Bayesian optimal design for ordinary differential equation models with application in biological science

Reproduction of results

Antony M. Overstall, David C. Woods & Ben M. Parker
Southampton Statistical Sciences Research Institute
University of Southampton
Southampton

A Introduction

This document describes how to reproduce all of the results in the main manuscript *Bayesian optimal design for ordinary differential equation models with application in biological science*. It was produced using `knitr` so is completely reproducible. The requirements are

- `R` (at least version 3.5.0);
- `acebayes R` package (at least version 1.6.0) available at `https://cran.r-project.org/web/packages/acebayes/index.html`;
- `aceodes R` package (at least version 1.12); available as Supplementary Material.

For information on the `acebayes` package, see Overstall et al. (2018).

We begin by loading the two `R` packages.

```
library(acebayes)
library(aceodes)
```

B Compartmental model

See Section 4.2 of the main manuscript for details on the structure of the model and design space.

There are six scenarios given by the combination of three utility functions,

- SIG - Shannon information gain;
- NSEL - negative squared error loss;
- NAEL - negative absolute error loss;

with two methods of solving the system of differential equations,

- exact;
- probabilistic.

We begin by setting the number of runs, n, to be 15.

```
n <- 15
```

The ACE algorithm is implemented in the `acebayes` package by the `R` function `ace`. To use `ace`, we need to supply a function which computes B Monte Carlo approximations to the utility function for a given design $\mathbf{d}$. Functions which do this for the compartmental model example are contained in the `aceodes` package. They have the following naming convention:

`Compartmental<utility><solver>`

where `<utility>` is one of `Sig`, `Nsel` or `Nael`, and `<solver>` is one of `Exact` or `Prob`. These functions take two arguments: `B` (number of Monte Carlo approximations) and `d` (an $n \times 1$ matrix giving the time points, scaled to $[-1, 1]$). The sample mean of the `B` Monte Carlo approximations to the utility function is a double loop Monte Carlo approximation to the expected utility. For example, the following code generates a design (a Latin-Hypercube design in one variable where each element is scaled to $[-1, 1]$) and then calculates the double loop Monte Carlo approximation to the expected SIG utility under the probabilistic solver.

```
set.seed(1)
d <- optimumLHS(n = n, k = 1) * 2 - 1
mean(CompartmentalSigProb(d = d, B = 20000))


## [1] 3.901361
```

We will pass this function to `ace`. Other specified arguments of `ace` are as follows.

- `start.d = d`. The starting design for the ACE algorithm.
- `limits = CompartmentalLimits`. In the compartmental model example, the time points are required to be 15 minutes apart. This argument to `ace` allows the user to specify constraints on the design space. The function `CompartmentalLimits` in the `aceodes` package implements these constraints.
- `N2 = 0`. This argument specifies the number of Phase 2 iterations. There are no Phase 2 iterations for this example because we do not want to consolidate clusters of time points into a repeated time point due to the constraints on the time points.
- `progress = TRUE`. This argument is set to `TRUE` to print out the progress (iteration number and current approximation to the expected utility).

All other arguments to `ace` are left as their default values. A notable example is `N1=20`, the number of Phase 1 iterations. This value of `N1` is used for all examples in the main manuscript. The results displayed in the main manuscript are based on running ACE $C = 20$ times from C different starting designs. The following code will reproduce the C runs of ACE that produced the results in the main manuscript for the SIG utility under the probabilistic solver. An iteration involves the specification of a seed, the generation of a starting design, the calculation of an approximation to the expected SIG utility for the starting design and a run of the ACE algorithm. After each run of the ACE algorithm, the terminal design is extracted as the object `aces$phase1.d`. Note that we could have also used `aces$phase2.d` since these two objects are the same as there are no Phase 2 iterations.

```
C <- 20
terminaldesigns <- list()
for (rep in 1:C) {
    set.seed(rep)
    d <- optimumLHS(n = n, k = 1) * 2 - 1
    ini <- mean(CompartmentalSigProb(d = d, B = 20000))
    aces <- ace(utility = CompartmentalSigProb,
        start.d = d, limits = CompartmentalLimits,
        N2 = 0, progress = TRUE)
    terminaldesigns[[rep]] <- aces$phase1.d
}
```

To reproduce the results in the main manuscript, the above code should be replicated for all six combinations of utility and solver. It would be straightforward to find the C different designs for each combination more efficiently in an embarrassingly parallel fashion.

Pre-computed terminal designs (as found by the above code) can be accessed using the function `CompartmentalTerminal` in the `aceodes` package. This function will return the terminal design under each value of `rep` and under each utility and solver. For example, the design from setting `rep <- 1`, under the SIG utility and probabilistic solver, is given by the following.

```
CompartmentalTerminal(rep = 1, utility = "SIG",
    solver = "probabilistic")
```

```
##  [1] -0.9815475 -0.9600214 -0.9391195 -0.9181459 -0.8970035 -0.6255546
##  [7] -0.5779383 -0.5355534 -0.3277919  0.4654740  0.5883633  0.6557577
## [13]  0.8949248  0.9159353  0.9370777
```

By default the elements of the design are scaled to $[-1, 1]$. However, if setting the optional argument `scaled = FALSE` the design is returned with elements on the original scale of $[0, 24]$ hours. For example, the terminal design from setting `rep <- 17`, under the NAEL utility and exact solver is given by the following.

```
CompartmentalTerminal(rep = 17, utility = "NAEL",
    solver = "exact", scaled = FALSE)
```

```
##  [1]  0.4227475  2.5069797  2.9221918  3.1731377  3.5293537  3.8659385
```

```
##  [7]  4.2287319  4.4883281  4.7404005  4.9913189  5.3451158  6.2438326
## [13] 20.1207381 20.3779731 23.7492068
```

We now need to determine which of the C terminal designs has the largest approximate expected utility. For each terminal design, we compute $R = 20$ approximations to the expected utility. We then calculate the mean of these R approximations. The final design is given by the terminal design with the largest of these means. For example, under the SIG utility and probabilistic solver, the following code will compute all necessary approximations.

```
R <- 20
set.seed(1)
evals <- matrix(0, nrow = C, ncol = R)
for (rep in 1:C) {
    d <- matrix(CompartmentalTerminal(rep = rep,
        utility = "SIG", solver = "probabilistic"),
        ncol = 1)
    for (j in 1:R) {
        evals[rep, j] <- mean(CompartmentalSigProb(d = d,
            B = 20000))
    }
}
```

The above code should be replicated for all six combinations of utility and solver. The pre-computed matrix `evals`, as found using the above code, can be accessed using the `CompartmentalEvals` function in the `aceodes` package. This function will return the `evals` matrix under each utility and solver. An optional argument `rep`, if specified, will lead `CompartmentalEvals` to return the `rep`-th row of `evals` corresponding to R approximations to the expected utility of the terminal design found from the `rep`th starting design. For example, the following code produces the matrix `evals` for the SIG utility and probabilistic solver (the argument `rep` is unspecified).

```
evals <- CompartmentalEvals(utility = "SIG", solver = "probabilistic")
```

We calculate the mean of each row and find the design with the largest approximation to the expected SIG utility.

```
rowmeans <- rowMeans(evals)
best <- which.max(rowmeans)
best
```

```
## [1] 13
```

Therefore the final design under a SIG utility and probabilistic solver can be found as follows.

```
CompartmentalTerminal(rep = best, utility = "SIG",
    solver = "probabilistic", scaled = FALSE)
```

```
##  [1]  0.2142733  0.4654244  0.7154794  0.9677044  1.2192196  4.1009335
##  [7]  4.6450292  5.4933413  7.0125478  7.6147953 19.7280335 22.7464777
## [13] 22.9972231 23.2476085 24.0000000
```

The pre-computed final design, as found using the above procedure, is returned by the function `CompartmentalFinal`. For example, the final design under a SIG utility and probabilistic solver can be found as follows (and can be confirmed to be identical to the above).

```
CompartmentalFinal(utility = "SIG", solver = "probabilistic",
    scaled = FALSE)
```

```
##  [1]  0.2142733  0.4654244  0.7154794  0.9677044  1.2192196  4.1009335
##  [7]  4.6450292  5.4933413  7.0125478  7.6147953 19.7280335 22.7464777
## [13] 22.9972231 23.2476085 24.0000000
```

The results of the compartmental model example are summarised by Figure 2 of the main manuscript. To reproduce, we first need to calculate double-loop Monte Carlo approximations to the expected utility (under the exact solver) for a) the final designs found by ACE under the probabilistic solver; and b) a uniformly-spaced design (named `uniform_d`). This is achieved by the following code.

```
set.seed(1)
sig_prob <- rep(0, 20)
nsel_prob <- rep(0, 20)
nael_prob <- rep(0, 20)
sig_uni <- rep(0, 20)
nsel_uni <- rep(0, 20)
nael_uni <- rep(0, 20)
sig_d <- matrix(CompartmentalFinal(utility = "SIG",
    solver = "probabilistic"), ncol = 1)
nsel_d <- matrix(CompartmentalFinal(utility = "NSEL",
    solver = "probabilistic"), ncol = 1)
nael_d <- matrix(CompartmentalFinal(utility = "NAEL",
    solver = "probabilistic"), ncol = 1)
uniform_d <- matrix(seq(from = -1, to = 1, length.out = n),
    ncol = 1)
for (i in 1:20) {
    sig_prob[i] <- mean(CompartmentalSigExact(d = sig_d,
        B = 20000))
    nsel_prob[i] <- mean(CompartmentalNselExact(d = nsel_d,
        B = 20000))
    nael_prob[i] <- mean(CompartmentalNaelExact(d = nael_d,
        B = 20000))
```

```
    sig_uni[i] <- mean(CompartmentalSigExact(d = uniform_d,
        B = 20000))
    nsel_uni[i] <- mean(CompartmentalNselExact(d = uniform_d,
        B = 20000))
    nael_uni[i] <- mean(CompartmentalNaelExact(d = uniform_d,
        B = 20000))
}
```

We do not need to do this for the final designs found by ACE under the exact solver since the approximations to the expected utility are already available, i.e. they are given by the `CompartmentalEvals` function with the argument `solver = "exact"`.

The following code will reproduce Figure 2 of the main manuscript. We first create `theta`, a `BB` by 3 matrix where each row is a vector generated from the prior distribution on the model parameters and `BB` is the number of draws from the exact solution. Then we calculate `mu`, a `BB` by `grid.size` matrix where each row is the exact solution, $u_2(t)$, calculated for a fine grid (of size `grid.size`) of time points for each vector of parameters generated from the prior. The result is shown in Figure A in this document.

```
grid.size <- 100
grid <- seq(from = 0, to = 24, length = grid.size)
BB <- 100
set.seed(1)
theta <- cbind(rlnorm(n = BB, meanlog = log(0.1),
    sdlog = sqrt(0.05)), rlnorm(n = BB, meanlog = log(1),
    sdlog = sqrt(0.05)), rlnorm(n = BB, meanlog = log(20),
    sdlog = sqrt(0.05)))
D <- 400
mu <- matrix(0, nrow = BB, ncol = grid.size)
for (b in 1:BB) {
    mu[b, ] <- (D * theta[b, 2]/(theta[b, 3] *
        (theta[b, 2] - theta[b, 1]))) * (exp(-theta[b,
        1] * grid) - exp(-theta[b, 2] * grid))
}

sig_evals <- CompartmentalEvals(utility = "SIG",
    solver = "exact")
nsel_evals <- CompartmentalEvals(utility = "NSEL",
    solver = "exact")
nael_evals <- CompartmentalEvals(utility = "NAEL",
    solver = "exact")

sig_best <- which.max(rowMeans(sig_evals))
nsel_best <- which.max(rowMeans(nsel_evals))
nael_best <- which.max(rowMeans(nael_evals))

layout(matrix(c(1, 2, 3, 4, 4, 4, 5, 5, 5), nrow = 3,
```

```
    byrow = TRUE), heights = c(1, 0.8, 0.29))
par(mai = c(0.35, 0.32, 0.2, 0.05), mgp = c(1.6,
    0.5, 0))

boxplot(sig_evals[sig_best, ], sig_prob, sig_uni,
    names = c("Exact", "Probabilistic", "Uniform"),
    ylab = "Expected SIG", main = "Shannon information gain")

boxplot(nsel_evals[nsel_best, ], nsel_prob, nsel_uni,
    names = c("Exact", "Probabilistic", "Uniform"),
    ylab = "Expected NSEL", main = "Negative squared error loss")

boxplot(nael_evals[nael_best, ], nael_prob, nael_uni,
    names = c("Exact", "Probabilistic", "Uniform"),
    ylab = "Expected NAEL", main = "Negative absolute error loss")

par(mai = c(0.4, 0.4, 0.05, 0.05), mgp = c(1.6,
    0.5, 0))

plot(grid, mu[1, ], type = "l", col = 8, xlab = "Time (h)",
    ylab = expression(paste("Amount of drug, ",
        u[2](t))), xlim = c(0, 24), ylim = c(0,
        32))
for (i in 2:BB) {
    lines(grid, mu[i, ], col = 8)
}

points((as.vector(uniform_d) + 1) * 12, rep(0,
    15), pch = 16)

points(CompartmentalFinal(utility = "SIG", solver = "probabilistic",
    scaled = FALSE), rep(28.5, 15), pch = 3)
points(CompartmentalFinal(utility = "SIG", solver = "exact",
    scaled = FALSE), rep(31.5, 15), pch = 4)

points(CompartmentalFinal(utility = "NSEL", solver = "probabilistic",
    scaled = FALSE), rep(18.5, 15), pch = 2)
points(CompartmentalFinal(utility = "NSEL", solver = "exact",
    scaled = FALSE), rep(21.5, 15), pch = 6)

points(CompartmentalFinal(utility = "NAEL", solver = "probabilistic",
    scaled = FALSE), rep(8.5, 15), pch = 5)
points(CompartmentalFinal(utility = "NAEL", solver = "exact",
    scaled = FALSE), rep(11.5, 15), pch = 1)

par(mai = c(0, 0, 0, 0), mgp = c(0, 0, 0))
```

```
plot(0, 0, xlab = "", ylab = "", xlim = c(0, 1),
    ylim = c(0, 1), type = "n", axes = FALSE)
legend(x = 0.2, y = 1, legend = c("SIG - exact",
    "SIG - approximate", "NSEL - exact", "NSEL - approximate",
    "NAEL - exact", "NAEL - approximate", "Uniform"),
    pch = c(4, 3, 6, 2, 1, 5, 16), ncol = 4, bty = "n")
```

C Fitzhugh-Nagumo model

See Section 4.3 of the main manuscript for details on the structure of the model and design space. There are three scenarios given by the three utility functions: SIG, NSEL and NAEL.

The description here follows the same format as Section B of this document. We begin by specifying the number of runs.

```
n <- 21
```

The `aceodes` package contains three functions which return B Monte Carlo approximations to the utility function for a given design $\mathbf{d}$ under each of the three utility functions. The naming convention for these functions is

`Fitzhugh<utility>`

where `<utility>` is one of `Sig`, `Nsel` or `Nael`. Each function takes two arguments: `B` (number of Monte Carlo approximations) and `d` (an $n \times 1$ matrix giving the time points, scaled to $[-1, 1]$). The following code will reproduce the C runs of ACE that produced the results in the main manuscript for the SIG utility. The arguments passed to the function `ace` are the same as in Section B with the exception of the argument `limits`. Here we use the function `FitzhughLimits` within the `aceodes` package which ensures that observation times are at least 0.25ms apart. We generate the starting design by using the function `FitzhughStart` which randomly generates a design, again ensuring that the observation times are 0.25ms apart.

```
C <- 20
terminaldesigns <- list()
for (rep in 1:C) {
    set.seed(rep)
    d <- FitzhughStart()
    ini <- mean(FitzhughSig(d = d, B = 20000))
    aces <- ace(utility = FitzhughSig, start.d = d,
        limits = FitzhughLimits, N2 = 0, progress = TRUE)
    terminaldesigns[[rep]] <- aces$phase1.d
}
```

As in Section B, pre-computed terminal designs equivalent to those found using the above code are available by using the function `FitzhughTerminal`. This function will return the terminal design

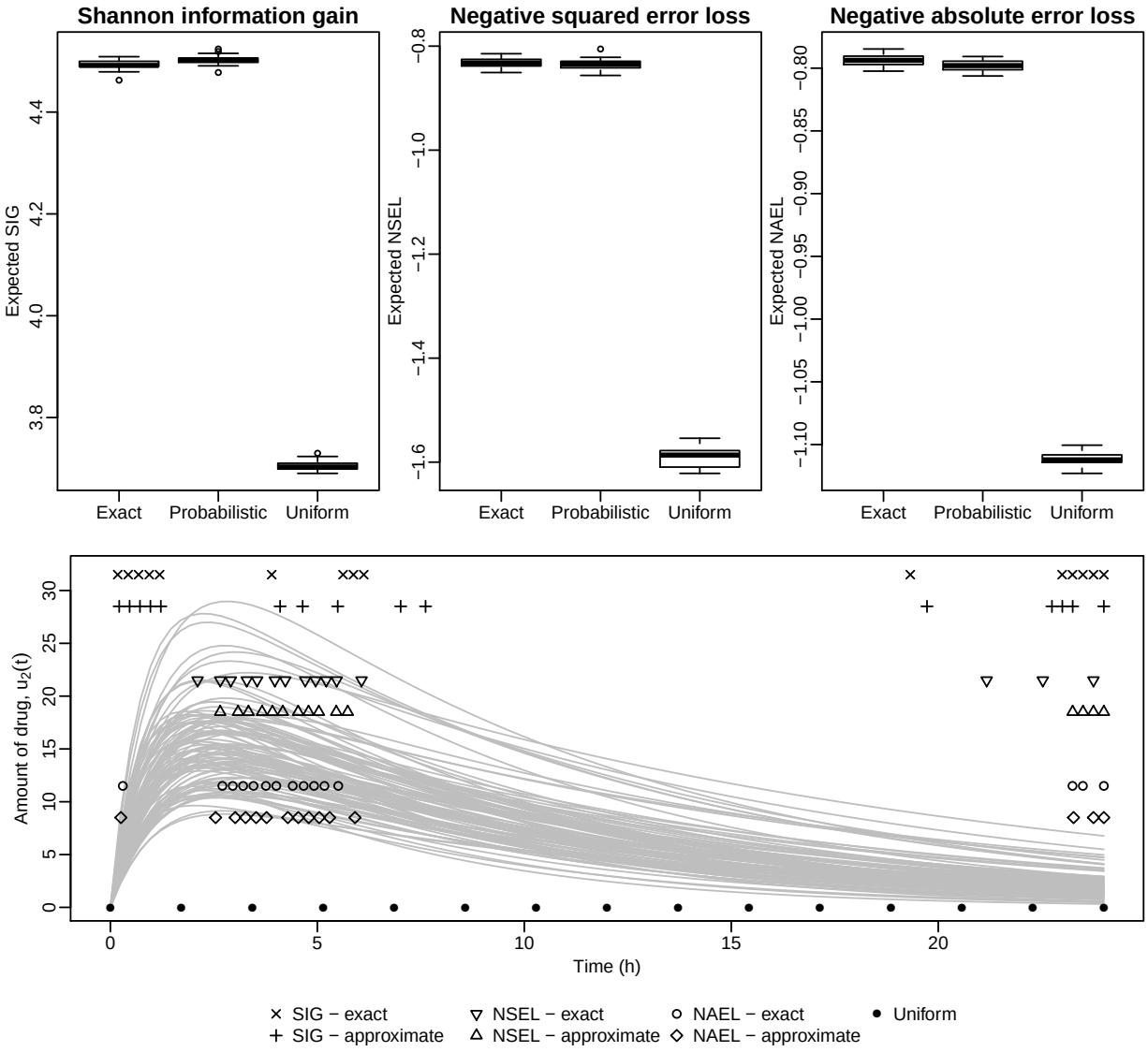

Figure A: Reproduction of Figure 2 of the main manuscript

under each value of `rep` and under each utility. For example, the terminal design from setting `rep <- 1`, under the SIG utility, is given by the following.

```
FitzhughTerminal(rep = 1, utility = "SIG")
```

```
##  [1] -0.930833600 -0.905001285 -0.879838896 -0.336504029 -0.311440232
##  [6] -0.224508307 -0.072990857 -0.037311541  0.001651859  0.131630113
## [11]  0.211824375  0.306997593  0.345619605  0.488234449  0.606085231
## [16]  0.772936830  0.852954176  0.878131481  0.923724834  0.948729851
## [21]  0.999587822
```

We now need to determine which of the C terminal designs has the largest approximate expected utility. We use the same procedure as explained in Section B and as implemented in the code below. However, for comparison purposes we have also calculated R approximations to the expected SIG utility for a uniformly-spaced design.

```
uniform_d <- matrix(seq(from = -1, to = 1, length.out = n),
    ncol = 1)

R <- 20
set.seed(1)
evals <- matrix(0, nrow = C, ncol = R)
for (rep in 1:C) {
    d <- matrix(FitzhughTerminal(rep = rep, utility = "SIG"),
        ncol = 1)
    for (j in 1:R) {
        evals[rep, j] <- mean(FitzhughSig(d = d,
            B = 20000))
    }
}

eval_default <- rep(0, R)
for (j in 1:R) {
    eval_default[j] <- mean(FitzhughSig(d = uniform_d,
        B = 20000))
}
```

Alternatively, the functions `FitzhughEvals` and `FitzhughDefaultEvals` will return pre-computed values for `evals` and `eval_default`. For example, R double-loop Monte Carlo approximations to the expected SIG utility for the terminal design with `rep <- 1` and for the uniformly-spaced design are given by the following calls.

```
FitzhughEvals(rep = 1, utility = "SIG")
```

```
##  [1] 3.524787 3.503806 3.509155 3.533653 3.533308 3.519999 3.531306
```

```
##  [8] 3.531043 3.530390 3.509835 3.499892 3.514787 3.508155 3.515520
## [15] 3.537855 3.517700 3.520251 3.520871 3.508232 3.524520
```

```
FitzhughDefaultEvals(utility = "SIG")
```

```
##  [1] 3.315241 3.307318 3.280405 3.319252 3.291936 3.299310 3.308236
##  [8] 3.292644 3.321684 3.296829 3.291588 3.312239 3.302104 3.288325
## [15] 3.312838 3.296880 3.298993 3.279409 3.296158 3.316047
```

We now determine which of the C terminal designs has the largest approximate expected utility. The following code will achieve this for the SIG utility.

```
sig_best <- which.max(rowMeans(FitzhughEvals(utility = "SIG")))
FitzhughTerminal(rep = sig_best, utility = "SIG",
    scaled = FALSE)
```

```
##  [1]  0.7655463  1.0532940  1.3034703  6.9845681  7.9104365  9.0266146
##  [7]  9.7420258 11.4653006 11.7177273 12.0466851 12.6604152 13.4597359
## [13] 14.5406849 15.3414634 15.9132283 17.0287406 17.3867651 18.8797852
## [19] 19.4976836 19.7481935 20.0000000
```

Alternatively, the function `FitzhughFinal` will return the pre-computed final design for each utility, e.g. we can immediately find the above design as follows.

```
FitzhughFinal(utility = "SIG", scaled = FALSE)
```

```
##  [1]  0.7655463  1.0532940  1.3034703  6.9845681  7.9104365  9.0266146
##  [7]  9.7420258 11.4653006 11.7177273 12.0466851 12.6604152 13.4597359
## [13] 14.5406849 15.3414634 15.9132283 17.0287406 17.3867651 18.8797852
## [19] 19.4976836 19.7481935 20.0000000
```

The following code will reproduce Figure 3 in the main manuscript. We first determine the final design under the NSEL and NAEL utilities. We then generate a sample of size $B = 100$ from the probabilistic solution for a fine grid of size `grid.size = 1000` on the interval $[0, 20]$ms. The result is shown in Figure B in this document.

```
nsel_best <- which.max(rowMeans(FitzhughEvals(utility = "NSEL")))
nael_best <- which.max(rowMeans(FitzhughEvals(utility = "NAEL")))

set.seed(1)
grid <- seq(from = 0, to = 20, length = 1000)
mu <- FitzhughSolver(grid = grid, B = 100)

layout(matrix(c(1, 2, 3, 4, 4, 4, 5, 5, 5), nrow = 3,
```

```
    byrow = TRUE), heights = c(1, 1, 0.29))
par(mai = c(0.35, 0.4, 0.2, 0.1), mgp = c(1.75,
    0.7, 0))

boxplot(FitzhughEvals(rep = sig_best, utility = "SIG"),
    FitzhughDefaultEvals(utility = "SIG"), ylab = "Expected SIG",
    names = c("Optimal", "Uniform"), main = "Shannon information gain")
boxplot(FitzhughEvals(rep = nsel_best, utility = "NSEL"),
    FitzhughDefaultEvals(utility = "NSEL"), ylab = "Expected NSEL",
    names = c("Optimal", "Uniform"), main = "Negative squared error loss")
boxplot(FitzhughEvals(rep = nael_best, utility = "NAEL"),
    FitzhughDefaultEvals(utility = "NAEL"), ylab = "Expected NAEL",
    names = c("Optimal", "Uniform"), main = "Negative absolute error loss")

par(mai = c(0.35, 0.4, 0.05, 0.1), mgp = c(1.75,
    0.7, 0))

plot(0, 0, xlim = c(0, 20), ylim = c(-2.5, 2.5),
    type = "n", xlab = "Time (ms)", ylab = expression(paste("Voltage, ",
        u[1](t))))
for (i in 1:100) {
    lines(grid, mu[i, ], col = 8)
}
points((as.vector(uniform_d) + 1) * 10, rep(-2,
    21), pch = 16)
points(FitzhughFinal(utility = "SIG", scaled = FALSE),
    rep(-0.666, 21), pch = 1)
points(FitzhughFinal(utility = "NSEL", scaled = FALSE),
    rep(0.666, 21), pch = 2)
points(FitzhughFinal(utility = "NAEL", scaled = FALSE),
    rep(2, 21), pch = 3)
plot(0, 0, xlab = "", ylab = "", xlim = c(0, 1),
    ylim = c(0, 1), type = "n", axes = FALSE)
legend(x = 0.25, y = 1, legend = c("SIG", "NSEL",
    "NAEL", "Uniform"), pch = c(1, 2, 3, 16),
    ncol = 4, bty = "n")
```

D JAKSTAT model

See Section 4.4 of the main manuscript for details on the structure of the model and design space. There are three scenarios given by the three utility functions: SIG, NSEL and NAEL.

The description here follows the same format as Sections B and C of this document. We begin by specifying the number of dimensions of the design space.

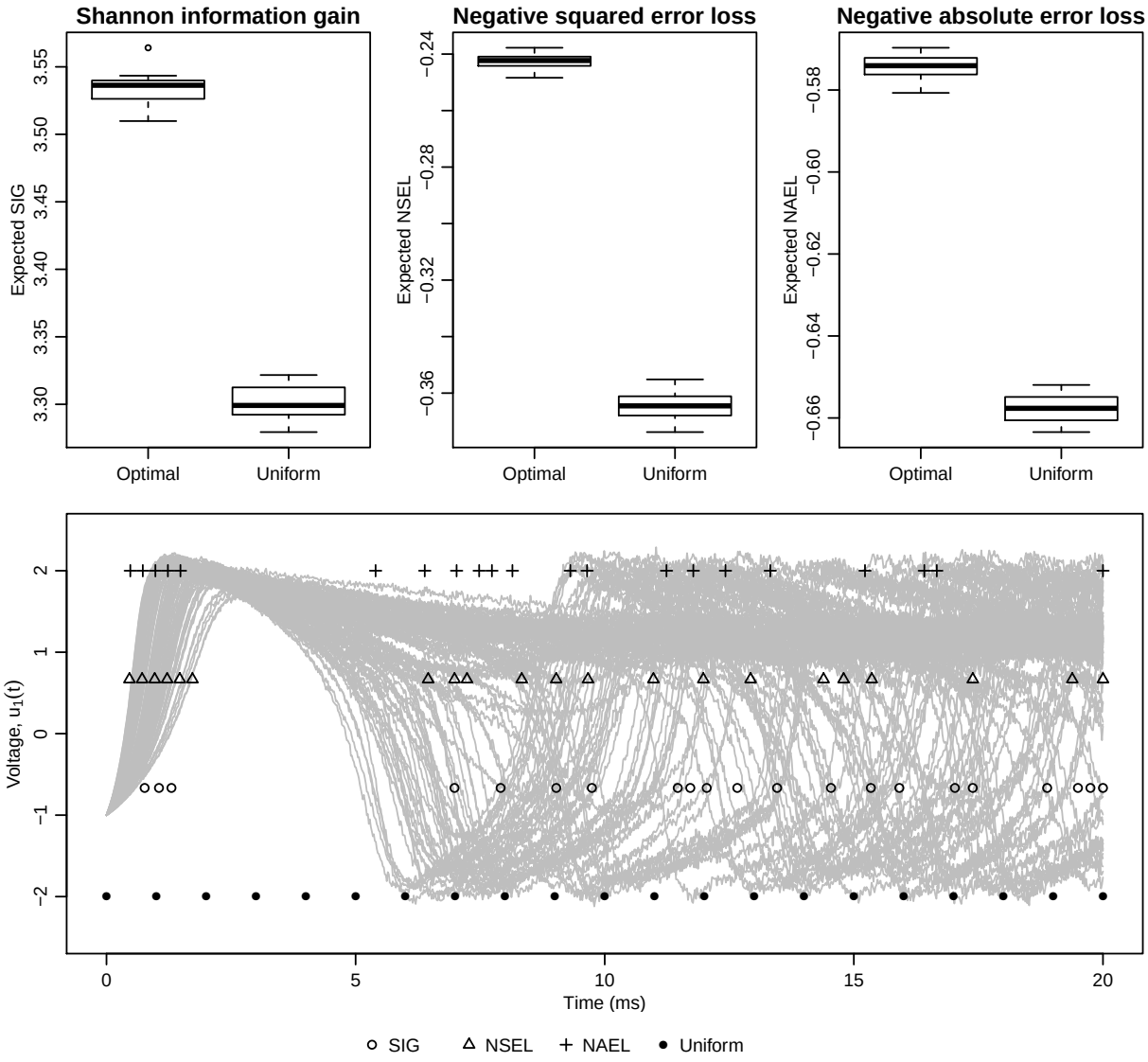

Figure B: Reproduction of Figure 3 of the main manuscript

```
n <- 17
```

The `aceodes` package contains three functions which return B Monte Carlo approximations to the utility function for a given design $\mathbf{d}$ under each of the three utility functions. The naming convention for these functions is

`Jakstat<utility>`

where `<utility>` is one of `Sig`, `Nsel` or `Nael`. Each function takes two arguments: `B` (number of Monte Carlo approximations) and `d` (an $n \times 1$ matrix giving the time points, scaled to $[-1, 1]$). The following code will reproduce the C runs of ACE that produced the results in the main manuscript for the SIG utility. The arguments passed to the function `ace` are the same as in Sections B and C with the exception of the argument `limits`. Here we use the function `JakstatLimits` within the `aceodes` package which ensures that observation times are at least one minute apart. We generate the starting design by using the function `JakstatStart` which randomly generates a design ensuring that the observation times are one minute apart.

```
C <- 20
terminaldesigns <- list()
for (rep in 1:C) {
    set.seed(rep)
    d <- JakstatStart()
    ini <- mean(JakstatSig(d = d, B = 20000))
    aces <- ace(utility = JakstatSig, start.d = d,
        limits = JakstatLimits, N2 = 0, progress = TRUE)
    terminaldesigns[[rep]] <- aces$phase1.d
}
```

Pre-computed terminal designs, as found by the above code, are available using the `JakstatTerminal` function. For example, the terminal design from setting `rep <- 1`, under the SIG utility is given by the following.

```
JakstatTerminal(rep = 1, utility = "SIG")
```

```
##  [1] -0.9575606 -0.8187177 -0.8522410 -0.1481346 -0.7183044  0.4071785
##  [7]  0.8666458  0.9332460  0.9666471 -0.9325970 -0.3478731  1.0000000
## [13] -0.9662321 -0.5261137 -0.8990559 -0.7853640 -0.6512926
```

We now need to determine which of the C terminal designs has the largest approximate expected utility. We use the same procedure as explained in Section C and as implemented in the code below. However, for comparison purposes we have also calculated R approximations to the expected SIG utility for the design used in the original experiment.

```
original_d <- rbind(-1/3, matrix(c((0:10) * 2,
    25, 30, 40, 50, 60)/30 - 1, ncol = 1))
```

```
R <- 20
set.seed(1)
evals <- matrix(0, nrow = C, ncol = R)
for (rep in 1:C) {
    d <- matrix(JakstatTerminal(rep = rep, utility = "SIG"),
        ncol = 1)
    for (j in 1:R) {
        evals[rep, j] <- mean(JakstatSig(d = d,
            B = 20000))
    }
}

eval_default <- rep(0, R)
for (j in 1:R) {
    eval_default[j] <- mean(JakstatSig(d = original_d,
        B = 20000))
}
```

Alternatively, the functions `JakstatEvals` and `JakstatDefaultEvals` will return pre-computed values for `evals` and `eval_default`. For example, R double-loop Monte Carlo approximations to the expected SIG utility for the terminal design with `rep <- 1` and for the original design are given by the following calls.

```
JakstatEvals(rep = 1, utility = "SIG")
```

```
##  [1] 6.652024 6.595477 6.519894 6.557432 6.516149 6.541579 6.465256
##  [8] 6.683091 6.561037 6.579198 6.423052 6.548960 6.597089 6.604062
## [15] 6.576135 6.520742 6.551353 6.514891 6.688837 6.637847
```

```
JakstatDefaultEvals(utility = "SIG")
```

```
##  [1] 5.313282 5.467824 5.513073 5.310051 5.496284 5.468294 5.502495
##  [8] 5.450233 5.471855 5.474786 5.490081 5.473036 5.427179 5.386047
## [15] 5.389823 5.401027 5.406474 5.616443 5.404650 5.391099
```

We now determine which of the C terminal designs has the largest approximate expected utility. The following code will achieve this for the SIG utility.

```
sig_best <- which.max(rowMeans(JakstatEvals(utility = "SIG")))
JakstatTerminal(rep = sig_best, utility = "SIG",
    scaled = FALSE)
```

```
##  [1]  1.174683 11.419421 58.996354  6.286307  8.292391 12.483140 42.949969
##  [8] 57.992491  1.009224 37.353905  4.274334  5.279898 18.956695  3.266914
## [15]  7.287799  9.295834 60.000000
```

Alternatively, the function `JakstatFinal` will return pre-computed final designs for each utility, e.g. we can immediately find the above design as follows.

```
JakstatFinal(utility = "SIG", scaled = FALSE)


##  [1]  1.174683 11.419421 58.996354  6.286307  8.292391 12.483140 42.949969
##  [8] 57.992491  1.009224 37.353905  4.274334  5.279898 18.956695  3.266914
## [15]  7.287799  9.295834 60.000000
```

The following code will reproduce Figure 4 in the main manuscript. We first determine the final design under the NSEL and NAEL utilities. We then generate a sample of size $B = 100$ from the probabilistic solution for a fine grid of size 1000 on the interval $[0, 60]$mins. The result is shown in Figure C in this document.

```
nsel_best <- which.max(rowMeans(JakstatEvals(utility = "NSEL")))
nael_best <- which.max(rowMeans(JakstatEvals(utility = "NAEL")))

set.seed(1)
grid.size <- 1000
grid <- seq(from = 0, to = 60, length.out = grid.size)
G <- JakstatSolver(grid = grid, B = 100)

layout(matrix(c(1, 2, 3, 4, 5, 6, 7, 7, 7), byrow = TRUE,
    ncol = 3), heights = c(1, 0.8, 0.29))
par(mai = c(0.35, 0.32, 0.2, 0.05), mgp = c(1.6,
    0.5, 0))

boxplot(JakstatEvals(rep = sig_best, utility = "SIG"),
    JakstatDefaultEvals(utility = "SIG"), ylab = "Expected SIG",
    names = c("Optimal", "Original"), main = "Shannon information gain")
boxplot(JakstatEvals(rep = nsel_best, utility = "NSEL"),
    JakstatDefaultEvals(utility = "NSEL"), ylab = "Expected NSEL",
    names = c("Optimal", "Original"), main = "Negative squared error loss")
boxplot(JakstatEvals(rep = nael_best, utility = "NAEL"),
    JakstatDefaultEvals(utility = "NAEL"), ylab = "Expected NAEL",
    names = c("Optimal", "Original"), main = "Negative absolute error loss")

par(mai = c(0.4, 0.4, 0.2, 0.05), mgp = c(1.6,
    0.5, 0))

plot(grid, G$G1[1, ], type = "l", col = 8, ylim = c(0,
    1.2), xlab = "Time (s)", ylab = expression(G[1]))
for (i in 2:BB) {
    lines(grid, G$G1[i, ], col = 8)
}
points(JakstatFinal(utility = "SIG", scaled = FALSE)[-1],
```

```
    rep(1.2, 16), pch = 3)
points(JakstatFinal(utility = "NSEL", scaled = FALSE)[-1],
    rep((2/3) * 1.2, 16), pch = 2)
points(JakstatFinal(utility = "NAEL", scaled = FALSE)[-1],
    rep((1/3) * 1.2, 16), pch = 1)
points(c((0:10) * 2, 25, 30, 40, 50, 60), rep(0,
    16), pch = 16)

plot(grid, G$G2[1, ], type = "l", col = 8, ylim = c(0,
    1.2), xlab = "Time (s)", ylab = expression(G[2]))
for (i in 2:BB) {
    lines(grid, G$G2[i, ], col = 8)
}
points(JakstatFinal(utility = "SIG", scaled = FALSE)[-1],
    rep(1.2, 16), pch = 3)
points(JakstatFinal(utility = "NSEL", scaled = FALSE)[-1],
    rep((2/3) * 1.2, 16), pch = 2)
points(JakstatFinal(utility = "NAEL", scaled = FALSE)[-1],
    rep((1/3) * 1.2, 16), pch = 1)
points(c((0:10) * 2, 25, 30, 40, 50, 60), rep(0,
    16), pch = 16)

plot(grid, G$G4[1, ], type = "l", col = 8, ylim = c(0,
    1.2), xlab = "Time (s)", ylab = expression(G[4]))
for (i in 2:BB) {
    lines(grid, G$G4[i, ], col = 8)
}
points(JakstatFinal(utility = "SIG", scaled = FALSE)[1],
    rep(1.2, 1), pch = 3)
points(JakstatFinal(utility = "NSEL", scaled = FALSE)[1],
    rep((2/3) * 1.2, 1), pch = 2)
points(JakstatFinal(utility = "NAEL", scaled = FALSE)[1],
    rep((1/3) * 1.2, 1), pch = 1)
points(20, rep(0, 1), pch = 16)

plot(0, 0, xlab = "", ylab = "", xlim = c(0, 1),
    ylim = c(0, 1), type = "n", axes = FALSE)
legend(x = 0.05, y = 1, legend = c("Shannon information gain",
    "Negative squared error loss", "Negative absolute error loss",
    "Original"), pch = c(3, 2, 1, 16), ncol = 4,
    bty = "n")
```

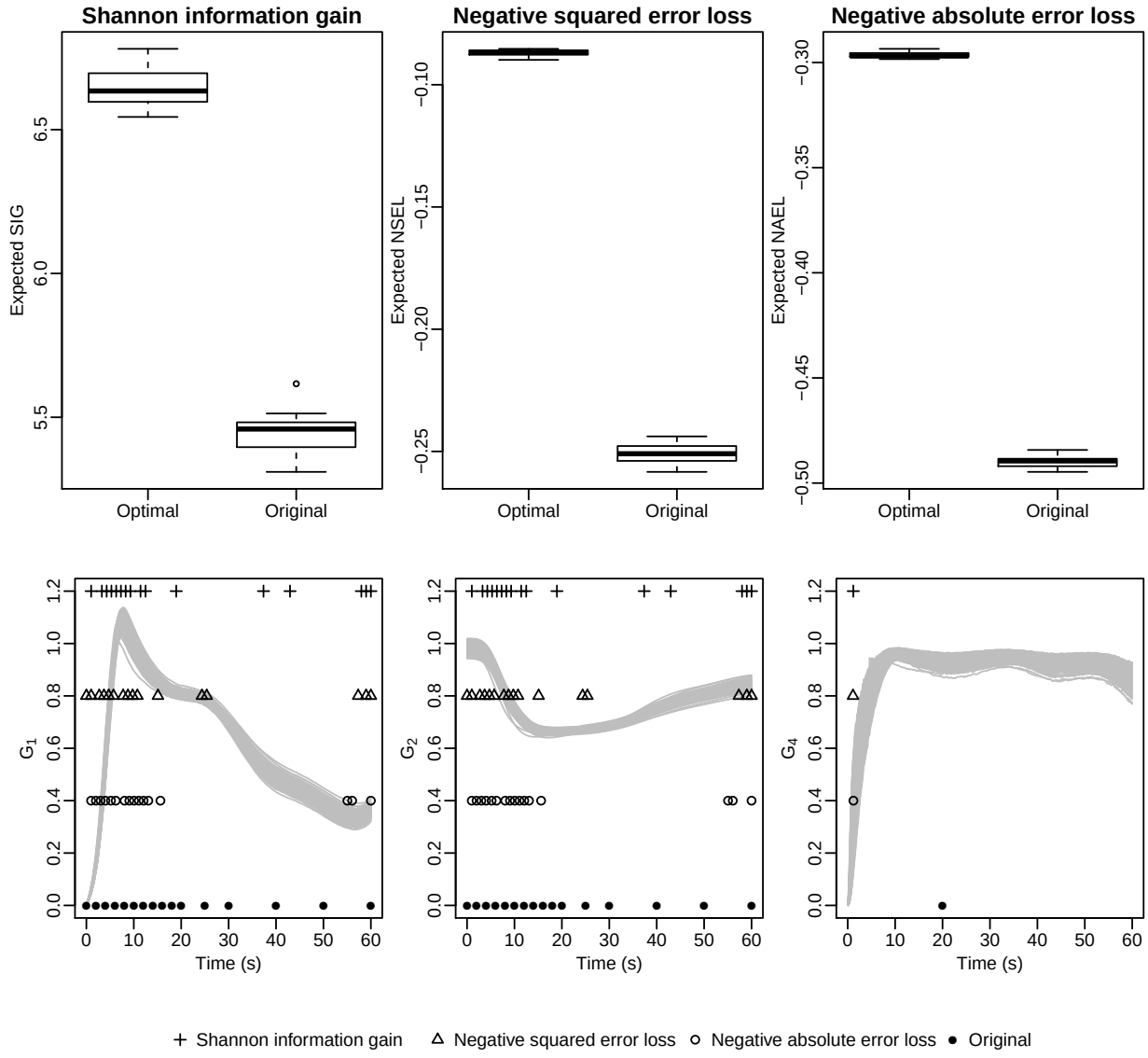

Figure C: Reproduction of Figure 4 of the main manuscript

E Placenta model

See Section 5 of the main manuscript for details on the structure of the model and design space. There are 24 scenarios given by the product of four utility functions: EST01, NSEL, NAEL and MOD01, and six different numbers of placentas: $n = 2, \ldots, 7$.

The `aceodes` package contains four functions which return B Monte Carlo approximations to the utility function for a given design $\mathbf{d}$ under each of the four utility functions. The naming convention for these functions is

`Placenta<utility>`

where `<utility>` is one of `Nsel`, `Nael`, `Est01` or `Mod01`. Each function takes, two arguments: `B` (number of Monte Carlo approximations) and `d` (a `K` by 2 matrix giving the design, scaled to $[-1, 1]$). The value of `K` is $n+n_t/2$ where $n_t = 8$ is the number of time points. The first n rows of `d` correspond to the amounts of radioactive and non-radioactive serine outside (first column) and inside (second column) the vesicle for each of the n placentas. The remaining $n_t/2 = 4$ rows correspond to the time points, where the first column gives the first four and the second column the last four. See page 22 for an example of this setup. The arguments passed to the function `ace` are the same as in Sections B to D with the exception of the argument `limits`. Here we use the function `PlacentaLimits` within the `aceodes` package which ensures that observation times are at least one minute apart. We generate the starting design by using the function `PlacentaStart` which randomly generates a design ensuring that the observation times are one minute apart.

Note that all of the code was run on the IRIDIS4 computational cluster at the University of Southampton. There is a 60 hour limit on all jobs. For all utilities with $n = 7$ placentas and for the MOD utility with $n = 6$ placentas, one run of ACE exceeded 60 hours. For simplicity, for all cases, we ran the ACE algorithm twice for ten iterations where the starting design for the second run used the terminal design from the first run. The code to reproduce the results is given below (in this case for the NSEL utility and $n = 7$ placentas).

```
n <- 7
C <- 20
terminaldesigns <- list()
for (rep in 1:C) {
    set.seed(rep)
    d <- PlacentaStart(M = n)
    set.seed(rep)
    ini <- mean(PlacentaNsel(d = d, B = 20000))
    aces1 <- ace(utility = PlacentaNsel, start.d = d,
        limits = PlacentaLimits, N1 = 10, N2 = 0,
        progress = TRUE)
    set.seed(rep)
    ini <- mean(PlacentaNsel(d = aces1$phase1.d,
        B = 20000))
    aces2 <- ace(utility = PlacentaNsel, start.d = aces2$phase1.d,
        limits = PlacentaLimits, N1 = 10, N2 = 0,
        progress = TRUE)
    terminaldesigns[[rep]] <- aces2$phase1.d
```

```
}
```

Pre-computed terminal designs, as found by the above code, are available using the `PlacentaTerminal` function in the `aceodes` package. For example, the terminal design from setting `rep <- 1`, under the NSEL utility and $n = 7$ placentas, is given by the following.

```
n <- 7
PlacentaTerminal(M = n, rep = 1, utility = "NSEL")
```

```
##              [,1]        [,2]
##  [1,] -1.0000000 -0.75367300
##  [2,] -0.6311862  1.00000000
##  [3,] -0.7070868  1.00000000
##  [4,] -1.0000000 -1.00000000
##  [5,] -1.0000000 -1.00000000
##  [6,] -1.0000000 -0.81180814
##  [7,] -0.6185544  1.00000000
##  [8,]  0.3632808 -0.09374992
##  [9,] -0.9828036 -0.94885678
## [10,] -0.6268980 -0.96584955
## [11,]  0.1989683 -0.88392369
```

We now need to determine which of the C terminal designs has the largest approximate expected utility. We use the same procedure as the previous sections. For example, the following code achieves this under the NSEL utility and for $n = 7$ placentas. Note for comparison purposes we have also calculated R approximations to the expected NSEL utility for the design used in the original experiment. Note that this last calculation is only performed for $n = 7$ placentas.

```
n <- 7

original_d <- (cbind(c(0, rep(250, 3), rep(1000,
    3)), c(0, rep(c(0, 250, 1000), 2)))/1000) *
    2 - 1
original_d <- rbind(original_d, matrix(2 * (c(5,
    10, 15, 20, 60, 120, 300, 600) - 5)/595 -
    1, ncol = 2))

R <- 20
set.seed(1)
evals <- matrix(0, nrow = C, ncol = R)
for (rep in 1:C) {
    d <- PlacentaTerminal(M = n, rep = rep, utility = "NSEL")
    for (j in 1:R) {
        evals[rep, j] <- mean(PlacentaNsel(d = d,
            B = 20000))
```

```
    }
}

set.seed(1)
eval_default <- rep(0, R)
for (j in 1:R) {
    eval_default[j] <- mean(PlacentaNsel(d = original_d,
        B = 20000))
}
```

Alternatively, the functions `PlacentaEvals` and `PlacentaDefaultEvals` will return pre-computed values for `evals` and `eval_default`. For example, R double-loop Monte Carlo approximations to the expected NSEL utility for $n = 7$ placentas for the terminal design with `rep <- 1` and for the a sub-sample of the original design are given by the following calls.

```
n <- 7

PlacentaEvals(M = n, rep = 1, utility = "NSEL")

##  [1] -27.40257 -27.89657 -27.36923 -27.40561 -27.33441 -27.50789 -27.79896
##  [8] -27.74763 -27.57775 -27.83810 -27.36022 -27.12493 -27.29571 -27.55527
## [15] -27.26772 -27.42656 -27.40861 -27.76683 -27.17378 -27.15977

PlacentaDefaultEvals(M = n, utility = "NSEL")

##  [1] -76.98590 -76.92217 -77.10167 -76.95250 -76.07635 -76.22886 -76.93698
##  [8] -77.72268 -77.03918 -77.54354 -76.93515 -76.46006 -77.32111 -76.66119
## [15] -76.59611 -76.32981 -77.09514 -77.18387 -76.93680 -76.21575
```

We now determine which of the C terminal designs has the largest approximate expected utility. The following code will achieve this for the NSEL utility for $n = 7$ placentas.

```
n <- 7

nsel_best <- which.max(rowMeans(PlacentaEvals(M = n,
    utility = "NSEL")))
PlacentaTerminal(M = n, rep = nsel_best, utility = "NSEL",
    scaled = FALSE)

## $concentrations
##          [,1]       [,2]
## [1,]   0.0000    0.00000
## [2,]   0.0000   38.27606
## [3,]   0.0000   50.26754
```

```
## [4,]   0.0000   67.85121
## [5,] 181.5726 1000.00000
## [6,] 184.9693 1000.00000
## [7,] 205.9363 1000.00000
##
## $times
## [1]   0.000000   5.071386  10.130448  15.180936  25.344212 108.624651
## [7] 257.122887 497.407378
```

Alternatively, the function `PlacentaFinal` will return the final design for each utility and value of n, e.g. we can immediately find the above design as follows.

```
PlacentaFinal(M = n, utility = "NSEL", scaled = FALSE)
```

```
## $concentrations
##          [,1]       [,2]
## [1,]   0.0000    0.00000
## [2,]   0.0000   38.27606
## [3,]   0.0000   50.26754
## [4,]   0.0000   67.85121
## [5,] 181.5726 1000.00000
## [6,] 184.9693 1000.00000
## [7,] 205.9363 1000.00000
##
## $times
## [1]   0.000000   5.071386  10.130448  15.180936  25.344212 108.624651
## [7] 257.122887 497.407378
```

The scaled version of this design is given below.

```
PlacentaFinal(M = n, utility = "NSEL", scaled = TRUE)
```

```
##             [,1]       [,2]
##  [1,] -1.0000000 -1.0000000
##  [2,] -1.0000000 -0.8642976
##  [3,] -1.0000000 -0.9234479
##  [4,] -0.6368548  1.0000000
##  [5,] -0.5881275  1.0000000
##  [6,] -1.0000000 -0.8994649
##  [7,] -0.6300613  1.0000000
##  [8,] -0.9662318 -1.0000000
##  [9,] -0.6379178 -0.1429237
## [10,]  0.6580246 -0.9830954
## [11,] -0.9493969 -0.9155193
```

Figure 5 in the main manuscript can be reproduced as follows. We first determine the final design under each utility and number of placentas. The result is shown in Figure D in this document.

```
nsel_plot <- matrix(0, nrow = 20, ncol = 6)
nsel_best <- rep(0, 6)
for (n in 2:7) {
    nsel_best[n - 1] <- which.max(rowMeans(PlacentaEvals(M = n,
        utility = "NSEL")))
    nsel_plot[, n - 1] <- PlacentaEvals(M = n,
        rep = nsel_best[n - 1], utility = "NSEL")
}
nsel_def <- PlacentaDefaultEvals(M = 7, utility = "NSEL")

nael_plot <- matrix(0, nrow = 20, ncol = 6)
nael_best <- rep(0, 6)
for (n in 2:7) {
    nael_best[n - 1] <- which.max(rowMeans(PlacentaEvals(M = n,
        utility = "NAEL")))
    nael_plot[, n - 1] <- PlacentaEvals(M = n,
        rep = nael_best[n - 1], utility = "NAEL")
}
nael_def <- PlacentaDefaultEvals(M = 7, utility = "NAEL")

est01_plot <- matrix(0, nrow = 20, ncol = 6)
est01_best <- rep(0, 6)
for (n in 2:7) {
    est01_best[n - 1] <- which.max(rowMeans(PlacentaEvals(M = n,
        utility = "EST01")))
    est01_plot[, n - 1] <- PlacentaEvals(M = n,
        rep = est01_best[n - 1], utility = "EST01")
}
est01_def <- PlacentaDefaultEvals(M = 7, utility = "EST01")

mod01_plot <- matrix(0, nrow = 20, ncol = 6)
mod01_best <- rep(0, 6)
for (n in 2:7) {
    mod01_best[n - 1] <- which.max(rowMeans(PlacentaEvals(M = n,
        utility = "MOD01")))
    mod01_plot[, n - 1] <- PlacentaEvals(M = n,
        rep = mod01_best[n - 1], utility = "MOD01")
}
mod01_def <- PlacentaDefaultEvals(M = 7, utility = "MOD01")

par(mai = c(0.5, 0.5, 0.5, 0.1), mgp = c(1.7,
    0.5, 0))
layout(matrix(c(1:4, 5, 5), ncol = 2, byrow = TRUE),
    heights = c(1, 1, 0.25))
```

```r
boxplot(nsel_plot, ylim = c(-100, 0), border = 1,
    names = as.character(2:7), ylab = "Expected negative squared error loss",
    xlab = "Number of placentas, n", main = "Negative squared error loss")
boxplot(nsel_def, add = TRUE, border = 8, names = as.character(7),
    at = 6)
boxplot(nael_plot, ylim = c(-15, 0), border = 1,
    names = as.character(2:7), ylab = "Expected negative absolute error loss",
    xlab = "Number of placentas, n", main = "Negative absolute error loss")
boxplot(nael_def, add = TRUE, border = 8, names = as.character(7),
    at = 6)
boxplot(est01_plot, ylim = c(0, 1), border = 1,
    names = as.character(2:7), ylab = "Expected Estimation 0-1",
    xlab = "Number of placentas, n", main = "Estimation 0-1")
boxplot(est01_def, add = TRUE, border = 8, names = as.character(7),
    at = 6)
boxplot(mod01_plot, ylim = c(0, 1), border = 1,
    names = as.character(2:7), ylab = "Expected model selection 0-1",
    xlab = "Number of placentas, n", main = "Model selection 0-1")
boxplot(mod01_def, add = TRUE, border = 8, names = as.character(7),
    at = 6)
par(mai = c(0, 0, 0, 0))
plot(0, 0, xlab = "", ylab = "", type = "n", axes = FALSE,
    xlim = c(0, 1), ylim = c(0, 1))
legend(x = 0.415, y = 1, ncol = 2, legend = c("Optimum",
    "Original"), col = c(1, 8), bty = "n", pch = c(16,
    16))
```

The numbers Table 1 in the main manuscript can be reproduced as follows.

```r
# NSEL utility
PlacentaFinal(M = 7, utility = "NSEL", scaled = FALSE)$concentrations
```

```
##          [,1]       [,2]
## [1,]   0.0000    0.00000
## [2,]   0.0000   38.27606
## [3,]   0.0000   50.26754
## [4,]   0.0000   67.85121
## [5,] 181.5726 1000.00000
## [6,] 184.9693 1000.00000
## [7,] 205.9363 1000.00000
```

```r
# NAEL utility
PlacentaFinal(M = 7, utility = "NAEL", scaled = FALSE)$concentrations
```

```
##          [,1]       [,2]
```

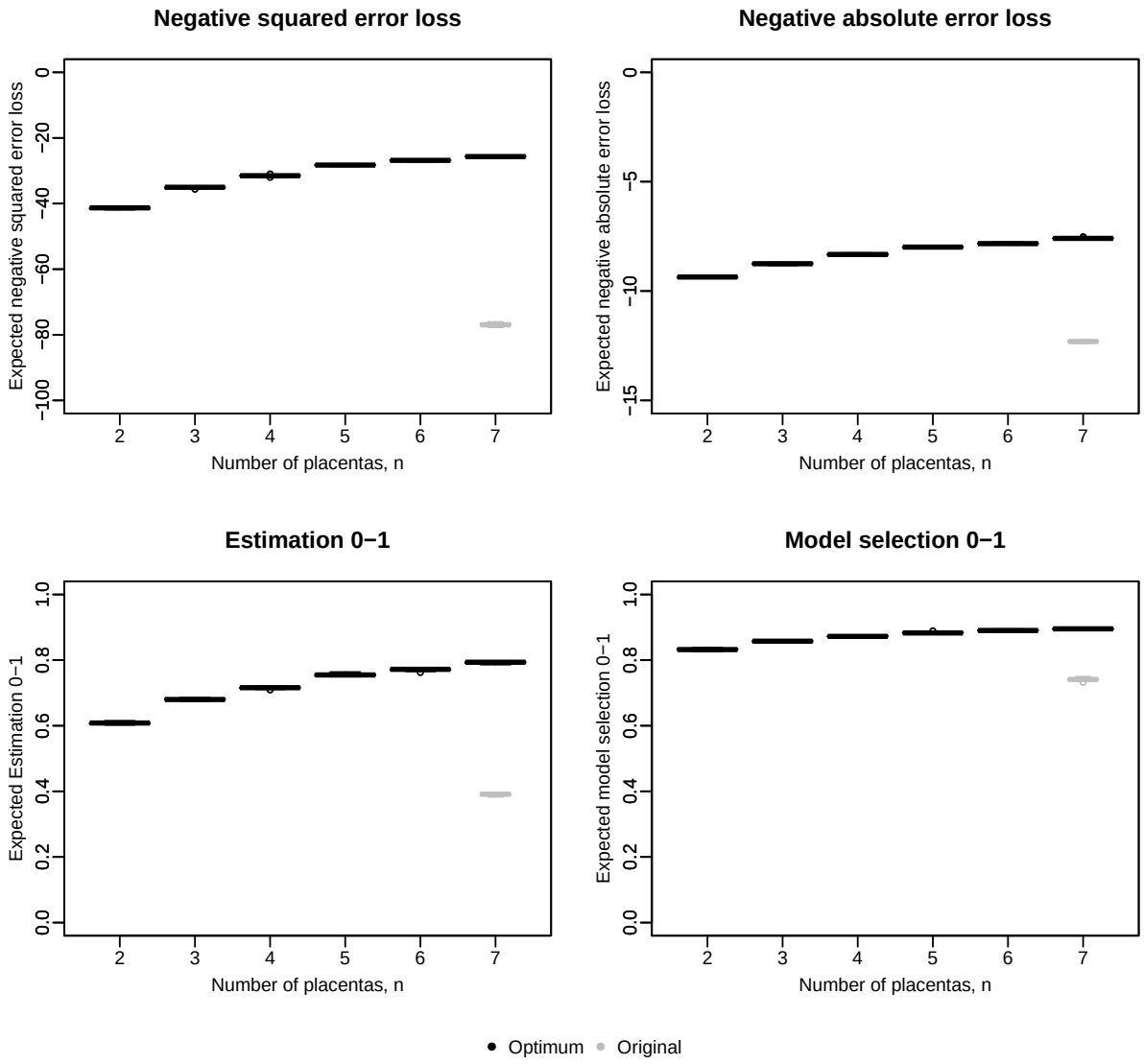

Figure D: Reproduction of Figure 5 of the main manuscript

```
## [1,]   0.0000    0.00000
## [2,]   0.0000    0.00000
## [3,]   0.0000   49.01689
## [4,]   0.0000   67.09268
## [5,] 160.3101 1000.00000
## [6,] 175.1976 1000.00000
## [7,] 211.1539 1000.00000
```

```
# EST01 utility
PlacentaFinal(M = 7, utility = "EST01", scaled = FALSE)$concentrations
```

```
##          [,1]       [,2]
## [1,]   0.0000    0.00000
## [2,]   0.0000    0.00000
## [3,]   0.0000   56.38176
## [4,]   0.0000   57.69525
## [5,] 176.5278 1000.00000
## [6,] 195.5801 1000.00000
## [7,] 209.9433 1000.00000
```

```
# MOD 01 utility
PlacentaFinal(M = 7, utility = "MOD01", scaled = FALSE)$concentrations
```

```
##         [,1]     [,2]
## [1,]   0.000  0.00000
## [2,]   0.000  0.00000
## [3,]   0.000  0.00000
## [4,]   0.000  0.00000
## [5,]   0.000 38.01907
## [6,]   0.000 41.33733
## [7,] 114.738 61.58456
```

```
# Original design
cbind(c(0, rep(250, 3), rep(1000, 3)), c(0, rep(c(0,
    250, 1000), 2)))
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]  250    0
## [3,]  250  250
## [4,]  250 1000
## [5,] 1000    0
## [6,] 1000  250
## [7,] 1000 1000
```

Figure 6 in the main manuscript can be reproduced as follows. We generate a sample of size $B = 100$ from the probabilistic solution for a fine grid of size 101 on the interval $[0, 600]$mins for each treatment of non-radioactive serine.

```
grid.size <- 101
grid <- seq(from = 0, to = 600, length.out = grid.size)
B <- 100

mu_nsel <- array(0, dim = c(7, B, grid.size))
for (m in 1:7) {
    mu_nsel[m, , ] <- PlacentaSolver(grid = grid,
        x2u2 = PlacentaFinal(M = 7, utility = "NSEL",
            scaled = FALSE)$concentrations[m,
            ], B = B)$mu
}

mu_nael <- array(0, dim = c(7, B, grid.size))
for (m in 1:7) {
    mu_nael[m, , ] <- PlacentaSolver(grid = grid,
        x2u2 = PlacentaFinal(M = 7, utility = "NAEL",
            scaled = FALSE)$concentrations[m,
            ], B = B)$mu
}

mu_est01 <- array(0, dim = c(7, B, grid.size))
for (m in 1:7) {
    mu_est01[m, , ] <- PlacentaSolver(grid = grid,
        x2u2 = PlacentaFinal(M = 7, utility = "EST01",
            scaled = FALSE)$concentrations[m,
            ], B = B)$mu
}

mu_mod01 <- array(0, dim = c(7, B, grid.size))
mod_mod01 <- matrix(0, nrow = 7, ncol = B)
for (m in 1:7) {
    modplac <- PlacentaSolver(grid = grid, x2u2 = PlacentaFinal(M = 7,
        utility = "MOD01", scaled = FALSE)$concentrations[m,
        ], B = B, model.uncertainty = TRUE)
    mu_mod01[m, , ] <- modplac$mu
    mod_mod01[m, ] <- modplac$mods
}

cx <- 0.8
par(mai = c(0.2, 0.3, 0.05, 0.1), mgp = c(0.5,
    0.05, 0), tcl = -0.2)
layout(matrix(c(1:32, rep(33, 4)), nrow = 9, ncol = 4,
    byrow = TRUE), heights = c(0.5, rep(1, 7),
```

```
    0.5), widths = c(0.5, rep(1, 3)))

par(mai = c(0, 0, 0, 0), mgp = c(0.5, 0.05, 0),
    tcl = -0.2)
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
text(0.5, 0.5, labels = "NSEL")
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
text(0.5, 0.5, labels = "NAEL")
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
text(0.5, 0.5, labels = "EST01")

for (m in 1:7) {
    par(mai = c(0, 0, 0, 0), mgp = c(0.5, 0.05,
        0), tcl = -0.2)
    plot(0, 0, type = "n", xlab = "", ylab = "",
        xlim = c(0, 1), ylim = c(0, 1), axes = FALSE)
    text(x = 0.5, y = 0.7, labels = paste("Placenta ",
        m))

    par(mai = c(0.2, 0.3, 0.05, 0.1), mgp = c(0.5,
        0.05, 0), tcl = -0.2)

    plot(grid, mu_nsel[m, 1, ], type = "l", col = 8,
        ylab = expression(u[1](t)), cex.lab = cx,
        cex.axis = cx, xlab = "t", axes = FALSE,
        ylim = c(0, 60))
    axis(side = 1, at = c(0, 600), cex.axis = cx)
    axis(side = 2, at = c(0, 600), cex.axis = cx)
    for (i in 2:B) {
        lines(grid, mu_nsel[m, i, ], col = 8)
    }
    points(PlacentaFinal(M = 7, utility = "NSEL",
        scaled = FALSE)$times, rep(10, 8), pch = 3)
    points(c(5, 10, 15, 20, 60, 120, 300, 600),
        rep(50, 8), pch = 16)

    plot(grid, mu_nael[m, 1, ], type = "l", col = 8,
        ylab = expression(u[1](t)), cex.lab = cx,
        cex.axis = cx, xlab = "t", axes = FALSE,
        ylim = c(0, 60))
    axis(side = 1, at = c(0, 600), cex.axis = cx)
```

```
    axis(side = 2, at = c(0, 600), cex.axis = cx)
    for (i in 2:B) {
        lines(grid, mu_nael[m, i, ], col = 8)
    }
    points(PlacentaFinal(M = 7, utility = "NAEL",
        scaled = FALSE)$times, rep(10, 8), pch = 3)
    points(c(5, 10, 15, 20, 60, 120, 300, 600),
        rep(50, 8), pch = 16)

    plot(grid, mu_est01[m, 1, ], type = "l", col = 8,
        ylab = expression(u[1](t)), cex.lab = cx,
        cex.axis = cx, xlab = "t", axes = FALSE,
        ylim = c(0, 60))
    axis(side = 1, at = c(0, 600), cex.axis = cx)
    axis(side = 2, at = c(0, 600), cex.axis = cx)
    for (i in 2:B) {
        lines(grid, mu_est01[m, i, ], col = 8)
    }
    points(PlacentaFinal(M = 7, utility = "EST01",
        scaled = FALSE)$times, rep(10, 8), pch = 3)
    points(c(5, 10, 15, 20, 60, 120, 300, 600),
        rep(50, 8), pch = 16)


}

par(mai = c(0, 0, 0, 0), mgp = c(0, 0, 0))

plot(0, 0, xlim = c(0, 1), ylim = c(0, 1), xlab = "",
    ylab = "", axes = FALSE, type = "n")

legend(x = 0.45, y = 1, legend = c("Optimal",
    "Original"), pch = c(3, 16), ncol = 2, bty = "n")
```

Figure 7 in the main manuscript can be reproduced as follows.

```
cx <- 0.8
par(mai = c(0.2, 0.3, 0.05, 0.1), mgp = c(0.5,
    0.05, 0), tcl = -0.2)
layout(matrix(c(1:24, rep(25, 3)), nrow = 9, ncol = 3,
    byrow = TRUE), heights = c(0.5, rep(1, 7),
    0.5), widths = c(0.5, rep(1, 2)))

par(mai = c(0, 0, 0, 0), mgp = c(0.5, 0.05, 0),
    tcl = -0.2)
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
```

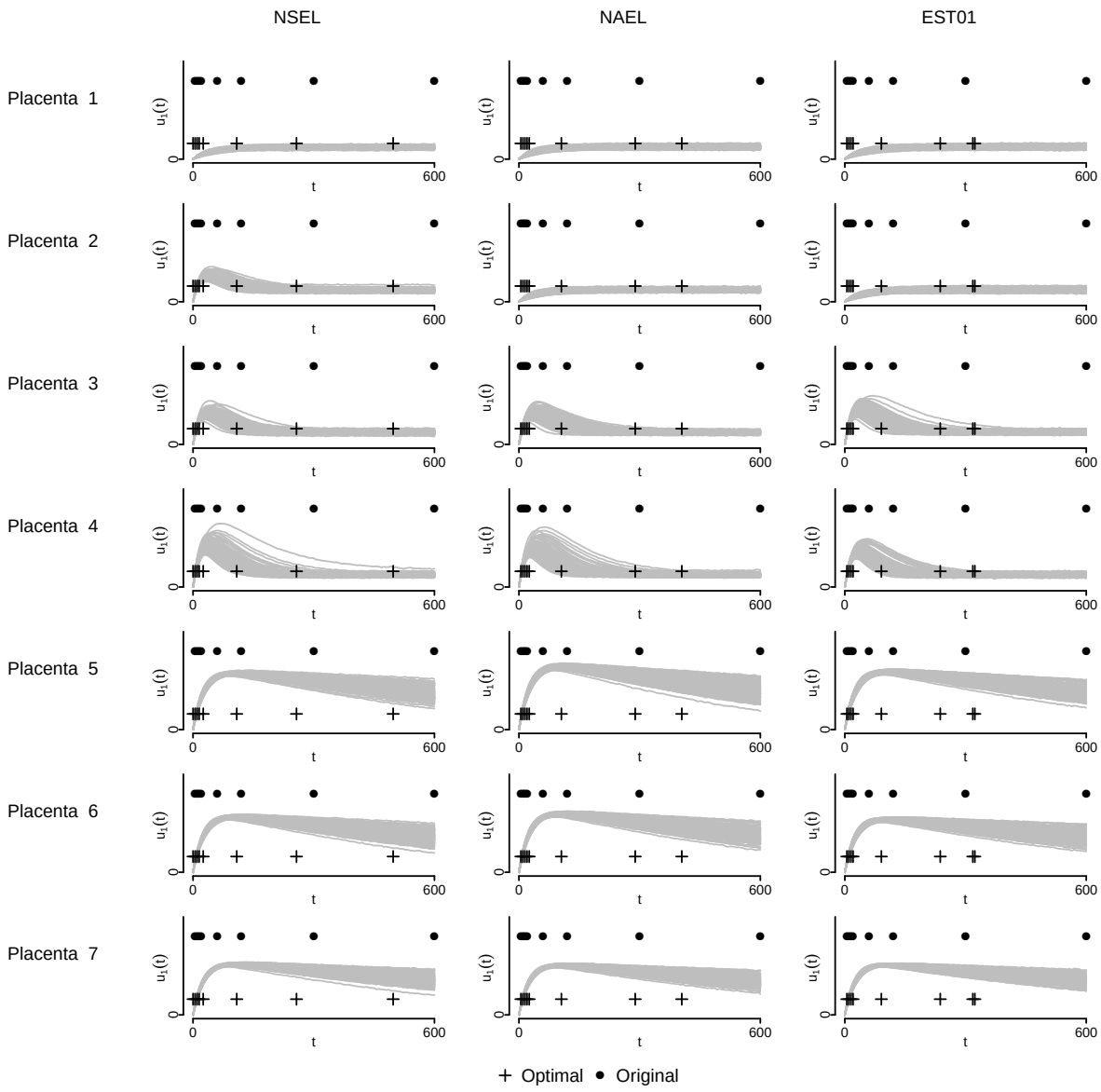

Figure E: Reproduction of Figure 6 of the main manuscript

```
    1), ylim = c(0, 1), axes = FALSE)
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
text(0.5, 0.5, labels = expression(paste("Model ",
    m[1])))
plot(0, 0, type = "n", xlab = "", ylab = "", xlim = c(0,
    1), ylim = c(0, 1), axes = FALSE)
text(0.5, 0.5, labels = expression(paste("Model ",
    m[2])))

for (m in 1:7) {
    par(mai = c(0, 0, 0, 0), mgp = c(0.5, 0.05,
        0), tcl = -0.2)
    plot(0, 0, type = "n", xlab = "", ylab = "",
        xlim = c(0, 1), ylim = c(0, 1), axes = FALSE)
    text(x = 0.5, y = 0.7, labels = paste("Placenta ",
        m))

    par(mai = c(0.2, 0.3, 0.05, 0.1), mgp = c(0.5,
        0.05, 0), tcl = -0.2)

    plot(grid, mu_mod01[m, 1, ], type = "n", col = 8,
        ylab = expression(u[1](t)), cex.lab = cx,
        cex.axis = cx, xlab = "t", axes = FALSE,
        ylim = c(0, 60))
    axis(side = 1, at = c(0, 600), cex.axis = cx)
    axis(side = 2, at = c(0, 600), cex.axis = cx)
    for (i in 1:B) {
        if (mod_mod01[m, i] == 1) {
            lines(grid, mu_mod01[m, i, ], col = 8)
        }
    }
    points(PlacentaFinal(M = 7, utility = "MOD01",
        scaled = FALSE)$times, rep(5, 8), pch = 3)
    points(c(5, 10, 15, 20, 60, 120, 300, 600),
        rep(25, 8), pch = 16)

    plot(grid, mu_mod01[m, 1, ], type = "n", col = 8,
        ylab = expression(u[1](t)), cex.lab = cx,
        cex.axis = cx, xlab = "t", axes = FALSE,
        ylim = c(0, 60))
    axis(side = 1, at = c(0, 600), cex.axis = cx)
    axis(side = 2, at = c(0, 600), cex.axis = cx)
    for (i in 1:B) {
        if (mod_mod01[m, i] == 2) {
            lines(grid, mu_mod01[m, i, ], col = 8)
```

```
        }
    }
    points(PlacentaFinal(M = 7, utility = "MOD01",
        scaled = FALSE)$times, rep(5, 8), pch = 3)
    points(c(5, 10, 15, 20, 60, 120, 300, 600),
        rep(25, 8), pch = 16)

}

par(mai = c(0, 0, 0, 0), mgp = c(0, 0, 0))

plot(0, 0, xlim = c(0, 1), ylim = c(0, 1), xlab = "",
    ylab = "", axes = FALSE, type = "n")

legend(x = 0.5, y = 1, legend = c("Optimal", "Original"),
    pch = c(3, 16), ncol = 2, bty = "n")
```

References

Overstall, A., Woods, D., and Adamou, M. (2018), "acebayes: An R Package for Bayesian Optimal Design of Experiments via Approximate Coordinate Exchange," arXiv:1705.08096.

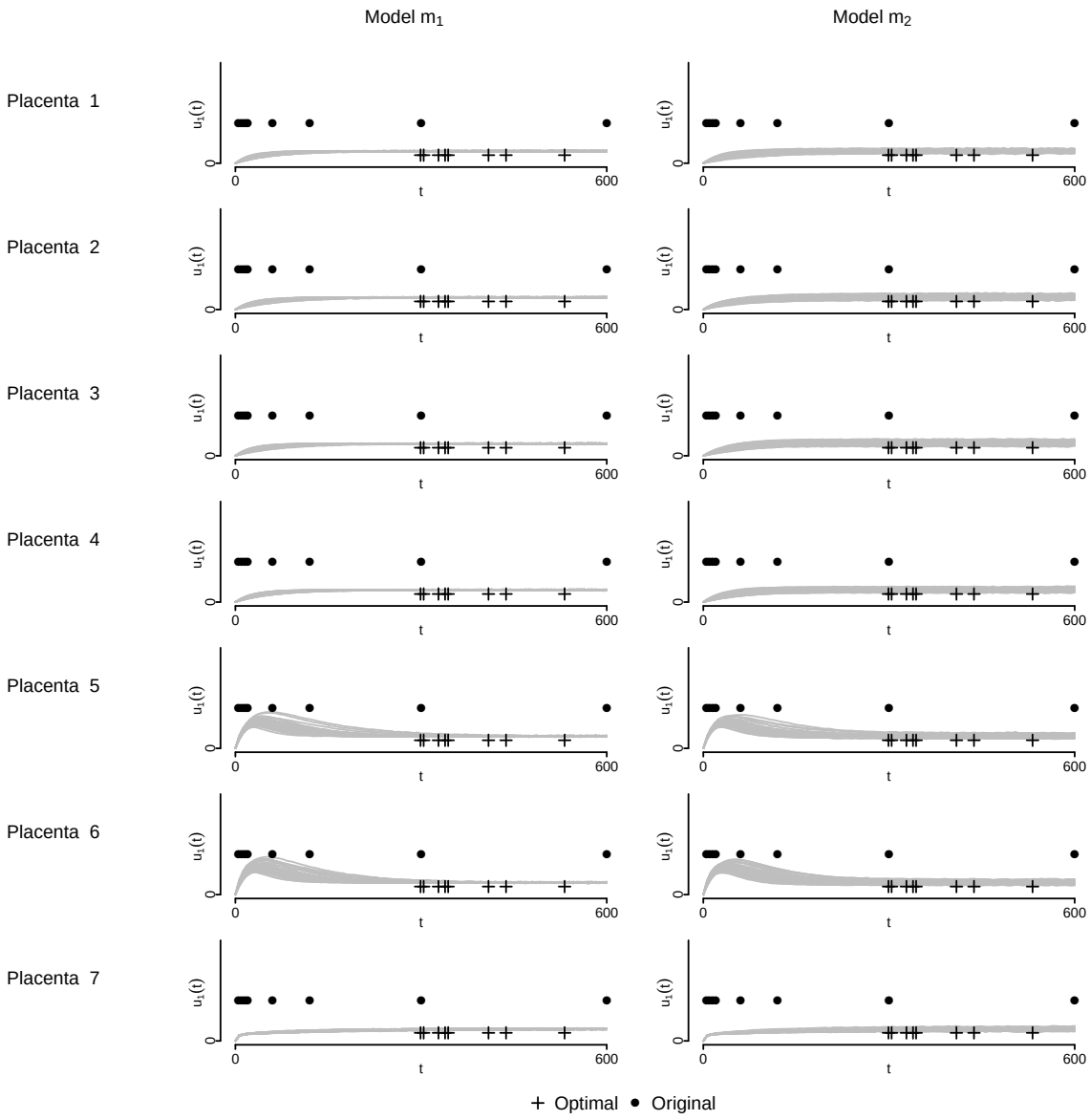

Figure F: Reproduction of Figure 7 of the main manuscript