swCRTdesign: An R Package for Stepped Wedge Trial Design and Analysis

Emily C. Voldal
University of Washington

Navneet R. Hakhu
University of California, Irvine

Fan Xia
University of Washington

Patrick J. Heagerty
University of Washington

James P. Hughes
University of Washington

Abstract

—!!!—an abstract is required—!!!—

Keywords: stepped wedge trial, cluster randomized trial, **shiny**, R.

Supplemental materials: swCRTdesign syntax and use

The full documentation for the most recent version of **swCRTdesign** can be found at https://cran.r-project.org/web/packages/swCRTdesign/swCRTdesign.pdf. This appendix provides a more thorough explanation and description of how to use these functions.

1. Creating a design

The functions `swPwr` and `swSim` both depend on defining a SWT design through `swDsn`:

```
swDsn(clusters, tx.effect.frac = 1, extra.time = 0, all.ctl.time0 = TRUE)
```

This function defines how many time points and clusters the design has, and the treatment effect for each cluster and time. The `clusters` argument is a vector that defines the number of sequences (via the length of the vector) and clusters per sequence (via the components of the vector). If observation continues after all sequences have switched to treatment, this can be specified using `extra.time`. The default is for all sequences to start on control; if the first sequence starts on treatment at the first time point, set `all.ctl.time0` to `FALSE`. If there is a fractional treatment effect, this is defined as a vector `tx.effect.frac`; recall that a value of 1 corresponds to the standard design with no delay of full treatment effect.

The `swDsn` function outputs important information about the design that has been created. Some of the output mirrors the input, like `tx.effect.frac` and `extra.time`. The `swDsn` function also reports the total number of sequences (`n.waves`), the total number of clusters (`n.clusters`), the number of clusters per sequence (`clusters`), the total number of time points (`total.time`), and two design matrices `swDsn` and `swDsn.unique.clusters`. In the matrix labeled `swDsn` each entry is the treatment effect for that cluster at that time; 0 represents the control, and 1 represents the full treatment effect. Rows correspond to clusters, and columns correspond to time points. The matrix `swDsn.unique.clusters` is similar, but each row represents a sequence, and does not indicate how many clusters are in each sequence.

For example, the design below has four sequences with five clusters each (20 clusters total), all starting on control. Examining the design matrix makes it easy to check the design.

```
R> library("swCRTdesign")
R> design <- swDsn(clusters = c(5, 5, 5, 5))
R> design$swDsn.unique.clusters

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    1
[2,]    0    0    1    1    1
[3,]    0    0    0    1    1
[4,]    0    0    0    0    1
```

It is possible to create more complicated designs using `swDsn`. For example, here is a design with extra time between sequences crossing over, extra time at the end, and a fractional treatment effect:

```
R> design.complex <- swDsn(c(10, 0, 10, 0, 10, 0), tx.effect.frac = c(0.5),
+    extra.time = 2, all.ctl.time0 = TRUE)
R> design.complex$swDsn.unique.clusters
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    0  0.5    1  1.0    1  1.0    1    1    1
[2,]    0  0.0    0  0.5    1  1.0    1    1    1
[3,]    0  0.0    0  0.0    0  0.5    1    1    1
```

Inserting 0's in the `clusters` vector can also be used to add extra time points before any sequences have crossed over.

2. Calculating power

Power is calculated using `swPwr`:

```
swPwr(design, distn, n, mu0, mu1, sigma, tau, eta, rho, gamma, icc, cac,
  alpha = 0.05, retDATA = FALSE, silent = FALSE)
```

The function `swDsn` can be used to define the `design`. The assumed distribution of the response can be either Gaussian (`'gaussian'`) or Bernoulli (`'binomial'`), specified by `distn`. The number of observations for each cluster at each time point can be the same for all clusters and time points (`n` a scalar); differ by cluster, but the same at all time points within a cluster (`n` a vector); or differ by both cluster and time, in which case `n` is a matrix with rows corresponding to clusters and columns corresponding to time. When `n` is not a scalar, clusters should be ordered by when they cross over, with the first clusters receiving treatment listed first. The mean outcomes in the control and treatment groups, `mu0` and `mu1`, are also required to calculate power. The correlation structure of the data can be defined in two ways: through the SD parameterization (`tau`, `eta`, `rho`, `gamma`), or through ICC and CAC (`icc`, `cac`). Using ICC and CAC is only possible under the assumption that there are no random treatment effects (i.e., η=0). If $\eta = 0$ (so $\rho = 0$ also), the same results can be achieved by specifying either `tau` and `gamma` (with `eta = 0, rho = 0`) or `icc` and `cac`. If the data is Gaussian, both parameterizations require an entry for `sigma`. Attempting to mix the parameterizations (e.g., `icc = .1, tau = 0, eta = .02`) will result in an error message. The `silent = FALSE` default prints a warning to users with code that was written for earlier versions of **swCRTdesign** because of a change in the order of arguments; after any necessary adjustments this can be changed to `TRUE` to hide the warning. When `n` is not a scalar, `silent = TRUE` also represses a reminder to check the order of clusters in `n`.

3. Simulating data

Most of the arguments in `swSim` mirror those in `swPwr`:

```
swSim(design, family, log.gaussian = FALSE, n, mu0, mu1, time.effect,
  sigma, tau, eta, rho, gamma, icc, cac,
  time.lab = NULL, retTimeOnTx = FALSE, silent = FALSE)
```

The simulated data can be Gaussian, Bernoulli, or Poisson, defined by `family`. Some families allow a log link (Bernoulli and Poisson) or logit link (Bernoulli). Setting `log.gaussian = TRUE` when the family is Gaussian simulates log-Gaussian data, i.e., data whose log follows a Gaussian distribution. If there is a time trend (β_j in Equation 1), it is specified using `time.effect`. Typically, the first entry in the `time.effect` vector should be `0`, so `mu0` and `mu1` are mean outcomes in the control and treatment groups at baseline. The ICC/CAC input is only available for Gaussian families.

The user must ensure that inputs are on the correct scale as determined by the family and link. For example, if `family = binomial(link = "log")`, and the expected probability of an event in the control group is 0.1, the correct entry for `mu0` would be log(0.1). Similarly, `tau`, `eta`, `rho`, and `gamma` are all random effects added to a linear model for $\log(\mu_{ij})$, where the outcome has a Bernoulli(μ_{ij}) distribution. That is, we replaced μ_{ij} in Equation 1 with $\log(\mu_{ij})$. For log-Gaussian data, all parameters are on the log scale, e.g., `mu0` is the mean of the log-transformed outcomes in the control group, and `sigma` is the standard deviation of the log-transformed outcomes.

The data generated by `swSim` are the outcome `response.var`, the treatment effect `tx.var`, the time `time.var`, and the cluster ID `cluster.var` for each observation. Setting `retTimeOnTx` to `TRUE` produces an additional column in the output, `timeOnTx.var`, which is the cumulative time on treatment (set to 0 during control periods).

4. Summarizing and plotting data

To use `swSummary` to summarize a SWT data set, one must specify the outcome `response.var`, treatment indicator `tx.var`, time index `time.var`, and cluster ID `cluster.var` variables:

```
swSummary(response.var, tx.var, time.var, cluster.var, data, type = "mean",
  digits = 16, fcn.Call = FALSE)
```

For convenience, if a data set is passed to `data`, the four variable arguments (`response.var`, `tx.var`, `time.var`, and `cluster.var`) may be column names instead of vectors. Summaries can be provided based on the mean, the sum, or the number of individuals per cluster per time, controlled by `type`. The output of `swSummary` includes the summary statistic of interest by sequence and time (`response.wave`) or by cluster and time (`response.cluster`).

The function `swPlot` takes a SWT data set and creates a plot of the mean response over time. Data input is the same as `swSummary`:

```
swPlot(response.var, tx.var, time.var, cluster.var, data,
  choose.mfrow = NULL, by.wave = TRUE, combined.plot = TRUE,
  choose.xlab = "Time", choose.main = NULL, choose.pch = NULL,
  choose.cex = 1, choose.tx.col = NULL, choose.tx.lty = c(2, 1),
  choose.ncol = 2, choose.tx.pos = "topright", choose.legend.pos = "right")
```

The default is to plot one line for each sequence, but the mean response can be plotted for each cluster by choosing `by.wave` as `FALSE`. Sequences can be split into separate plots by choosing `combined.plot` as `FALSE`. A number of other arguments allow the user to customize labels, legends, line types, and colors if defaults are not appropriate, and are analogous to plotting options in base R; see **swCRTdesign** documentation. For convenience, `choose.legend.pos` can be set to `"mouseclick"`, which allows the user to specify the location of the legend by clicking on the plot.

5. Extended EPT trial example

This is a continuation of the EPT trial example. For the purposes of demonstration, suppose the authors now wish to simulate data for a design with transition periods, where there is one time point for each sequence where treatment has been administered but no data is collected. We can do this by recording a sample size of zero in those time periods, as follows. For simplicity, we return to the original scenario and assume there were 162 observations per cluster at every time period and a balanced design.

```
R> #Matrix of sample sizes for one cluster per sequence:
R> n.one.cluster.per.sequence <- cbind(rep(162, 4),
+    (matrix(162, ncol = 4, nrow = 4) - diag(162, nrow = 4)))
R> n.one.cluster.per.sequence

     [,1] [,2] [,3] [,4] [,5]
[1,]  162    0  162  162  162
[2,]  162  162    0  162  162
[3,]  162  162  162    0  162
[4,]  162  162  162  162    0

R> #Matrix of sample sizes for the full design:
R> n.all.clusters <- matrix(rep(n.one.cluster.per.sequence, each = 6),
+    nrow = 24)
R> #Using a matrix of sample sizes for n:
R> example.data.bernoulli.transitions <- swSim(design = design.bernoulli,
+    family = binomial(link = "log"), n = n.all.clusters,
+    mu0 = log(0.05), mu1 = log(0.035), time.effect = 0,
+    tau = 0.33, eta = 0, rho = 0, gamma = 0, silent = TRUE)
```

Transition periods can also be specified like this in `swPwr`.

6. Simulation-based power example

THIS WHOLE SECTION IS NEW

The `swSim` function allows a user to easily simulate data from one SWT. Below is an example of how to use this function to calculate power, using the **lme4** package to fit mixed models (Bates D, Maechler M, Bolker B, and Walker S; see https://CRAN.R-project.org/package=lme4).

```
R> library("lme4")
R> set.seed(74)
R> #Function: simulate one SWT, fit an appropriate mixed model,
R> #and return the p-value corresponding to the treatment effect estimate
R> OneSWT <- function(){
+    #Simulate data
+    my.data <- swSim(design = swDsn(c(6, 6, 6, 6, 6)), family = 'gaussian',
+                     n = 50, mu0 = 0, mu1 = 0.1, time.effect = 0,
+                     sigma = 1,
+                     tau = 0.1, eta = 0.2, rho = 0.01, gamma = 0.1)
+    #set up variables for random effects
+    my.data$fcluster <- factor(my.data$cluster.var)
+    my.data$ftime <- factor(my.data$time.var)
+    my.data$clustime <- interaction(my.data$fcluster, my.data$ftime)
+    #Fit mixed model
+    my.model <- lmer(response.var ~ time.var + tx.var +
+                        (tx.var | fcluster) + (1 | clustime), data = my.data)
+    #Note: (1|clustime) induces random time
+    #according to the model described in this paper;
+    #(tx.var | fcluster) induces both
+    #random treatment and random cluster effects as described.
+    #To fit a model with random cluster effects when there are not
+    #random treatment effects, (1 | fcluster) can be used.
+    #To fit the full model with rho=0,
+    #use ...+ (tx.var || fcluster) + (1|clustime).
+    return(summary(my.model)$coefficients["tx.var", 3])
+    #Return t value from treatment effect coefficient
+  }
R> #Replicate many times
R> #(using a small number of replications for this demonstration)
R> t.val.list <- replicate(n = 100, OneSWT())
R> #Calculate rejection rate, using a normal approximation
R> #For designs with large sample sizes,
R> #this should be a reasonable approximation.
R> mean(2 * (1 - pnorm(abs(t.val.list))) < 0.05)

[1] 0.46
```

In this case, although the number of replications executed here is quite small, results are similar to the closed-form `swPwr` results:

```
R> swPwr(design = swDsn(c(6, 6, 6, 6, 6)), distn = 'gaussian', n = 50,
+        mu0 = 0, mu1 = 0.1, sigma = 1,
+        tau = 0.1, eta = 0.2, rho = 0.01, gamma = 0.1, silent=TRUE)

[1] 0.4286845
```

Since `swSim` is more flexible than `swPwr` (e.g. Poisson outcomes, different links), it may sometimes be necessary to rely on simulation-based power calculations.

7. Technical details

THIS WHOLE SECTION IS NEW

The closed-form calculations implemented in `swPwr` have been tested via comparison to simulation-based power estimates, and perform as expected. The **swCRTdesign** package was first posted to CRAN in 2016 (???), and subsequent versions have corrected errors and made adjustments based on user feedback.

Computation time for `swPwr` is limited mainly by inversion of a matrix whose dimension is the product of the number of time points and the total number of clusters. This is typically trivial when computing power for a single design. However, to make a smooth power curve in the GUI, `swPwr` may be run thousands of times, and larger designs are noticably slower than SWT designs with a small number of clusters or time points. Future work could be done to reduce this computation time, since the matrix has a specific form.

Affiliation:

Emily Voldal
Department of Biostatistics
University of Washington
Department of Biostatistics, Box 357232
University of Washington
Seattle, WA 98195
E-mail: `voldal@uw.edu`