SWSamp: Simulation-based sample size calculations for a Stepped Wedge Trial (and more)

Gianluca Baio
University College London

Abstract

Sample size calculations are a fundamental part of experimental research. Often these are based on closed-form approximations, which while extremely helpful and quick to compute, can fail to accommodate for all the nuisances and specificity of some more complex study designs. One such example is the Stepped Wedge Trial (SWT), which is becoming increasingly popular. **SWSamp** is a package which aims at providing a means of applying a simulation-based approach to sample size calculations, specifically for a SWT, but with a view at rendering this process fully integrated with a wider range of data generating processes.

Keywords: Sample size calculations, simulations, Stepped Wedge Design, R.

1. Introduction

Sample size calculations are one of the fundamental components in the design of experimental studies and are mandatory in virtually all settings involving randomised trials, *e.g.* in medical research. Sample size calculations aim at determining the smallest sample that is necessary to observe, under a given design (*e.g.* distributional assumptions and expected characteristics of the intervention(s) being assessed), in order to correctly determine the "signal" (*e.g.* the "treatment effect") as statistically significant and thus not due to chance.

Usually, sample size calculations are based on analytic formulæ, often relying on some simplifying assumption, such as (asymptotic) Normality of the outcome. Once the significance level (*type I error* or false positive rate, typically $\alpha = 0.05$) and the *type II error* (false negative rate, typically $\beta = 0.2$) are set, a formula is obtained linking the "power" $(1 - \beta)$ to the expected effect, a measure of variability in the relevant population and the number of observations, n. Solving the equation for n provides the required sample size. Notice that, while slightly abusing the common notation, in order to avoid confusion we prefer to indicate α as '*sl*' and $(1 - \beta)$ as '*power*', in the rest of this paper. This allows us to use Greek letters to indicate parameters in the models we consider.

The classic example of a sample size calculation based on an analytic formula is given by the case of an outcome $y \sim \text{Normal}(\mu + \theta t, \sigma^2)$, where μ is a baseline level, $t = 0, 1$ is a "treatment" indicator, θ is the alleged treatment effect and σ^2 is the (pooled) population variance, assumed common across the two groups of treatment. Under these assumptions,

the power is determined as

$$power = \Phi\left(\sqrt{\frac{n\theta^2}{4\sigma^2}} - z_{sl/2}\right),$$

where Φ is the cumulative standard Normal distribution and $z_{sl/2}$ is its $(1-sl/2)-$th quantile. This can be solved for n to determine the optimal sample size

$$n = \frac{2\left(z_{power} + z_{1-sl/2}\right)^2}{\theta^2}.$$

While extremely helpful in principle, sample size calculations based on closed-form analytic solutions may suffer from some drawbacks. First, as a general point often the input parameters (*e.g.* the expected treatment effect or the relevant population variance) are plugged in the formula as point estimates. This does not account for any uncertainty in their "true" value, which may lead to biased estimates of the required sample size. In addition, there may be complex designs for which closed-form solutions do not exist or if they do, they rely on asymptotic results or approximations, which may render their applicability less general.

One such case is perhaps represented by the *Stepped Wedge* (SW) design, a variant of cluster randomised trials (CRTs) where all clusters receive the intervention in a randomised order — the main difference between a SW trial (SWT) and a CRT is that in the former design all groups eventually receive the intervention, while in a standard CRT the groups are randomly assigned to either treatment groups (*e.g.* intervention or control).

Figure 1 shows a very basic example of as SWT, conducted with $I = 6$ clusters and for $J = 3$ active time points at which the units are randomised to switch permanently to the treatment arm. In this case, all clusters would be measured at baseline, where all are in the control arm — this could be considered as a preliminary step before the intervention is implemented at all. Then, clusters 3 and 6 are randomised to switch to the active treatment at time point 1 and continue with this ever since. At time point 2, clusters 1 and 4 are also randomised to treatment and at time point 3 the remaining clusters 2 and 5 join the treatment arm.

Information about the randomisation scheme is included in a design matrix $\boldsymbol{X}$, in this case

$$\boldsymbol{X} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

There are several potential complications with the desing of SWTs, as systematically reviewed and analysed for example by Hargreaves et al. (2015); Beard et al. (2015) and Copas et al. (2015). These, however, are beyond the purpose of the current work and thus we do not explore them further here, unless they are directly relevant.

1.1. Analytic sample size calculations for a SWT

Analytic formulæ for sample size calculations exist for specific SWTs. For example, Hussey and Hughes (2007) considered a SW design with cross-sectional data assuming I clusters, J

```
I = number of clusters: 6
J = number of time points: 3 (not including baseline)
H = number of units randomised at each time: 2

           Baseline  Time1  Time2  Time3
Cluster 1      0        0      1      1
Cluster 2      0        0      0      1
Cluster 3      0        1      1      1
Cluster 4      0        0      1      1
Cluster 5      0        0      0      1
Cluster 6      0        1      1      1

0 = control (no intervention, Treatment as Usual, ...)
1 = active intervention
```

Figure 1: A simple example of a Stepped Wedge design, showing the the clusters' progression over time to the treatment arm

crossover points and K individuals sampled per cluster at each time point. In the most basic formulation, the observed continuous response is then modelled as $Y_{ijk} = \mu_{ij} + e_{ijk}$, where

$$\mu_{ij} = \mu + \alpha_i + \beta_j + X_{ij}\theta \tag{1}$$

is the cluster- and time-specific mean, while $e_{ijk} \sim \text{Normal}(0, \sigma^2_e)$ represent independent individual-level error terms (within-cluster variability). Here, μ is the overall intercept, $\alpha_i \sim \text{Normal}(0, \sigma^2_\alpha)$ are a set of cluster-specific random effects, β_j are fixed effects for time j, X_{ij} is an intervention indicator taking on the value 1 if cluster i is allocated to the active intervention at time j and 0 otherwise, and θ is the intervention effect. This model implies that the response Y_{ijk} is normally distributed with mean μ_{ij} and total variance $\sigma^2_y = \sigma^2_\alpha + \sigma^2_e$, while the cluster-level variance is $\frac{\sigma^2_\alpha + \sigma^2_e}{K}[1 + (K-1)\rho]$, where $\rho = \frac{\sigma^2_\alpha}{\sigma^2_\alpha + \sigma^2_e}$ is the *intra-class correlation* (ICC), a measure of the proportion of the total variance due to variation between the clusters.

For the basic model considered by Hussey and Hughes (HH), power calculations can be obtained by using the following relationship

$$power = \Phi\left(\frac{\theta}{\sqrt{V(\theta)}} - z_{sl/2}\right),$$

where $V(\theta)$ is the variance of the estimator of the intervention effect θ, defined as a relatively simple function of the design matrix $\boldsymbol{X}$. This formula can be used to determine the optimal sample size n.

1.2. Simulation-based sample size calculations

The use of a simulation-based approach to determine the optimal sample size for a study is not a new concept, nor is it specific to the design of SWTs (Gelman and Hill 2006; Burton et al. 2006; Landau and Stahl 2013). In a nutshell, the idea is to consider a model to represent

the data generating process (DGP), which describes how the researchers envisage the way in which the trial's data will eventually be observed. This should be the model that is used to analyse the data, after the study has been conducted. Using the assumed DGP, data can be simulated a large number of times and the resulting "virtual trials" can be analysed using the proposed analysis model.

Some of the parameters may be varied across the simulations: for example, it is interesting to investigate the results obtained by varying the total number of observations. The optimal sample size is set to the minimum number of subjects for which the proportion of simulated trials that correctly deem the intervention as significant at the set $sl-$level is greater than or equal to the required power.

The main advantage of using simulation-based approaches to determine the sample size is that, in principle, any DGP can be assumed, no matter how complex. Of course, trials associated with more complicated designs will also require longer computational time to produce a sufficient number of runs to fully quantify the operating characteristics, e.g. in terms of the relationship between power and sample size. This is essential to estimate the required sample size properly.

In the case of a SWT, regardless of the modelling assumptions for the outcomes or the form assumed for the cluster- and time-specific mean, the simulation procedure can be schematically described as follows.

i. Select a total sample size n (e.g. total number of individuals measured) and a suitable combination of the number of clusters I and time points J.

ii. Provide an estimate of the main parameters. These can be derived from the relevant literature or expert opinion. We recommend thorough sensitivity analyses to investigate the impact of these assumptions on the final results, in terms of optimal sample size.

iii. Simulate a dataset of size n from the assumed model.

iv. Analyse the resulting dataset and record whether the intervention effect is detected as statistically significant.

Steps *iii.* and *iv.* are repeated for a large number S of times for each of the selected values of n and the proportion of times in which the analysis correctly detects as significant the assumed intervention effects is used as the estimated power. The lowest value of n in correspondence of which the estimated power is not less than the pre-specified threshold (usually, 0.8 or 0.9) is selected as the optimal sample size. A Monte Carlo estimate of the error around the estimated power can be easily computed and used as a guideline to determine the optimal number of simulations to be used. In many situations, a value of S in the order of 1000s will suffice.

Sensitivity to the choice of the fundamental parameters can be checked by selecting different values and repeating the procedure. For example, it is possible to assess the impact of varying the cluster size. An alternative version of this algorithm may involve the adoption of a fully Bayesian approach (Spiegelhalter et al. 2004; Cunanan et al. 2016); this amounts to modelling the uncertainty in the basic parameters using suitable probability distributions. For example, one could assume that, based on currently available evidence, the between-cluster standard deviation is likely to lie in a range between two extreme values a and b. This may be translated, for example, into a prior Uniform distribution defined in (a, b). The

sample size calculations would then account for the extra uncertainty in the actual value of this parameter. The benefits of this strategy are of course higher if genuine information is available to the researchers.

Baio et al. (2015) introduce a general framework for sample size calculations in a SWT using a simulation-based approach. In this paper, we explore the technical details of that framework and introduce the R package **SWSamp**.

2. The R package SWSamp

SWSamp is designed to allow a wide range of simulation-based sample size calculations, specifically (but not exclusively!) for a SWT. In its current version, **SWSamp** consists of 5 main functions: the first one (which is currently in fact specified by three different commands) performs the analytic sample size calculations using the method of HH. This can be used as a quick alternative for more standard designs or as a first-order approximation in the case of complicated designs (*e.g.* including multiple layers of correlation). The second function replicates the sample size calculations based on the (correct form of) the "design effect" specified by Woertman et al. (2013). This too can be used very effectively in relatively standard cases, but is less efficient in cases of more complex designs. The core functions of **SWSamp** are `make.swt`, which can be used to simulate data as obtained by a reasonably wide range of possible SWTs and `sim.power`, which actually performs the simulation-based computation of the required sample size. As we will show in the following, `sim.power` can be used for other DGPs (*i.e.* not specifically for a SWT), which increases the applicability of **SWSamp**.

In order to speed up the computation process (which can be intensive, under particularly complex designs), **SWSamp** is designed to take full advantage of R multi-core processing. For this reason, **SWSamp** "imports" the `NAMESPACE` of the R packages **foreach**, **doParallel**, **iterators** and **parallel** (technically, this means that these packages need to be installed on the user's local machine, for **SWSamp** to work). In addition to these, **SWSamp** formally "depends" on the R package **lme4**, which can be used to fit linear and generalized linear mixed-effects models. This is relevant when analysing a SWT (which invariably includes some form of clustering in the DGP). Other modelling strategies could be used instead, for example Generalised Estimating Equations (GEE, Zeger and Liang 1986) or a full Bayesian model, e.g. based on Integrated Nested Laplace Approximation (INLA, Rue et al. 2009). These extensions are under development or testing in the current version of **SWSamp**.

2.1. Installation

Currently, a "stable" version of **SWSamp** is packaged and binary files are available for `MS Windows` and as source. To install the stable version on a `MS Windows` machine, run the following commands.

```
> install.packages("SWSamp",repos=c("http://www.statistica.it/gianluca/R",
+        "https://cran.rstudio.org","https://www.math.ntnu.no/inla/R/stable"),
+        dependencies=TRUE
)
```

Note that the user needs to specify a *vector* of repositories — the first one hosts **SWSamp**, while the second one should be an official `CRAN` mirror. Anyone can be selected, but a `CRAN`

mirror must be provided, so that the R command `install.packages()` can also install the "dependencies" (e.g. other packages that are required for **SWSamp** to work). The third one is used to install the package **INLA**, which can be used to perform simulation-based sample size calculations using a Bayesian approach. This process can be quite lengthy, if miss many of the relevant packages are not already installed on the user's machine.

To install from source (e.g. on a `Linux` machine), it is sufficient to run the following commands.

```
> install.packages("SWSamp",repos=c("http://www.statistica.it/gianluca/R",
+        "https://cran.rstudio.org","https://www.math.ntnu.no/inla/R/stable"),
+        type="source",dependencies=TRUE
)
```

An alternative way of installing **SWSamp** is by using the R package **devtools** and the following command.

```
> devtools::install_github("giabaio/SWSamp")
```

2.2. A simple example for a SWT: analytic sample size calculations

As a first simple example, consider a SWT based on a continuous outcome for which the experimenter is assuming the following setting:

- $J = 5$ time points at which measurements are taken, in addition to a baseline time — this amounts to a total of $J + 1 = 6$ measurements;
- An average cluster size of $K = 20$ individuals;
- A (relatively large!) cluster-level ICC $\rho = 0.5$, perhaps estimated using previous data from similar studies;
- A baseline average value for the outcome $\mu = 0.3$;
- A treatment effect $\theta = -0.3875$, implying a reduction in the outcome level for the treated units;
- A *residual* standard deviation $\sigma_e = 1.55$;
- A cross-sectional design, in which measurements are taken at discrete time points. Individuals are measured only once and at the next time point it is assumed that the sample is made by independent units (*i.e.* the case of a Intensive Care Unit, ICU, where patients are assumed to spend a small amount of time; if the distance between consecutive time points is reasonably large, it can be assumed that the sample of patients at time j is different than that observed at time $j + 1$);
- A pre-specified significance level of 0.05 and a target power of at least 0.8.

Analytic sample size calculations based on HH: Normally distributed outcome

Because of the specific design considered here, it is possible to directy use HH's exact formula to compute the power, for any given value of the number of clusters I. For instance, we could use the **SWSamp** function `HH.normal` in the following way.

```
> library(SWSamp)
> HH.normal(mu=0.3,b.trt=-0.3875,sigma=1.55,I=14,J=5,K=20,rho=.5)
```

The function `HH.normal` returns several outputs, including the estimated power, estimates for the standard deviation components and information about the assumed SWT setting.

```
$power
[1] 0.8112651

$sigma.y
[1] 2.192031

$sigma.e
[1] 1.55

$sigma.a
[1] 1.55

$setting
$setting$n.clusters
[1] 14

$setting$n.time.points
[1] 5

$setting$avg.cluster.size
[1] 20

$setting$design.matrix
   Baseline Time 1 Time 2 Time 3 Time 4 Time 5
10        0      1      1      1      1      1
4         0      1      1      1      1      1
13        0      0      1      1      1      1
12        0      0      1      1      1      1
8         0      0      1      1      1      1
6         0      0      0      1      1      1
11        0      0      0      1      1      1
9         0      0      0      1      1      1
1         0      0      0      0      1      1
2         0      0      0      0      1      1
14        0      0      0      0      1      1
7         0      0      0      0      0      1
5         0      0      0      0      0      1
3         0      0      0      0      0      1
```

The last element returned as output is a design matrix with a suggested randomisation list — in this case, clusters 10 and 4 switch to the intervention at the first active time, clusters

13, 12 and 8 switch at the second active time point and so on. The function `HH.normal` has three additional inputs: `sig.level` is the significance level and by default is set at 0.05. If this is acceptable for the user, then there is no need to specify it explicitly.

Similarly, the argument `which.var` is a text string taking values on either `'within'` (the default) or `'total'`. If nothing is specified (*e.g.*, in this case), then **SWSamp** will assume that the value given in the argument `sigma` is for σ_e, the residual standard deviation and will compute the total and cluster-specific standard deviations σ_y and σ_α, respectively, using the fact that by definition $\sigma_\alpha = \sqrt{\rho\sigma_e^2/(1-\rho)}$ and $\sigma_y = \sqrt{\sigma_\alpha^2 + \sigma_e^2}$. On the other hand, if the user sets `which.var='total'`, then the value of 1.55 will be associated with the total standard deviation σ_y; `HH.normal` then computes and $\sigma_\alpha = \sqrt{\sigma_y^2\rho}$ and $\sigma_e = \sqrt{\sigma_y^2 - \sigma_\alpha^2}$. In the case described above, since no value is specified for the argument `which.var`, **SWSamp** estimates $\sigma_e = 1.55$, the value set for the input `sigma` and because ρ is set to 0.5 then $\sigma_e = \sigma_\alpha$.

Finally, the third argument is `X`, which indicates the (re-ordered, so as to become triangular) SWT design matrix and takes the default value `NULL`. When nothing is specified, then **SWSamp** will automatically compute a "balanced" design matrix, trying to divide as evenly as possible the I clusters across the $(J+1)$ time points. In this case, the ratio I/J is not an integer and thus the allocation is done by having 2 clusters switching at the first active time point and then having 3 clusters switching at each of the remaining five active time points. This means that out of the $I(J+1) = 84$ cells of the design matrix, 48% are "1"s, indicating instances where a cluster is allocated to the intervention arm. While in this case the effect of this choice is likely to be minimal in terms of the resulting power, there may be cases where the automatic allocation favours the intervention arm (*i.e.* produces a design matrix in which the number of "1"s is much greater than the number of "0"s).

For example, suppose the user specified a design matrix using the following code

```
> X <- matrix(0,14,6)
> X[1:4,2:6] <- X[1:8,3:6] <- X[1:10,4:6] <- 1; X[1:12,5:6] <- 1; X[1:14,6] <- 1
> X
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    0    1    1    1    1    1
 [2,]    0    1    1    1    1    1
 [3,]    0    1    1    1    1    1
 [4,]    0    1    1    1    1    1
 [5,]    0    0    1    1    1    1
 [6,]    0    0    1    1    1    1
 [7,]    0    0    1    1    1    1
 [8,]    0    0    1    1    1    1
 [9,]    0    0    0    1    1    1
[10,]    0    0    0    1    1    1
[11,]    0    0    0    0    1    1
[12,]    0    0    0    0    1    1
[13,]    0    0    0    0    0    1
[14,]    0    0    0    0    0    1
```

This would imply a much higher proportion of the cells in $\boldsymbol{X}$ that are "1"s (57%). If the power was computed using this specification, the results would be slightly different than the one in

which the design matrix is automatically determined to have an allocation that is as even as possible.

```
> auto <- HH.normal(mu=0.3,b.trt=-0.3875,sigma=1.55,I=14,J=5,K=20,rho=.5)
> user <- HH.normal(mu=0.3,b.trt=-0.3875,sigma=1.55,I=14,J=5,K=20,rho=.5,X=X)
> user$power

[1] 0.8027561
```

If on the other hand, the user had specified yet another design matrix, this time defined as

```
> X2 <- matrix(0,14,6)
> X2[1:2,2:6] <- X2[1:4,3:6] <- X2[1:6,4:6] <- X2[1:8,5:6] <- X2[1:14,6] <- 1
```

(this implies that only 40% of the cells are "1"s) and then computed the analytic power, the results would be slightly different again.

```
> user2 <- HH.normal(mu=0.3,b.trt=-0.3875,sigma=1.55,I=14,J=5,K=20,rho=.5,X=X2)
> user2$power

[1] 0.7971512
```

As a general rule, the application of a "balanced" design will tend to increase the power by balancing the number of measurements in either arm of the trial.

Analytic sample size calculations based on HH: non-Normally distributed outcome

HH's computations can be brought to bear even in cases where the outcome is not directly Normally distributed, by considering asymptotic arguments. For instance, we could consider a SWT where the outcome is the presence of bacterial infection among patients in Intensive Care Units (ICUs), which are sequentially moved to a regime of specialised cleaning. In this case, we could assume that the response for the $k-$th individual in cluster i at time j is $Y_{ijk} \sim \text{Bernoulli}(\mu_{ij}, n_{ijk})$, where μ_{ij} and n_{ij} are the cluster- and time-specific probability of infection and sample size, respectively. In this case, the model specified in (1) can be generalised to

$$\phi_{ij} = g(\mu_{ij}) = \text{logit}(\mu_{ij}) = \log\left(\frac{\mu_{ij}}{1-\mu_{ij}}\right) = \alpha_i + \beta_j + X_{ij}\theta. \tag{2}$$

The model in (2) is a simple generalised mixed linear model; more complex structures may be necessary (*e.g.* to account for additional layers of clustering, or include other covariates).

To fit this model in **SWSamp** and determine the optimal sample size under given, pre-specified circumstances, it is possible to use the function `HH.binary`, for example using the following commands.

```
# Defines the basic parameters
> p1 <- 0.26            # baseline probability of the outcome in the control group
> OR <- 0.56            # assumed treatment effect (in terms of odds ratio)
```

```
> J <- 5                # number of time points (excluding the baseline measurement)
> K <- 20               # assumed average cluster size
> rho <- 0.3            # assumed intra-class correlation
> sig.level <- 0.05     # assumed significance level
> which.var <- "within" # specifies whether the variance is within or total

# Computes the power using the analytic formula
> HH.binary(p1,OR,I=8,J,K,rho,sig.level,which.var,X=NULL)
```

which returns the following output

```
$power
[1] 0.5276896

$p1
[1] 0.26

$p2
[1] 0.1644083

$sigma.y
[1] 0.485341

$sigma.e
[1] 0.4060654

$sigma.a
[1] 0.2658322

$setting
$setting$n.clusters
[1] 8

$setting$n.time.points
[1] 5

$setting$avg.cluster.size
[1] 20

$setting$design.matrix
  Baseline Time 1 Time 2 Time 3 Time 4 Time 5
8        0      1      1      1      1      1
7        0      0      1      1      1      1
5        0      0      1      1      1      1
4        0      0      0      1      1      1
3        0      0      0      0      1      1
2        0      0      0      0      1      1
```

```
6        0      0      0      0      0      1
1        0      0      0      0      0      1
```

This works pretty much in the same way as the function `HH.normal` and in fact most of the inputs are the same. What does change is the set of input values that are specific to the type of outcome: `p1` instead of `mu` and `OR` instead of `b.trt`. In addition, the Bernoulli model does not require the specification of the residual variance and thus there is no need to provide the argument `sigma` to the call to the function `HH.binary`.

As suggested by Baio et al. (2015), a related issue concerns the fact that, in the binary and count outcome cases, it is more cumbersome to assume that information is provided in terms of the total variance. This is because, unlike the Normal distribution, the Bernoulli and Poisson distributions are characterised by a single parameter, which simulataneously determines both the linear predictor and the variance. Consequently, because the linear predictor includes the cluster-specific random effects α_i, assuming a fixed total variance σ^2_y implies in (2) a re-scaling of the baseline value μ to guarantee that the resulting total variance approximates the required value. For this reason, when using a simulation-based approach for non-normally distributed outcomes it is easier to provide information on the within-cluster variance σ^2_e as input.

In the case of count outcomes, **SWSamp** allows analytic computations using the function `HH.count`, which works in a very similar way. A call to this function would be done by using the following code.

```
HH.count(lambda1,RR,I,J,K,rho,sig.level,which.var,X)
```

where `lambda1` is the baseline (controls) rate of occurrence of the event under investigation; `RR` is the treatment effect measured in terms of the relative risk; and similar arguments apply to the other inputs.

Analytic sample size calculations based on the Design Effect

The **SWSamp** function `DE.woert` can be also used to compute the "Design Effect" (DE) for a SWT, based on the (correct) formulation of the equation provided by Woertman et al. (2013) and presented in Baio et al. (2015). In a nutshell, the DE is an inflation factor that corrects the analytic sample size based on an individual-randomised trial to account for the specific form of correlation induced by the SW design.

Consider for example a SWT in which the outcome is a binary variable and the treatment effect is assumed to translate to an odds ratio of 0.53 for the outcome of interest. The main inputs that are required for `DE.woert` are:

- `outcome`: a string (default `'cont'` to indicate a continuous outcome, but possible values are `'bin'` or `'count'`, for binary or count data, respectively);
- `input`: a R list containing the following arguments:
  - For a continuous outcome
    1. `delta`: treatment effect;
    2. `sd`: pooled population standard deviation.
  - For a binary outcome

1. `p1`: baseline probability of outcome;
2. either `p2` (treatment probability of outcome), or `OR` (treatment effect as odds ratio).

– For a count outcome

1. `r1`: baseline rate of outcome;
2. either `r2` (treatment rate of outcome), or `RR` (treatment effect as relative risk).

- `K`: the average cluster size;
- `J`: the number of time points (excluding baseline);
- `B`: the number of baseline measurement times (default = 1);
- `T`: the number of measurement times during each crossover point;
- `rho`: the ICC;
- `sig.level`: the significance level (default at 0.05);
- `power`: the required power (default at 0.8).

The following is a simple example of how the function `DE.woert` can be used.

```
> input <- list(OR=.53,p1=.26)
> K <- 20
> J <- 5
> rho <- .2
> x <- DE.woert(outcome="bin",input=input,K=K,J=J,rho=rho)
> x

$n.cls.swt
[1] 11

$n.pts.swt
[1] 1221.568

$DE.woert
[1] 2.513514

$CF
[1] 0.4189189

$n.rct
[1] 486
```

The output of this function is: `n.cls.swt`, the number of clusters required for a SWT (in this case, 11); `n.pts.swt`, the total number of subjects that need to be observed under the current design (1221); `DE.woert`, the design effect due to clustering (2.51); `CF`, the correction

factor attributable to the use of the SWT (0.41); and finally `n.rct`, the number of subjects that would be included in a RCT with the same power as that of the SWT.

2.3. Simulation-based sample size calculations

The **SWSamp** function `sim.power` can be used for different choices of the parameter `I`, indicating the number of clusters and assess which of these is the minimum required to hit the pre-specified power.

For example, the following code can be used to estimate the power corresponding to such a design including $I = 14$ clusters, using the following R call:

```
> library(SWSamp)
> x <- sim.power(I=14,J=5,K=20,rho=0.5,mu=0.3,sigma.e=1.55,
+ b.trt=-0.3875,n.sims=100)
```

which would simulate and analyse `n.sims`=100 "virtual trials" with the given design. The object `x` is a R list containing the following elements:

- `power`: the estimated power for the current configuration. If the model does include random effects (which is the case for a SWT), then **SWSamp** assesses whether the "true" effect is correctly detected by computing the $100 \times (1 - sl)\%$ interval around the point estimate for the "treatment effect" and checking whether it is entirely above or below 0 (in which case, the results are deemed as "significant" for that particular simulated trial) The average of the simulations is the estimate of the power. The choice of focussing on estimation rather than hypothesis testing is also justified because it is in general difficult to assess the correct degrees of freedom of the resulting (linear) mixed model. The p-value could be computed using the approximation in Satterthwaite (1946), or by using a rougher Normal approximation. Neither however is free from problems, as suggested by Pinheiro and Bates (2000).
- `time2run`: the computational time (in seconds);
- `ci.power`: an estimated $100 \times (1 - sl)\%$ interval for the power. This is based on a relatively crude Normal approximation and computed using the results from the simulation procedure;
- `theta`: the estimated treatment effect from the simulation procedure, together with an estimate of its variability;
- `rnd.eff.sd`: the estimated standard deviation components, including the cluster and residual values indicated by σ_a and σ_e, respectively, in the model described in equation (1);
- `setting`: a R list summarising the assumptions in terms of number of clusters, time points, type of model, formula used.

These can be explored by typing in R:

```
> x
```

which returns the following output

```
$power
[1] 0.8

$time2run
Time to run (secs)
             1.857

$ci.power
[1] 0.7212065 0.8787935

$theta
      Estimate Standard Error
    -0.3660754      0.1362529

$rnd.eff.sd
[1] 1.534364

$setting
$setting$n.clusters
[1] 14

$setting$n.time.points
[1] 5

$setting$avg.cluster.size
[1] 20

$setting$design
[1] "cross-sec"

$setting$formula
y ~ treatment + factor(time) + (1 | cluster)
<environment: 0xd6aa96c>

$setting$method
[1] "lmer"

$setting$family
[1] "gaussian"
```

The command `sim.power` has also some additional arguments that can be used to produce variants of the default setting or extra outputs. For example, the user can specify a string `family` to define the type of outcome: the default value is `"gaussian"`, but other possibilities are `"binomial"` and `"poisson"`, in which case the procedure runs a suitable generalised linear model.

Another option is related to the type of design to be considered, which can be specified in the input `design`; currently, the options are `"cross-sec"` (the default choice) or `"cohort"` (which would consider a repeated measures design).

Similarly, the user can specify a particular *formula* to describe the relationship between the outcome and the predictors. If nothing is specified, **SWSamp** assumes that the user wants to consider a model `formula=y~treatment+factor(time)+(1|cluster)`. This uses the notation of the R package **lme4** and considers an outcome named `y`; a "fixed effect" for a variable named `treatment`; a "fixed effect" for a categorical (factor) variable named `time`; and a "random effect" to account for correlation within the levels of a variable named `cluster`. The notation `1|cluster` instructs **lme4** that this model should consider a varying intercept for each levels of the clustering variable. When the user defines a custom formula, the name of the "treatment" variable can be specified as a string in the extra argument `treat.name`. If this is not present, then it is assumed that the name of the treatment variable is, unsurprisingly, `"treatment"`.

By default, `sim.power` performs an analysis based on a frequentist approach; if the model does contain clustering this is done using **lme4**, while for simpler structures (e.g. without random effects), the standard `(g)lm` function is used. However, if the user specifies `method="inla"`, **SWSamp** performs the analysis of the simulated datasets fitting a Bayesian model using Integrated Nested Laplace Approximation. In this case, the resulting values are estimated from the relevant posterior distributions. Future developments will explore the use of Hamiltonian Monte Carlo to perform Bayesian analysis.

To help in the computationally intensive task of analysing a very large number of simulated dataset, the user can take advantage of R multi-core capability, by specifying the option `n.cores=K`, where `K` is the number of CPUs to be used for parallel processing. By default, `n.cores` is set to `NULL`, which means that R will figure out the number of available cores and use all but one, to keep the machine sufficiently responsive.

Finally, the option `plot` (default value at `FALSE`) instructs **SWSamp** on whether a plot showing the performance of the simulation procedure should be printed or not. If `plot=TRUE`, then a plot is created with the moving average of the computed power at successive batches of iteration (every 10%). This can be useful to assess whether the procedure has reached some sort of "convergence" to a stable value of the computed power.

2.4. User-defined data generating processes

Of course, each trial may have some characteristic that is just specific to the particular setting under consideration. Thus, **SWSamp** allows the user to specify a data generating process to be used in the analysis of the "virtual trials" and thus the estimation of the resulting power/required sample size.

As a first example, consider a very simple trial based on a continuous outcome with two treatment groups: the first one (coded as $t = 0$) is the control arm of the trial (*e.g.* individuals subject to the status quo, or a placebo), while the second one ($t = 1$) is the group of individuals given the active treatment.

This situation can be again modelled by assuming $y \sim \text{Normal}(\mu + \theta t, \sigma^2)$, where μ is the baseline outcome for the controls and θ is the treatment effect. The user could specify a function in R, something like

```
> simple.trial <- function(n,mu=0,theta,sigma) {
```

```
+  x <- rbinom(n,1,.5)
+  linpred <- mu+theta*x
+  y <- rnorm(n,linpred,sigma)
+  return(data.frame(y=y,x=x))
+ }
```

The function `simple.trial` takes as arguments the overall sample size `n`, the baseline mean outcome `mu` (for which the default value is specified as 0), the treatment effect `theta` and the common standard deviation in the two groups `sigma`. For example, a R call

```
> data <- simple.trial(n=10,mu=0,theta=1,sigma=1)
```

returns the simulated dataset

```
            y x
1  -1.8201517 0
2   0.1865416 1
3   0.9559052 0
4   2.2738202 1
5   0.3133810 0
6   2.3277645 1
7  -0.6531422 0
8   1.6258888 1
9   0.4091159 1
10  0.5854884 1
```

For such a design, it is possible to determine the optimal sample size analytically, for instance using the R function `power.t.test`, *e.g.* calling

```
> power.t.test(delta=1,sd=1,sig.level=0.05,power=0.8)

     Two-sample t test power calculation

              n = 16.71477
          delta = 1
             sd = 1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

which suggests that a total sample size of $n = 34$ individuals is required (notice that the function `power.t.test` indicates the treatment effect by means of the parameter `delta`).

The user can also perform this calculation based on simulations. For example, typing

```
> x <- sim.power(data=simple.trial,inpts=list(n=34,theta=1,sigma=1),
+ formula=y~x,treat.name="x",n.sims=1000)
> x
```

returns the output

```
$power
[1] 0.824

$time2run
Time to run (secs)
            0.787

$ci.power
[1] 0.8003851 0.8476149

$theta
      Estimate Standard Error
     0.9991047      0.3425531

$sd.comps
NULL

$setting
$setting$formula
y ~ x

$setting$method
[1] "lm"

$setting$family
[1] "gaussian"
```

This time, the call to the function `sim.power` has different arguments than in the standard case. In fact, it is possible to specify the optional argument `data` which is simply the name of the user-defined function telling R what the data generating process to be used to create the "virtual trials" is; this can be complemented by the argument `inpts`, a R list including the values that the user wants to assign to the inputs of the function defined in `data`. In this case, `simple.trial` requires that the values for `n`, `theta` and `sigma` are given (and if nothing is specified for `mu`, the default value is taken). If the function specified as `data` had no input, then the optional argument `inpts` would not be necessary; if the user does not specify it, `sim.power` will assume that it is not needed and will automatically set it to `NULL`.

When considering user-defined functions for the data generating process, it is probably necessary to specify a `formula` to instruct R and **SWSamp** as to what is the underlying model that should be used for the analysis of each simulated trial. In this simple case, we specify `formula=y~x`. In addition to this, **SWSamp** needs to know which of the covariate(s) is the "treatment" — this is because it needs to compute the "treatment effect" θ and then the resulting power based on whether it is detected as "significantly" different from 0. If the optional argument `treat.name` is not specified, `sim.power` assumes that one of the covariates is called `'treatment'`; this is not the case in the current example and thus the user needs to tell `sim.power` that in fact the variable called `'x'` is the one for which the treatment effect

should be computed.

Finally, we can specify the number of simulations to be used, in this case we selected `n.sims=1000` — notice that since this is the default value, the inclusion of this extra argument is not essential.

While analysing the simulated data, **SWSamp** checks that the model specification contains structured (random) effects, in the syntax of the R package **lme4**, *i.e.* using the form `(1|x)` to indicate a varying intercept for a covariate `x`, or `(z|x)` to indicate a random slope for the covariate `z`, depending on the levels of the covariate `x`. If random effects are present, **SWSamp** will analyse the simulated dataset using `(g)lmer`, the **lme4** functions to fit (generalised) linear mixed models. If there are no random effects in the model specification (*e.g.* in the present example), then **SWSamp** will analyse the simulated datasets using a simple (generalised) linear model, *i.e.* the R command `(g)lm`.

The choice of the distributional assumption is governed by the extra argument `family` that the user can specify to instruct **SWSamp** on the nature of the outcome. The default value is `family='gaussian'`, indicating that the outcome is associated with a Normal distribution and thus the resulting analysis will be performed using a linear model (*i.e.* using `lmer`, in the presence of random effects). Other options include `family='binomial'` or `family='poisson'`, in which cases a suitable generalised linear (mixed) model is used.

For example, consider again a very simple DGP in which the outcome variable is $y \sim \text{Binomial}(\pi, n)$, with $\text{logit}(\pi) = \mu + \theta t$ and where μ and θ represent the baseline log-OR for the event described by y (which applies to the controls) and the log-OR for the treatment effect, assuming $t = 0, 1$ is the treatment, respectively.

A simple R function to simulate this DGP is the following

```
> bin.trial <- function(n,p1,OR) {
+  p2 <- OR*(p1/(1-p1))/(1+OR*(p1/(1-p1)))
+  x <- rbinom(n,1,.5)
+  linpred <- log(p1/(1-p1)) + log(OR)*x
+  pi <- exp(linpred)/(1+exp(linpred))
+  y <- rbinom(n,1,pi)
+  return(data.frame(y=y,x=x))
}
```

The function `bin.trial` takes as inputs the sample size `n`, the baseline *probability* of the outcome `p1` and the *odds ratio* `OR` — obviously

$$\mu = \log\left(\frac{p_1}{1-p_1}\right) \qquad \text{and} \qquad \theta = \log(\text{OR}).$$

The power for this trial can be computed using `sim.power` by specifying

```
> x <- sim.power(data=bin.trial,inpts=list(n=n,p1=p1,OR=OR),
+ formula=y~x,treat.name="x",family="binomial")
```

assuming that suitable values for `n`, `p1` and `OR` are specified by the user. For instance, setting `n` $= 100$, `p1` $= 0.54$ and `OR` $= 1.3$ and running `sim.power` would return the following results

```
$power
[1] 0.107

$time2run
Time to run (secs)
            1.127

$ci.power
[1] 0.08783173 0.12616827

$theta
      Estimate Standard Error
     0.2847529      0.4122627

$sd.comps
NULL

$setting
$setting$formula
y ~ x

$setting$method
[1] "glm"

$setting$family
[1] "binomial"
```

As a more complex example, we could actually define a function that creates data from a SW design. **SWSamp** has a built-in function for data generating processes that is suitable for a SWT; this is called `make.swt` and basically takes as inputs many of the arguments that are specified in the default call to `sim.power`. For example, we could define

```
> sw.trial <- function(){
+  make.swt(I=8,J=5,K=10,mu=.3,b.trt=-.3875,sigma.e=1.55,rho=.4)
+ }
> data <- sw.trial()
> rbind(head(data),tail(data))

             y person time cluster treatment    linpred   b.trt
1    0.3609523      1    0       1         0 -1.1180467 -0.3875
2    0.1632296      2    0       1         0 -1.1180467 -0.3875
3   -2.0574118      3    0       1         0 -1.1180467 -0.3875
4    1.2822661      4    0       1         0 -1.1180467 -0.3875
5   -2.7932521      5    0       1         0 -1.1180467 -0.3875
6   -2.7640185      6    0       1         0 -1.1180467 -0.3875
475 -1.8814225      5    5       3         1 -0.5500236 -0.3875
476  2.9808526      6    5       3         1 -0.5500236 -0.3875
```

```
477  0.9249720      7    5       3         1 -0.5500236 -0.3875
478 -0.7509720      8    5       3         1 -0.5500236 -0.3875
479 -3.3756894      9    5       3         1 -0.5500236 -0.3875
480 -3.7219299     10    5       3         1 -0.5500236 -0.3875
```

This function generates a dataset that could arise from a SWT with $I = 8$ clusters, $J = 5$ time point (excluding the baseline), an average cluster size of $K = 10$ individuals, a baseline outcome of $\mu = 0.3$, a treatment effect of -0.3875, a cluster-level ICC $\rho = 0.4$ and assuming a *residual* variance $\sigma_e = 1.55$. Executing the command `sw.trial()` generates the full dataset (which, incidentally contains 480 rows). Notice that since all the relevant inputs are defined inside the function `sw.trial`, there are no arguments to be added to the call to the function. As no argument `design` is specified in the call to `make.swt`, **SWSamp** assumes that the default `cross-sec` should be used, implying that the resulting dataset is constructed under a cross-sectional design, with individuals observed only once.

The function `sw.trial` can be used as an argument to `sim.power` to estimate the power for this particular SWT. For example, the user can type

```
> x <- sim.power(data=sw.trial)
> x


$power
[1] 0.32

$time2run
Time to run (secs)
            11.209

$ci.power
[1] 0.2910736 0.3489264

$theta
      Estimate Standard Error
    -0.3835148      0.2530941

$sd.comps
cluster (Intercept)            Residual
          0.9874177           1.6554100

$setting
$setting$formula
y ~ treatment + factor(time) + (1 | cluster)
<environment: 0xe79aa20>

$setting$method
[1] "lmer"
```

```
$setting$family
[1] "gaussian"
```

In this case, `n.sims`=1000 simulations are used (since no value is selected for this argument), which combined with the more complex DGP associated with the SWT implies a longer running time (about 11 seconds). The power for this configuration is estimated at 0.33 and likely to range between 0.30 and 0.36. This is in line with the fact that the estimates of the treatment effect `theta` (-0.38) is not obtained with large precision — the standard error is 0.25, which means an approximate 95% interval is $[-0.89; 0.13]$, *i.e.* it crosses the threshold value of 0.

3. Conclusions

SWSamp is designed with the specific aim of dealing with sample size calculations for a Stepped Wedge trial; this implies the need to account for complications in the underlying set up, including different layers of structured effects, time trends and other issues that may be complex to comprehensively address using closed-form calculations.

However, we have made an effort to code the core functions so that the user can actually extend the general framework and apply the methods to a wider range of designs. Ideally, suitable functions specifying different data generating process will be made available (and the hope is to create a repository to which user can contribute) so that the process can be made extremely general and widely applicable in practice.

Simulations-based sample size calculations can be very useful in real-world applications, particularly with a view at aligning the model assumed to describe how the data are generated in the context of the trial under investigation and the one used to analyse the data, once they become available. Future extensions to the package involve the use of more complex analysis models (such as GEE or full Bayesian models).

References

G. Baio, A. Copas, G. Ambler, J. Hargreaves, E. Beard, and R. Omar. Sample size calculation for a stepped wedge trial. *Trials*, 16:354, Aug 2015. doi:10.1186/s13063-015-0840-9. URL http://www.trialsjournal.com/content/16/1/354.

E. Beard, J. Lewis, A. Prost, A. Copas, C. Davey, D. Osrin, G. Baio, J. Thompson, K. Fielding, R. Omar, S. Ononge, and J. Hargreaves. Stepped wedge randomised controlled trials: Systematic review. *Trials*, 2015.

A. Burton, D. Altman, P. Royston, and R. Holder. The design of simulation studies in medical statistics. *Statistics in Medicine*, 25:4279–4292, 2006.

A. Copas, J. Lewis, J. Thompson, C. Davey, K. Fielding, G. Baio, and J. Hargreaves. Designing a stepped wedge trial: three main designs, carry-over effects and randomisation approaches. *Trials*, 2015.

K. M. Cunanan, B. P. Carlin, and K. A. Peterson. A practical Bayesian stepped wedge design for community-based cluster-randomized clinical trials: The British Columbia Telehealth Trial. *Clin Trials*, 13(6):641–650, Dec 2016.

A. Gelman and J. Hill. *Data analysis using regression and multilevel/Hierarchical models*. Cambridge University Press, Cambridge, UK, 2006.

J. Hargreaves, A. Copas, E. Beard, D. Osrin, J. Lewis, C. Davey, J. Thompson, G. Baio, K. Fielding, and A. Prost. Five questions to consider before conducting a stepped wedge trial. *Trials*, 2015.

M. Hussey and J. Hughes. Design and analysis of stepped wedge cluster randomised trials. *Contemporary Clinical Trials*, 28:182–191, 2007.

S. Landau and S. Stahl. Sample size and power calculations for medical studies by simulation when closed form expressions are not available. *Statistical Methods in Medical Research*, 22 (3):324–345, 2013.

J. Pinheiro and D. Bates. *Mixed-effects models in S and S-PLUS*. Springer, New York, NY, 2000.

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society: Series B (statistical methodology)*, 71(2):319–392, 2009.

F. E. Satterthwaite. An Approximate Distribution of Estimates of Variance Components. *Biometrics Bulletin*, 2:110–114, 1946.

D. Spiegelhalter, K. Abrams, and J. Myles. *Bayesian Approaches to Clinical Trials and Health-Care Evaluation*. Wiley and Sons, London, UK, 2004.

W. Woertman, E. de Hoop, M. Moerbeek, S. Zuidema, D. Gerritsen, and S. Teerenstra. Stepped wedge designs could reduce the required sample size in cluster randomized trials. *Journal of Clinical Epidemiology*, 66(7):52–58, 2013.

S. Zeger and K. Liang. Longitudinal data analysis for discrete and continuous outcomes. *Biometrics*, 42(1):121–130, Mar 1986.

Affiliation:

Gianluca Baio
Department of Statistical Science
University College London
Gower Street, London, WC1E 6BT (UK)
E-mail: g.baio@ucl.ac.uk
URL: http://www.statistica.it/gianluca