of the expected utility distribution. Unfortunately it is difficult to obtain an algebraic understanding of how increasing J focuses on designs further into the upper tail of the distribution of expected utility values, that is, its effect on k. However we can calculate the (exact) expected utility (for this simple death model) at all 1000 single timepoint designs in the design space using (2) and thereby determine values of k for different choices of J for various values of α_m; see Fig. 3. For example, when $\alpha_m = 0.5$ and $J = 1$, we obtain $k = 454$ as the sum of the largest 454 normalised utility values is greater than 0.5 (but that of the largest 453 is not). We see that k decreases as J increases (for fixed α_m). However for $J > 20$ the change in k drastically slows down. Furthermore, using values of $J > 100$, results in numerical issues. Also, although the traces in k for different α_m look parallel, they are only roughly parallel with, for example, $k_{\alpha=0.5}/k_{\alpha=0.25}$ ranging between 2.0 and 2.3. The figure highlights the main issue with the powering up approach: whilst being an intuitively good idea, in practice the rate at which this homes in on the optimal design as J increases is unclear and difficult to predict. Also working with large powers of utilities often introduces problems of numerical instability.

3.2 Oscillatory systems

Solutions to deterministic or stochastic descriptions of biological systems often display oscillatory behaviour, particularly those describing homoeostasis maintained by regulatory mechanisms. Here we consider a toy model which exhibits this behaviour but also has a straightforward conjugate

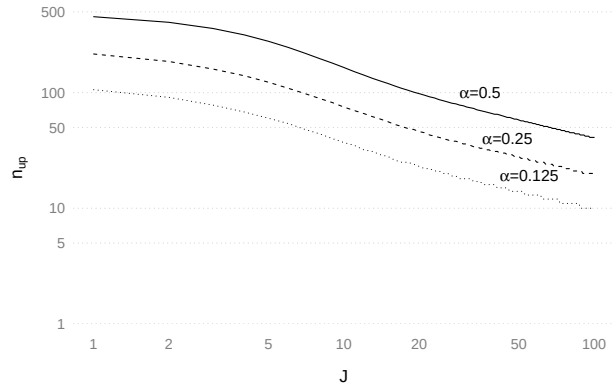

Figure 3: Relationship between the level of powering up (J) of the utility function in the Müller algorithm and, for the new algorithm, the number of designs (n_{up}) in the upper tail of the expected utility distribution, calculated over all 1000 single timepoint designs for the death model.

Bayesian analysis. The model is one of damped oscillations where observations follow

$$y_t = \theta e^{-t} \sin 6\pi t + \varepsilon_t, \tag{4}$$

with $\varepsilon_t \overset{indep}{\sim} \mathrm{N}(0, \sigma^2)$ and $t \in [0, T = 1]$. The (unknown) parameters are the size of the oscillations (described by θ) and the level of observational noise (σ). A key feature of this model is that multiple cycles can be observed within the observed time period.

If this observational model is rewritten as $y_t = \theta f(t) + \varepsilon_t$ then it is clear that it is a simple normal regression model. Such models have a conjugate normal-gamma prior distribution, which in this case is $\theta|\sigma \sim \mathrm{N}(b, \sigma^2/c)$ and $\sigma^{-2} \sim \mathrm{Ga}(g, h)$. The posterior distribution after observing data $\boldsymbol{y}$ from a design with k timepoints takes the same form with $\theta|\sigma, \boldsymbol{y} \sim \mathrm{N}(B, \sigma^2/C)$ and $\sigma^{-2}|\boldsymbol{y} \sim \mathrm{Ga}(G, H)$, where $B = (bc+P)/C$, $C = c+Q$, $G = g+k/2$, $H = h+[\sum_{i=1}^{k}\{y_{t_i} - Pf(t_i)/Q\}^2 + b^2c + P^2c/(QC)]/2$ with $P = \sum_{i=1}^{k} f(t_i)y_{t_i}$ and $Q = \sum_{i=1}^{k} f(t_i)^2$. It is easily shown that $Cov(\theta, \sigma^2|\boldsymbol{y}) = 0$ and so the posterior generalised precision for (θ, σ^2) is $C(G-1)^3(G-2)/H^3$. Therefore, as G does not depend on data values, we take our utility function as the logged generalised precision $u(\boldsymbol{d}, \boldsymbol{y}) = \log C - 3\log H$.

We now compare the performance of our algorithm with that of ACE in determining the optimal $d = 2$ time-point design. We construct the prior distribution so that values of θ and σ are typically around ten and one respectively but fairly uncertain ($b = 10$, $c = 0.01$, $g = 3$ and $h = 3$). Figure 4 shows typical realisations from the prior predictive distribution, together with the prior predictive mean trace. We will again assume a computational budget of 24K utility evaluations for each algorithm. For our algorithm we use $N_m = 2.4$K particles in the initialisation and in each step $m = 1, 2, \ldots, 9$ and search for optimal designs on a grid $t = 0(0.002)1$. Random walk perturbations were made using $\lambda = 4$ except in the final step ($\lambda = 0$).

Figure 5 shows the optimal designs obtained from 500 runs of each algorithm. It also shows the multi-modal nature of the underlying expected utility surface. Clearly the ACE algorithm gets stuck in local modes whereas our algorithm hits only local modes with the correct optimal design (two replicates at time $t = 0.082$) occurring on 48% of occasions. The performance of the ACE algorithm is perhaps not surprising as it initialises at a random design and works best with uni-modal expected utility surfaces (for high dimensional designs). Although ACE is likely to perform much better with more utility evaluations, we did find that the ACE solutions were similarly scattered after using 48K evaluations, whereas our algorithm found the correct optimal design 96% of the time.

3.3 High dimensional designs

The Müller and Amzal algorithms are not computationally efficient when determining high dimensional designs. However, the ACE algorithm has been designed to solve this problem in a

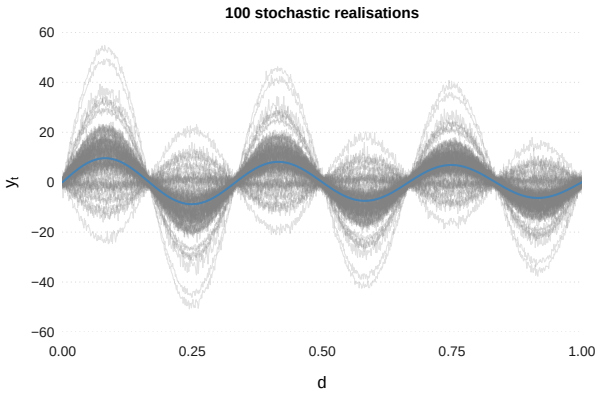

Figure 4: 100 prior predictive realisations from the oscillatory system model, together with the prior predictive mean trace.

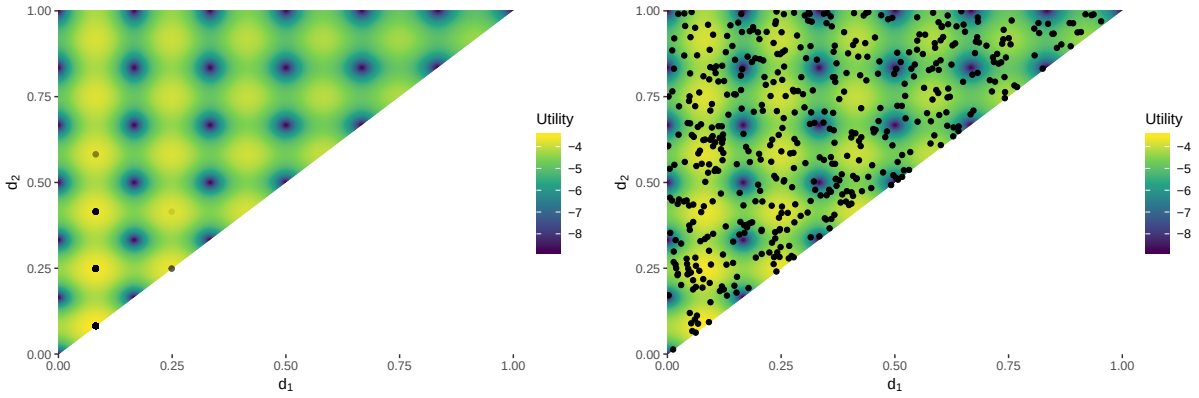

Figure 5: 500 optimal designs determine by our new algorithm (top) and the ACE algorithm (bottom), together with contours of the expected utility surface.

computationally efficient way. We now compare the performance of the ACE algorithm and our algorithm in determining a $d = 15$ time-point design. As we need to run the algorithms many times we have chosen to examine the performance of these algorithms using a utility function which is quick to evaluate but also has features seen in real problems, namely that it is unimodel and fairly flat. We base our utility function on a 15-dimensional normal density with mean vector $\boldsymbol{\mu} = (0.5, 1.5, \ldots, 14.5)^T$ and covariance matrix $\Sigma = 10I_{15}$, where I_{15} is the 15×15 identity matrix. Specifically we take the utility function for data $\boldsymbol{y}_i$ to be

$$u(\boldsymbol{d}, \boldsymbol{y}_i) = \exp\left\{-(\boldsymbol{d} - \boldsymbol{\mu})^T(\boldsymbol{d} - \boldsymbol{\mu})/20\right\}\, \varepsilon_i$$

where $\varepsilon_i \overset{indep}{\sim} \text{LN}(0, 0.03^2)$. Note that this scaling of the normal density gives it a maximum value of one. Figure 6 displays the median utility function and utility realisations for the design with middle time-point $d_8 = 6.5(0.01)8.5$ and other time-points at their optimal choice ($d_i = \mu_i$, $i \neq 8$). The figure shows a typical view of the median utility function: it is unimodal, fairly flat and its realisations are quite noisy.

We now look for the optimal 15 time-point design in the observational period $[0, T = 15]$ using a fairly fine grid $t = 0(0.01)15$. We compare the algorithms assuming a computational budget of 360K utility evaluations. For our algorithm, after the initialisation we move through steps $m = 1, 2, \ldots, 14$ with $N_m = 24$K particles in each step using random walk perturbations with $\lambda = 1$, except in the final step ($\lambda = 0$). The algorithms were run on a single CPU and the computational time taken for our algorithm was around ten times that for ACE. This is because our algorithm is

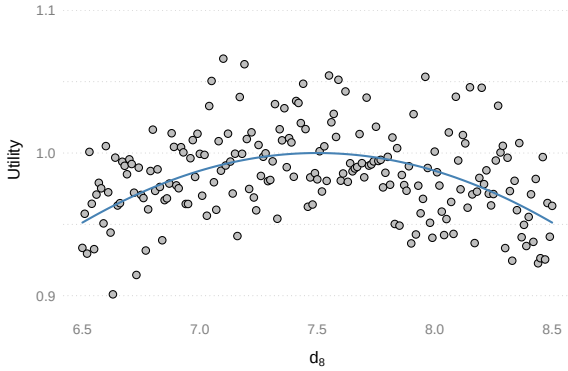

Figure 6: The median utility function and its realisations at the 15-dimensional design with middle time-point $d_8 = 6.5(0.01)8.5$ and other time-points at their optimal choice ($d_i = \mu_i$, $i \neq 8$).

designed to sacrifice storage for speed and, in particular, retrieving past results can be (relatively) time-consuming. However employing ten CPUs in parallel would give the algorithms a similar duration. Also, for more realistic models, the utility evaluation would be significantly longer and so running our algorithm in parallel would be quite beneficial, if not essential.

The RMSE of 100 algorithm solutions about the correct optimal design $\boldsymbol{d}^* = \boldsymbol{\mu}$ is New: 0.01 and ACE: 0.001, with the best expected utility found in these solutions being New: 0.98 and ACE: 0.9997 to be judged against the maximum achievable expected utility $u(\boldsymbol{d}^*) = 1$. Clearly ACE out-performs our algorithm in determining the 15-d optimal design. However our algorithm performs reasonably well.

3.4 Cotton aphids

A cotton aphid infestation of a cotton plant can result in many problems such as leaves that curl and pucker, seedling plants become stunted and may die, a late season infestation can result in stained cotton. Also cotton aphids have developed resistance to many chemical treatments and so can be difficult to treat. Therefore considerable effort and cost is used in the maintenance of cotton plants. Matis et al. (2007) have developed a stochastic model of aphid population growth. Gillespie and Golightly (2010) give a Bayesian analysis of this model using data given in Matis et al. (2008). The data contain aphid counts on twenty randomly chosen leaves in each plot, for twenty-seven treatment-block combinations. The treatment-blocks were formed from three three-level factors (nitrogen and irrigation levels and block). Observations were taken roughly every 7 to 8 days within a 32 day period.

Let $N(t)$ and $C(t)$ denote the size and cumulative size of the aphid population respectively at time t. Matis et al. (2007) modelled aphid dynamics using a birth rate of $\lambda N(t)$ and a death rate of $\mu N(t)C(t)$. Therefore, in a small time period $(t, t+\delta t]$, so that at most one event can occur, we have

$$Pr\{N(t+\delta t) = n(t)+1, C(t+\delta t) = c(t)+1 | n(t), c(t)\} = \lambda n(t)\,\delta t + o(\delta t),$$
$$Pr\{N(t+\delta t) = n(t)-1, C(t+\delta t) = c(t) | n(t), c(t)\} = \mu n(t) c(t)\,\delta t + o(\delta t),$$

and the probability of staying in the same state is one minus the sum of these probabilities.

Gillespie and Golightly (2010) analysed these data by making a normal approximation to the stochastic model using moment closure. This gives transition distributions

$$(N(t_i), C(t_i))^T | N(t_{i-1}), C(t_{i-1}), \lambda, \mu$$

which follow $N_2\{\boldsymbol{m}(t_{i-1}), \boldsymbol{v}(t_{i-1})\}$ distributions, where $\boldsymbol{m}(t) = (m_1(t) = \mathrm{E}\{N(t)\},\ m_2(t) = \mathrm{E}\{C(t)\})^T$ and $\boldsymbol{v}(t)$ is a matrix with diagonal elements $v_{11}(t) = \mathrm{Var}\{N(t)\}$ and $v_{22}(t) = \mathrm{Var}\{C(t)\}$, and off-diagonal elements $v_{12}(t) = \mathrm{Cov}\{N(t), C(t)\}$. Note that the (λ, μ)-dependence of these functions has been suppressed to simplify the exposition. These functions are determined as solutions of the ODE system

$$\begin{aligned} \frac{dm_1(t)}{dt} &= \lambda m_1(t) - \mu\{m_1(t)m_2(t) + v_{12}(t)\} \\ \frac{dm_2(t)}{dt} &= \lambda m_1(t) \end{aligned} \tag{5}$$

$$\begin{aligned} \frac{dv_{11}(t)}{dt} &= \mu[v_{12}(t) - 2m_1(t)v_{12}(t) - 2\kappa_{21} + m_2(t)\{m_1(t) - 2v_{11}(t)\}] \\ &\qquad + \lambda\{m_1(t) + 2v_{11}(t)\} \\ \frac{dv_{12}(t)}{dt} &= \lambda\{m_1(t) + v_{11}(t) + v_{12}(t)\} - \mu\{m_1(t)v_{22}(t) + m_2(t)v_{12}(t)\} \\ \frac{dv_{22}(t)}{dt} &= \lambda\{m_1(t) + 2v_{12}(t)\}, \end{aligned} \tag{6}$$

with initial conditions $m_1(0) = n_0$, $m_2(0) = c_0$ and $v_{11}(0) = v_{12}(0) = v_{22}(t) = 0$. This system can be solved numerically using standard ODE solvers. Note that this approximation is the same as that when using the linear noise approximation if the term $v_{12}(t)$ in (5) is ignored.

We now construct optimal designs assuming a similar treatment-block pattern and observation period. The aim is to optimise the posterior generalised precision for the rate parameters $\boldsymbol{\theta} = (\lambda, \mu)$. Gillespie and Golightly (2010) found only small differences in the rate parameters for the various treatments and blocks and so we base our prior distribution on that of the posterior distribution for the base treatment and block rates, however we inflate prior uncertainty by increasing the variances by a factor of ten, giving

$$\begin{pmatrix} \lambda \\ \mu \end{pmatrix} \sim N_2\left\{ \begin{pmatrix} 0.246 \\ 0.000134 \end{pmatrix}, \begin{pmatrix} 0.0079^2 & 5.8 \times 10^{-8} \\ 5.8 \times 10^{-8} & 0.00002^2 \end{pmatrix} \right\}.$$

Also to simplify the analysis, we assume known initial aphid levels ($n_0 = c_0 = 28$) as there was very little variability in these quantities in the Matis et al. (2008) dataset.

One hundred simulations from the prior predictive are shown in Fig. 7. All simulations ultimately end in the extinction of the aphid population, since the cumulative aphid population is increasing with each birth, resulting in $\mu n(t)c(t) > \lambda n(t)$ as t increases. All simulations show a peak in aphid population around 20–25 days.

The posterior density for the rate parameters when using a design with k timepoints yielding data $\boldsymbol{y}$ is

$$\pi(\lambda, \mu|\boldsymbol{y}) \propto \pi(\lambda, \mu) \prod_{i=1}^{k} \phi_2\{n(t_i), c(t_i)|\boldsymbol{m}(t_{i-1}), \boldsymbol{v}(t_{i-1})\},$$

where $\phi_2(\cdot, \cdot|\boldsymbol{m}, \boldsymbol{v})$ is the $N_2(\boldsymbol{m}, \boldsymbol{v})$ density. Realisations from this posterior were obtained using an MCMC scheme with a bivariate normal random walk proposal, centred at the current value and a covariance matrix with standard deviations 0.0009 and 0.000004, and correlation equal to the prior correlation. We seek optimal $k = 1, 2, 3, 4$ timepoint designs within a $T = 49$ day period. Note that this period is slightly longer than that used in the original experiment so we can investigate whether the experiment was stopped too early. Thus the design timepoints are in units of 24 hours (called days) after the start of the new experiment (day zero). Again each posterior distribution was determined by initialising the chain at the parameter values used to simulate the responses $\boldsymbol{y}$ and

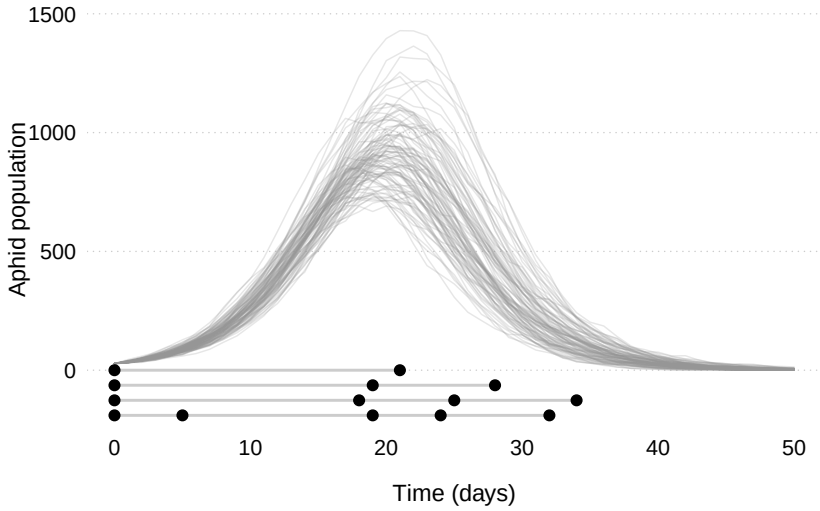

Figure 7: 100 prior predictive realisations together with the optimal one, two, three and four timepoint designs.

we found that very little burn-in was necessary. Each time the algorithm was run for 10K iterations, and typically took no more than two cpu seconds on an Intel Core i7-6700 CPU.

Again we determine the optimal design using the predictive precision utility function (1). We look for one, two, three and four–timepoint optimal designs by moving through steps $m = 1, 2, \ldots, 8$ after initialisation. We use a computational budget of 432K utility evaluations (48K in each of the initialisation and each step) and use random perturbations with $\lambda = 4$ except in the final step ($\lambda = 0$).

Fig. 8 shows the top 100 designs (by estimated expected utility) with 1, 2, 3, and 4 timepoints together with (rough) 95% confidence bounds calculated assuming asymptotic posterior normality. Note that there are only 50 possible single timepoint designs. One of the benefits of using our algorithm is that it is straightforward to keep track of the precision of the current expected utility estimates by also keeping a running total of the $u(\boldsymbol{d}, \boldsymbol{y}_i)^2$. The figure shows that the confidence bounds for the one and two time point designs are very small. The intervals for the three point design are also relatively narrow. However, the optimal design is not clear and further runs will be needed to reduce uncertainty on the expected utility estimates. There is much more uncertainty in expected utility estimates for the four design point problem and clearly many more simulations will be needed to pick out the optimal design. All that said, the differences in expected utility for all top 100 designs (of a given size) are quite small.

One advantage of our new algorithm (over Müller and Amzal) is that it is trivial to extend a search for an optimal design so long as the particle weights are retained from the previous run. A further advantage is that any extension to the optimal search run is not restricted to being on the same computer resource (desktop, cluster, cloud) as the initial run. This can be advantageous when determining the optimal design for a complex model which has a time consuming utility calculation as decisions on numbers of particles and their location can be amended easily (by the user) at each step of the algorithm. Also inspection of the (estimated) expected utilities can be examined at each step of the algorithm to determine how many steps are needed before the location of the optimal design is clear. For example, Fig. 9 shows the top 100 three timepoint designs for the initialsation and steps $m = 1, 2, 3$ and shows that uncertainty around the top designs is sufficiently small that there is no need to extend the algorithm to another step to determine the top five designs in contention for being optimal design.

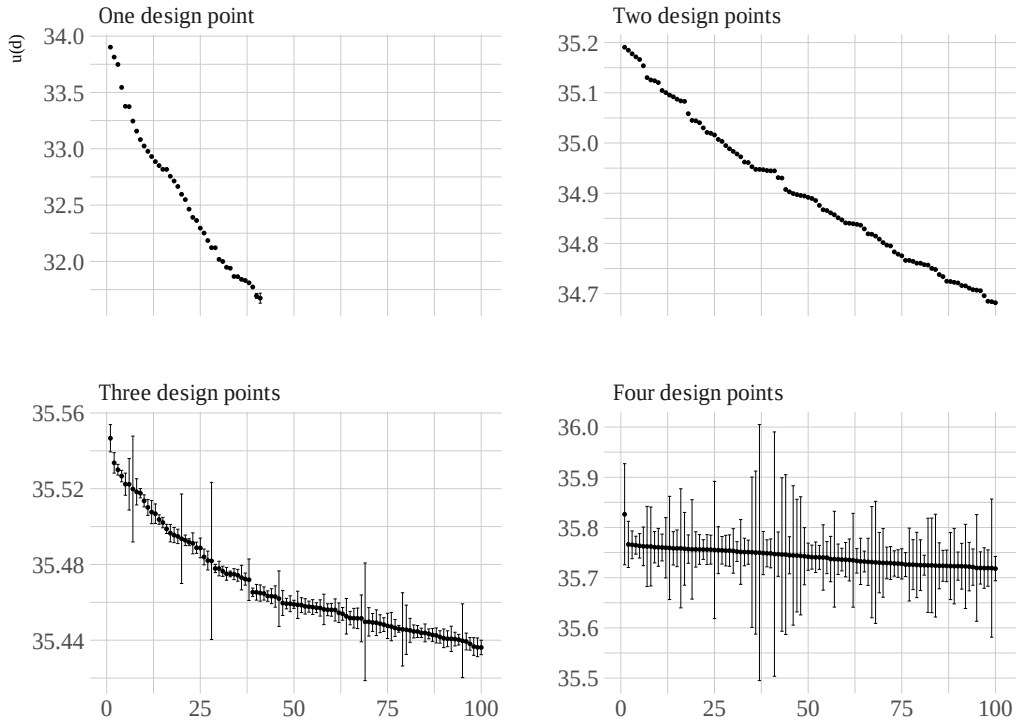

Figure 8: Estimated expected utilities (and central 95% intervals) for the top 100 designs with 1, 2, 3 and 4 timepoints.

4 Conclusion

The search for Bayes optimal designs is time consuming as it often requires the evaluation of a non-trivial utility function at very many datasets simulated from the prior predictive distribution. In this paper we have focused on utility functions that depend on the generalised posterior precision, and generally this can be fairly slow to evaluate if, for example, it is estimated by determining the parameter posterior distribution via MCMC. The search is further complicated because, in general, the expected utility function is fairly flat. In contrast to other methods, we use a stepwise approach to focus in on designs in the upper tail of the distribution of expected utility, as we believe this measure to be more intuitive than criteria used in other methods.

This new algorithm is generally much more efficient in determining the optimal design in stochastic process models than others available in the literature, though the ACE algorithm performs particularly well in determining large designs when the expected utility function is unimodal. The algorithm also out performs others when the expected utility function is multi-modal and the computational budget is fairly limited. It uses a particle representation over design space to focus increasingly on regions of high utility. There are no issues of convergence in the scheme (as there are in, for example, the Müller scheme) and the output facilitates a simple comparison of near-optimal designs via their (estimated) expected utility allowing for uncertainty in the estimates.

The new algorithm is initialised using a variation of the general scheme. However, it could be initialised in a variety of ways. For example, a (space filling) maximin Latin hypercube design could be used, as is typical in the calibration of stochastic emulators (Baggaley et al., 2012). Alternatively the algorithm could be first run using a delta approximation to the expected utility function to find designs maximising $u\{\boldsymbol{d}, E(\boldsymbol{y})\}$ and then determining the initial weights using the evaluations of

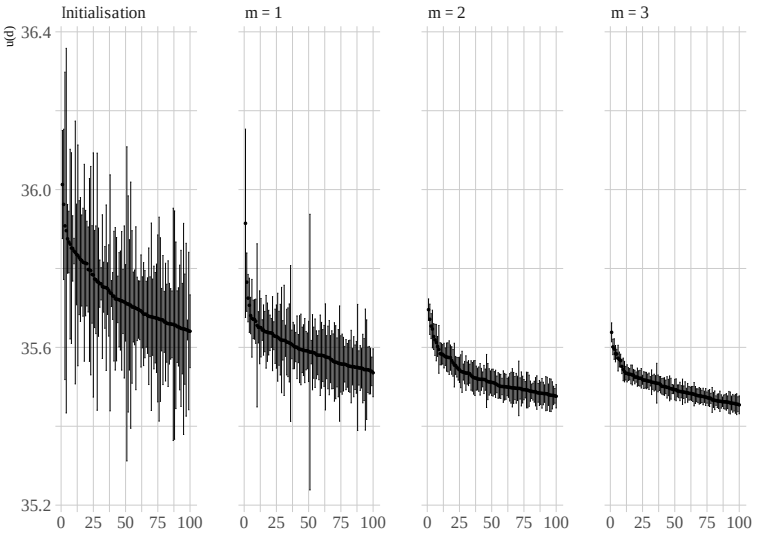

Figure 9: Estimated expected utilities (with central 95% confidence intervals) for the top 100 three timepoint designs for the aphid model after the initialisation and each step.

this approximate expected utility function. The algorithm is embarrassingly parallel and ideally suited to running on multi-core computers. The search for the optimal design is easily interrupted and restarted, allowing for the search to be monitored and additional compute resource to be added.

References

Amzal, B., F. Y. Bois, E. Parent, and C. P. Robert (2006). Bayesian-optimal design via interacting particle systems. *Journal of American Statistical Association 101*, 773–785.

Baggaley, A. W., R. J. Boys, A. Golightly, G. R. Sarson, and A. Shukurov (2012). Inference for population dynamics in the neolithic period. *The Annals of Applied Statistics 6*(4), 1352–1376.

Cook, A. R., G. J. Gibson, and C. A. Gilligan (2008). Optimal observation times in experimental epidemic processes. *Biometrics 64*(3), 860–868.

Drovandi, C. C. and A. N. Pettitt (2013). Bayesian experimental design for models with intractable likelihoods. *Biometrics 69*(4), 937–948.

Galassi, M., J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi (1996). Gnu scientific library.

Gillespie, C. S. and A. Golightly (2010). Bayesian inference for generalized stochastic population growth models with application to aphids. *Journal of Royal Statistical Society, Series* C *59*(2), 341–357.

Henderson, D. A., R. J. Boys, K. J. Krishnan, C. Lawless, and D. J. Wilkinson (2009). Bayesian emulation and calibration of a stochastic computer model of mitochondrial DNA deletions in substantia nigra neurons. *Journal of the American Statistical Association 104*(485), 76–87.

Khatab, A., E. H. Aghezzaf, C. Diallo, and I. Djelloul (2017). Selective maintenance optimisation for series-parallel systems alternating missions and scheduled breaks with stochastic durations. *International Journal of Production Research 55*(10), 3008–3024.

Matis, J. H., T. R. Kiffe, T. I. Matis, and D. E. Stevenson (2007). Stochastic modeling of aphid population growth with nonlinear, power-law dynamics. *Mathematical Biosciences 208*(2), 469–494.

Matis, T. I., M. N. Parajulee, J. H. Matis, and R. B. Shrestha (2008). A mechanistic model based analysis of cotton aphid population dynamics data. *Agricultural and Forest Entomology 10*(4), 355–362.

Müller, P. (1999). Simulation-based optimal design. In *Bayesian Statistics 6: Proceedings of Sixth Valencia International Meeting*, Volume 6, pp. 459. Oxford University Press.

Overstall, A. M., J. M. McGree, and C. C. Drovandi (2018). An approach for finding fully Bayesian optimal designs using normal-based approximations to loss functions. *Statistics and Computing 28*(2), 343–358.

Overstall, A. M. and D. C. Woods (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics 59*(4), 458–470.

Overstall, A. M., D. C. Woods, and M. Adamou (2017). *acebayes: Optimal Bayesian experimental design using the ACE algorithm.* R package version 1.4.1.

Price, D. J., N. G. Bean, J. V. Ross, and J. Tuke (2018). An induced natural selection heuristic for finding optimal Bayesian experimental designs. *Computational Statistics & Data Analysis 126*, 112 – 124.

Ryan, E. G., C. C. Drovandi, and A. N. Pettitt (2015). Fully Bayesian experimental design for pharmacokinetic studies. *Entropy 17*(3), 1063–1089.