Randomization Inference with **permuter**

Dustin J. Rabideau

February 11, 2019

The **permuter** package was developed to carry out randomization inference for a treatment effect in cluster randomized trials (CRTs) (Rabideau and Wang, 2019). In this vignette, we introduce the package and illustrate some of its functionality.

1 Introduction

The **permuter** package contains various functions to calculate p-values and confidence intervals (CIs) for common regression models using randomization (or randomization-based, permutation) inference. For example, `permtest_glm()` and `permci_glm()` correspond to `stats::glm()`, while `permtest_coxph()` and `permci_coxph()` correspond to `survival::coxph()`. Much of the syntax and arguments are kept consistent between the randomization-based functions and their counterparts. For example, to fit a simple logistic regression model, we could use

```
glm(outcome ~ exposure, family = binomial, data = ds)
```

To carry out randomization inference for this model, we instead use

```
permtest_glm(outcome ~ exposure, family = binomial, data = ds, ...)
permci_glm(outcome ~ exposure, family = binomial, data = ds, ...)
```

There are a few necessary (and other optional) arguments in `...` that we specify for the randomization-based functions (more on this later), but the basic formulation should be familiar. One optional argument worth mentioning here is `ncores`, which allows us to carry out randomization inference in parallel across multiple cores using functionality from **doParallel** and **doRNG**. E.g. we can specify `permtest_glm(..., ncores = 3)` to run the randomization test in parallel across 3 cores.

2 The Basics

Let's learn the basics of using **permuter** with an example. The Pneumococcal Conjugate Vaccine Trial was a CRT carried out from 1997 to 2000 to assess the safety and efficacy of a seven-valent conjugate pneumococcal vaccine (O'Brien et al., 2003). The study population

was Navajo and White Mountain Apache children younger than 2 years, a group with one of the highest documented rates of invasive pneumococcal disease in the world at that time. A total of 38 geographic areas were randomized: 19 areas were offered pneumococcal vaccine and 19 were offered a comparator (meningococcal vaccine). One individual-level outcome measured during the trial was the total number of bacterial pneumonia episodes experienced by each child during follow-up. We will analyze this count outcome for a random subsample of 449 children drawn from the original 8,292 trial participants (Hayes and Moulton, 2017). These data are available in the **permuter** package and here on Harvard Dataverse.

The `pneumovac` data frame contains columns for the individual-level count outcome (`bpepisodes`), an indicator of randomization to the pneumococcal vaccine (`spnvac`), and a distinct identifier for each geographic area (`randunit`) and individual (`fakeid`).

```
library(permuter)
head(pneumovac)

##   randunit bpepisodes spnvac  fakeid
## 1        1          0      1 3771740
## 2        1          0      1 3784845
## 3        1          0      1 3792151
## 4        1          0      1 3823072
## 5        1          0      1 3847406
## 6        1          1      1 3869199
```

2.1 Randomization Test

Let's use randomization inference to determine whether there was a difference in the rate of bacterial pneumonia episodes between the two intervention groups. The test statistic we will use is the estimated log incidence rate ratio (IRR) from a Poisson generalized linear model (GLM). Let's calculate the p-value based on 1,000 permutations. To carry out this randomization test, we'll use the `permtest_glm()` function in the **permuter** package.

```
test <- permtest_glm(bpepisodes ~ spnvac, trtname = 'spnvac',
                     runit = 'randunit', family = poisson, data = pneumovac,
                     nperm = 1000, ncores = 1, seed = 444)
print(test) # logIRR

##     spnvac     p.perm
## -0.4466939  0.0780000

print(exp(test['spnvac'])) # IRR

##    spnvac
## 0.6397397
```

The `formula` (first argument), `family`, and `data` arguments are passed to the corresponding regression function, `stats::glm()`; `trtname` specifies the column name of the randomized treatment variable and `runit` specifies the column name of the unit of randomization identifier (e.g. cluster id); `nperm` specifies the number of permutations used for the Monte Carlo approximation of the exact p-value (Dwass, 1957). To speed up compuation, we could instead specify a larger integer value for `ncores` to run the randomization test in parallel across multiple cores. Finally, the `seed` argument is passed to `set.seed()` (if `ncores` = 1) or `doRNG::registerDoRNG()` (if `ncores` > 1) to make results reproduceable (i.e. use a particular set of permutations).

We get an estimated IRR of 0.64 (logIRR = -0.45) with a p-value of 0.078.

2.2 Randomization-Based Confidence Interval

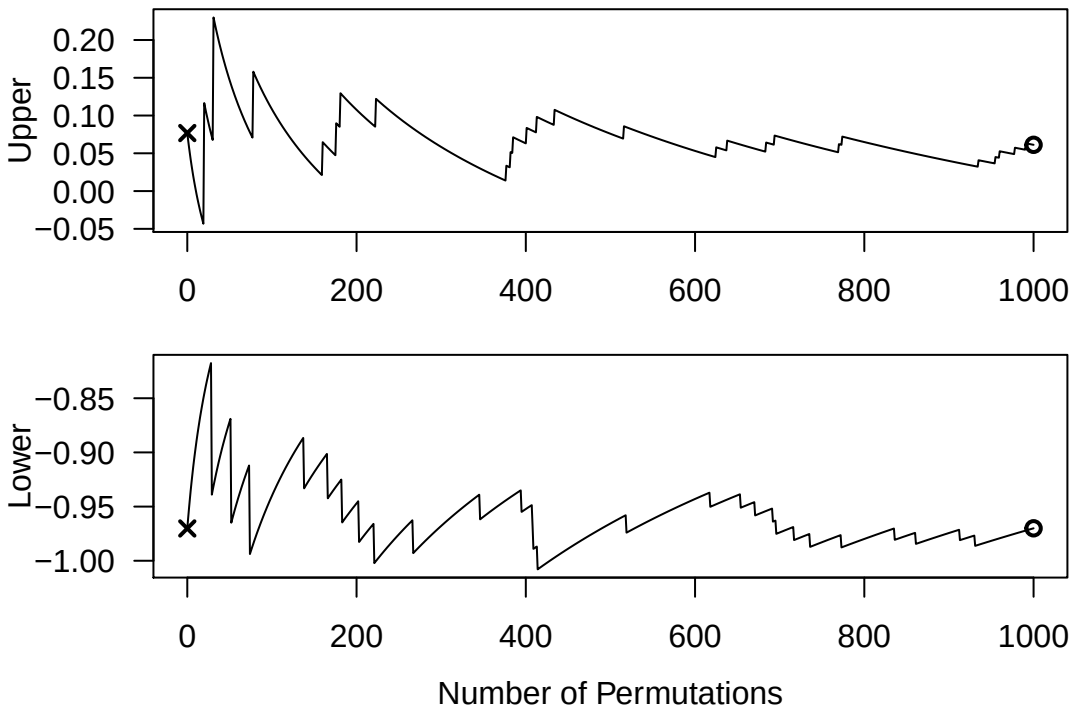
Now let's get the corresponding randomization-based 95% CI using `permci_glm()`.

```
ci <- permci_glm(bpepisodes ~ spnvac, trtname = 'spnvac',
                 runit = 'randunit', family = poisson, data = pneumovac,
                 nperm = 1000, ncores = 2, seed = 445, level = 0.95,
                 initmethod = 'perm')
print(ci$ci) # logIRR

##       lower       upper
## -0.97004101  0.06123393

print(exp(ci$ci)) # IRR

##     lower     upper
## 0.3790675 1.0631476
```

Most of the arguments are the same except the additional specification of the confidence `level` (95% here) and `initmethod`, which specifies the method to obtain initial values for the CI procedure (see `permci_glm()` documentation for more detail). If `ncores` > 1 for `permci_glm()`, lower and upper bound search procedures run in parallel across two cores (but not more than two).

We get a 95% CI for the IRR of 0.37 to 1.07.

In **permuter**, we use an efficient algorithm (Garthwaite, 1996) based on the Robbins-Monro search process to calculate the CI. We can inspect convergence of each bound simply by using `plot(ci)`:



Note: we've used only 1,000 permutations in this example for convenience. In practice, we'll usually need to increase the number of permutations to ensure adequate approximation of the exact p-value and convergence of the CI bounds.

3 Going a Bit Deeper

3.1 Stratified Randomization

CRTs often employ stratified randomization to improve balance between treatment arms, which can increase the power of the trial. Stratified designs can be accomodated simply by restricting the set of permutations considered. In **permuter**, this just means including one additional argument, `strat`, when calling the test or CI function of interest.

To illustrate this, let's consider the Botswana Combination Prevention Project (BCPP), a pair-matched HIV prevention CRT to test whether a combination treatment and prevention intervention could reduce population-level cumulative HIV incidence over 3 years of follow-up. A total of 30 communities were randomized, 15 to each arm. The primary study endpoint was cumulative HIV incidence, measured at scheduled study visits as time to HIV-infection within a cohort of individuals identified as HIV-negative among a 20% random sample of eligible households at baseline. That is, we have an interval-censored time-to-event outcome for each cohort participant. Since the primary trial results have not yet been published, we use a simulated data set, which was generated to mimic the BCPP by applying an agent-based epidemic model to a dynamic network of simulated sexual partnerships (Goyal, Blitzstein, and De Gruttola, 2013; Wang et al., 2014). These data are also available in the **permuter** package.

The `bcpp` data frame contains an indicator for assigment to the intervention arm (`treat`), a distinct identifier for community (`group`) and matched pair (`pair.id`), and two columns representing the interval-censored time-to-event outcome (`left`, `right`) measured in weeks.

```
head(bcpp)

##     group pair.id treat      left    right
## 59      1       1     0 169.57143      Inf
## 65      1       1     0 170.14286      Inf
## 136     1       1     0 164.14286      Inf
## 127     1       1     0 196.42857      Inf
## 17      1       1     0  43.57143 95.71429
## 199     1       1     0 198.57143      Inf
```

Let's carry out randomization inference for these data, making sure we restrict the analysis based on the pair-matched design by including `strat = 'pair.id'`. We'll use the estimated treatment effect from an interval-censored Weibull regression model as our test statistic.

```
library(survival)
test <- permtest_survreg(Surv(left, right, type = 'interval2') ~ treat,
                         trtname = 'treat', runit = 'group',
                         strat = 'pair.id', data = bcpp, dist = 'weibull',
                         nperm = 1000, ncores = 3, seed = 446)
ci <- permci_survreg(Surv(left, right, type = 'interval2') ~ treat,
                     trtname = 'treat', runit = 'group',
                     strat = 'pair.id', data = bcpp, dist = 'weibull',
                     nperm = 1000, ncores = 2, seed = 447)
print(test)

##     treat    p.perm
## 0.5047558 0.0010000

print(ci$ci)

##     lower     upper
## 0.3209915 0.7079766
```

These results suggest significantly better outcomes (i.e. longer average times to HIV-infection) among those living in communities randomized to the intervention. Note, we didn't explicitly set `level = 0.95` or `initmethod = 'perm'` this time, but these are the default values.

Matching and stratification are just special forms of *restricted* randomization, which is also handled by restricting the set of permutations in the analysis. We plan to implement this more general functionality in future versions of **permuter**.

3.2 Fine-Tuning the Confidence Interval

The efficient CI algorithm (Garthwaite, 1996) used in **permuter** substantially reduces the computational burden of obtaining randomization-based CIs, but its performance may be sensitive to some of the choices that have been made under the hood. For example, one may wish to consider different starting values, restart the algorithm if the starting values seem poor, or modify the magnitude of each step in the algorithm. Let's see how we can fine-tune our CIs in **permuter**.

Let's generate some data from a CRT with a binary outcome.

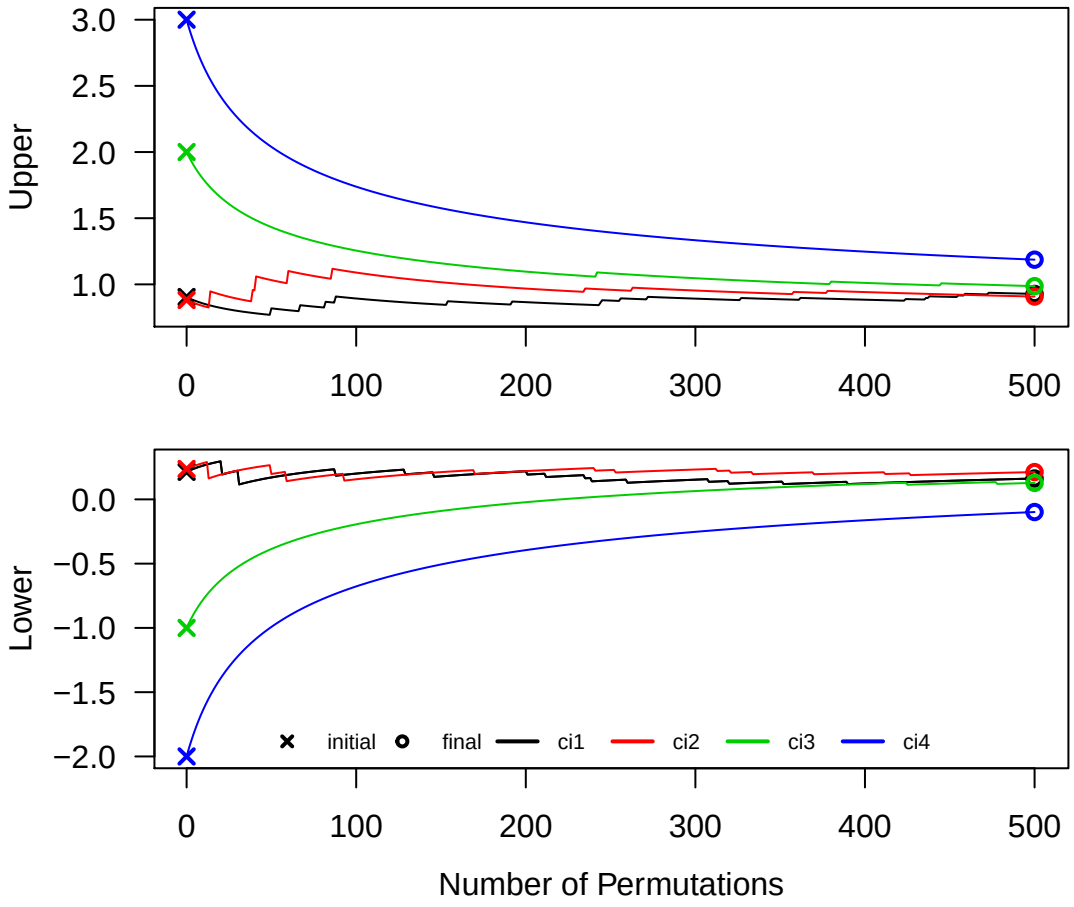
```
ds <- gendata_crt(family = binomial, nclus = c(10, 10), size = c(30, 50),
                  theta = log(1.5), mu = qlogis(0.25), sigma = 0.2)
dim(ds)

## [1] 786   5

head(ds)

##   unique.id clusid id trt y
## 1       1.1      1  1   1 0
## 2       1.2      1  2   1 0
## 3       1.3      1  3   1 0
## 4       1.4      1  4   1 0
## 5       1.5      1  5   1 1
## 6       1.6      1  6   1 0
```

We've created a data set with 10 clusters in each study arm, ranging in size from 30 to 50 individuals. Other arguments control the true underlying treatment effect (e.g. an odds ratio of 1.5), the prevalence the outcome in the control group (e.g. 25%), and the level of between-cluster hetergeneity (see `gendata_crt()` documentation for more details).

Let's compute four CIs with different sets of starting values: the first based on a small permutation test, the second based on a naive asymptotic CI, and the last two specifying the initial values ourselves.

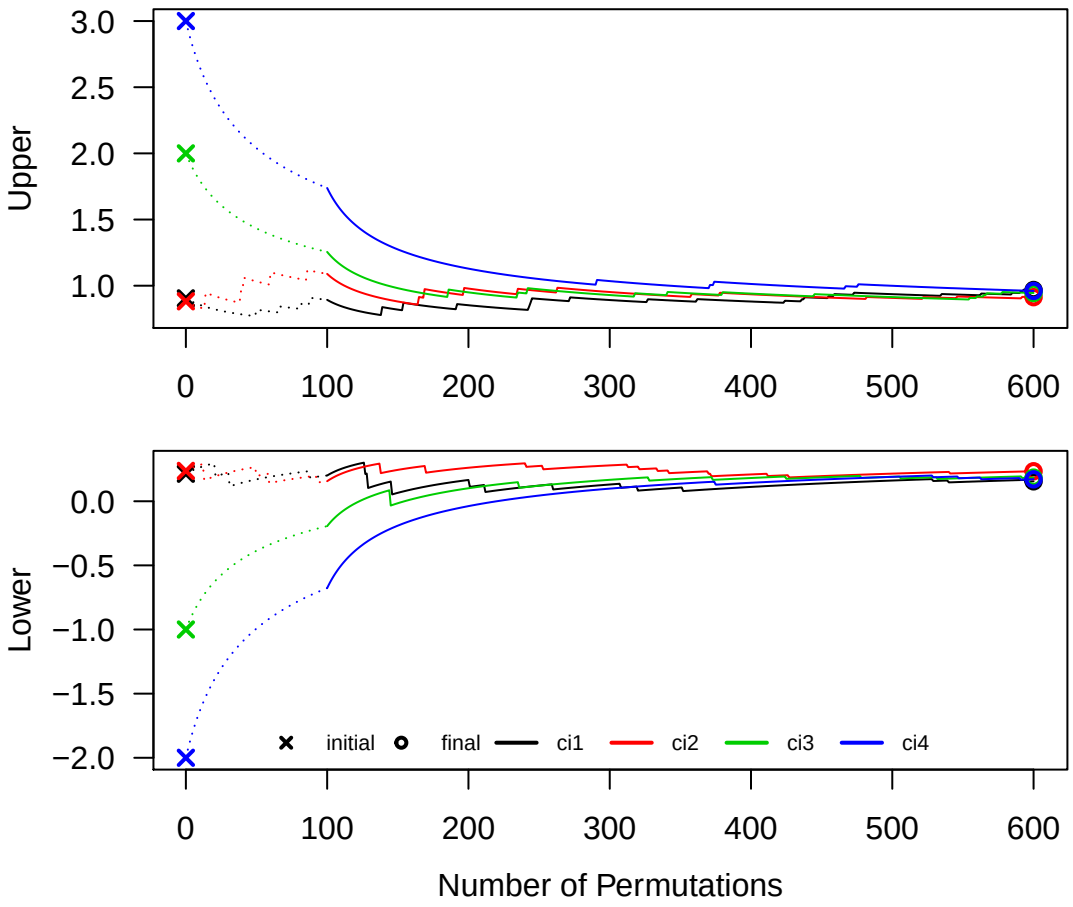
```
nperm <- 500
ci1 <- permci_glm(y ~ trt, trtname = 'trt', runit = 'clusid',
                  family = binomial, data = ds, nperm = nperm, ncores = 2,
                  seed = 448, initmethod = 'perm')
ci2 <- permci_glm(y ~ trt, trtname = 'trt', runit = 'clusid',
                  family = binomial, data = ds, nperm = nperm, ncores = 2,
                  seed = 449, initmethod = 'asymp')
ci3 <- permci_glm(y ~ trt, trtname = 'trt', runit = 'clusid',
                  family = binomial, data = ds, nperm = nperm, ncores = 2,
                  seed = 450, init = c(-1, 2))
ci4 <- permci_glm(y ~ trt, trtname = 'trt', runit = 'clusid',
                  family = binomial, data = ds, nperm = nperm, ncores = 2,
```

```
                seed = 451, init = c(-2, 3))
```



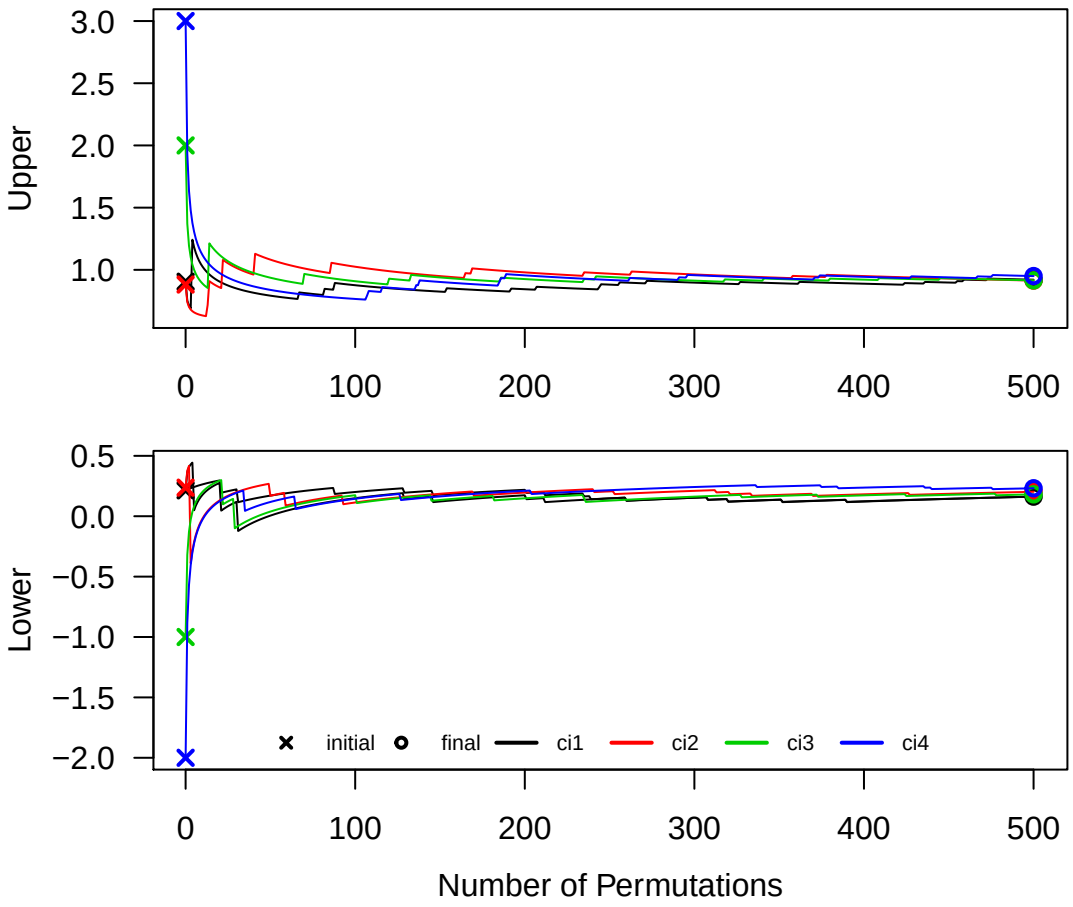
We see that `ci4` resulted in the widest CI, suggesting this search was sensitive to the wide starting values. Increasing the number of permutations may mitigate this sensitivity.

Another approach we could use to combat poor starting values is restarting the algorithm after a fixed number of permutations. Since the magnitude of the steps gets smaller as the search proceeds, it may take a large number of permutations to recover from bad starting values. Using a "burn-in" phase can speed up its recovery. Let's consider the same four sets of starting values as before, but now specify the additional argument `nburn = 100`.



Comparing back to the plot without the burn-in permutations, our final estimates are much less sensitive to the poor initial values of `ci3` and `ci4`. Even if we had fixed the total number of permutations at 500 and stopped after 400 post-burn-in permutations, we would have done better than before.

We can go even deeper by modifying the Robbins-Monro search proccess itself. For example, we can specify `permci_glm(..., m = 1)` to set the largest initial step magnitude (the magnitude is inversely proportional to `m`). This is yet another approach that may speed up convergence if we choose poor starting values, but could introduce problems if the first few steps happen to jump even further away from the true bounds. Below, we see improved convergence by using a larger initial step size.



For more details, see the `permci_glm()` documentation.

References

Dwass, M. (1957). Modified Randomization Tests for Nonparametric Hypotheses. *The Annals of Mathematical Statistics* **28**, 181–187.

Garthwaite, P. H. (1996). Confidence intervals from randomization tests. *Biometrics* **52**, 1387–1393.

Garthwaite, P. H. and Buckland, S. T. (1992). Generating Monte Carlo Confidence Intervals by the Robbins-Monro Process. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **41**, 159–171.

Goyal, R., Blitzstein, J., and De Gruttola, V. (2013). Simulating Bipartite Networks to Reflect Uncertainty in Local Network Properties. *Harvard University Biostatistics Working Paper Series*.

Hayes, R. J. and Moulton, L. H. (2017). *Cluster Randomised Trials* 2nd edition. New York: Chapman and Hall/CRC.

O'Brien, K. L. et al. (2003). Efficacy and safety of seven-valent conjugate pneumococcal vaccine in American Indian children: group randomised trial. *Lancet* **362**, 355–361.

Rabideau, D. J., and Wang, R. (2019). Randomization inference for a marginal treatment effect in cluster randomized trials. Manuscript sumbitted for publication.

Wang, R., Goyal, R., Lei, Q., Essex, M., and De Gruttola, V. (2014). Sample size considerations in the design of cluster randomized trials of combination HIV prevention. *Clinical Trials* **11**, 309–318.