I put up that purported to compare stepped-wedge study designs with more traditional cluster randomized trials. Either because I rushed or was just lazy, I didn't exactly do what I set out to do. I *did* confirm that a multi-site randomized clinical trial can be more efficient than a cluster randomized trial when there is variability across clusters. (I compared randomizing within a cluster with randomization by cluster.) But, this really had nothing to with stepped-wedge designs.

Randomize by or within Cluster

Under this design, 33 sites around the country will receive the training at some point, which is no small task (and fortunately as the statistician, this is a part of the study I have little involvement). After hearing about this ambitious plan, a colleague asked why we didn’t just randomize half the sites to the intervention and conduct a more standard cluster randomized trial, where a site would either get the training or not. I quickly simulated some data to see what we would give up (or gain) if we had decided to go that route. (It is actually a moot point, since there would be no way to simultaneously train 16 or so sites, which is why we opted for the stepped-wedge design in the first place.)

I simplified things a bit by comparing randomization *within* site with randomization *by* site. The stepped wedge design is essentially a within-site randomization, except that the two treatment arms are defined at different time points, and things are complicated a bit because there might be time by intervention confounding. But, I won’t deal with that here.

**Simulate data**

**library**(simstudy)

# define data

cvar <- iccRE(0.20, dist = "binary")

d <- defData(varname = "a", formula = 0, variance = cvar,

dist = "normal", id = "cid")

d <- defData(d, varname = "nper", formula = 100, dist = "nonrandom")

da <- defDataAdd(varname = "y", formula = "-1 + .4\*rx + a",

dist="binary", link = "logit")

**Randomize *within* cluster**

set.seed(11265)

dc <- genData(100, d)

di <- genCluster(dc, "cid", "nper", "id")

di <- trtAssign(di, strata = "cid", grpName = "rx")

di <- addColumns(da, di)

di

## id rx cid a nper y

## 1: 1 1 1 -0.4389391 100 1

## 2: 2 0 1 -0.4389391 100 0

## 3: 3 1 1 -0.4389391 100 0

## 4: 4 0 1 -0.4389391 100 0

## 5: 5 0 1 -0.4389391 100 1

## ---

## 9996: 9996 0 100 -1.5749783 100 0

## 9997: 9997 1 100 -1.5749783 100 0

## 9998: 9998 0 100 -1.5749783 100 0

## 9999: 9999 1 100 -1.5749783 100 0

## 10000: 10000 1 100 -1.5749783 100 0

I fit a **conditional** mixed effects model, and then manually calculate the conditional log odds from the data just to give a better sense of what the conditional effect is

**library**(lme4)

rndTidy(glmer(y ~ rx + (1 | cid), data = di, family = binomial))

## term estimate std.error statistic p.value group

## 1 (Intercept) -0.86 0.10 -8.51 0 fixed

## 2 rx 0.39 0.05 8.45 0 fixed

## 3 sd\_(Intercept).cid 0.95 NA NA NA cid

calc <- di[, .(estp = mean(y)), keyby = .(cid, rx)]

calc[, lo := log(odds(estp))]

calc[rx == 1, mean(lo)] - calc[rx == 0, mean(lo)]

## [1] 0.3985482

Next, I fit a **marginal** model and calculate the effect manually as well.

**library**(geepack)

rndTidy(geeglm(y ~ rx, data = di, id = cid, corstr = "exchangeable",

family = binomial))

## term estimate std.error statistic p.value

## 1 (Intercept) -0.74 0.09 67.09 0

## 2 rx 0.32 0.04 74.80 0

log(odds(di[rx==1, mean(y)])/odds(di[rx==0, mean(y)]))

## [1] 0.323471

**Randomize *by* cluster**

Next we repeat all of this, though randomization is at the cluster level.

dc <- genData(100, d)

dc <- trtAssign(dc, grpName = "rx")

di <- genCluster(dc, "cid", "nper", "id")

di <- addColumns(da, di)

di

## cid rx a nper id y

## 1: 1 0 0.8196365 100 1 0

## 2: 1 0 0.8196365 100 2 1

## 3: 1 0 0.8196365 100 3 0

## 4: 1 0 0.8196365 100 4 0

## 5: 1 0 0.8196365 100 5 0

## ---

## 9996: 100 1 -0.1812079 100 9996 1

## 9997: 100 1 -0.1812079 100 9997 0

## 9998: 100 1 -0.1812079 100 9998 0

## 9999: 100 1 -0.1812079 100 9999 1

## 10000: 100 1 -0.1812079 100 10000 0

Here is the conditional estimate of the effect:

rndTidy(glmer(y~rx + (1|cid), data = di, family = binomial))

## term estimate std.error statistic p.value group

## 1 (Intercept) -0.71 0.15 -4.69 0.00 fixed

## 2 rx 0.27 0.21 1.26 0.21 fixed

## 3 sd\_(Intercept).cid 1.04 NA NA NA cid

And here is the marginal estimate

rndTidy(geeglm(y ~ rx, data = di, id = cid, corstr = "exchangeable",

family = binomial))

## term estimate std.error statistic p.value

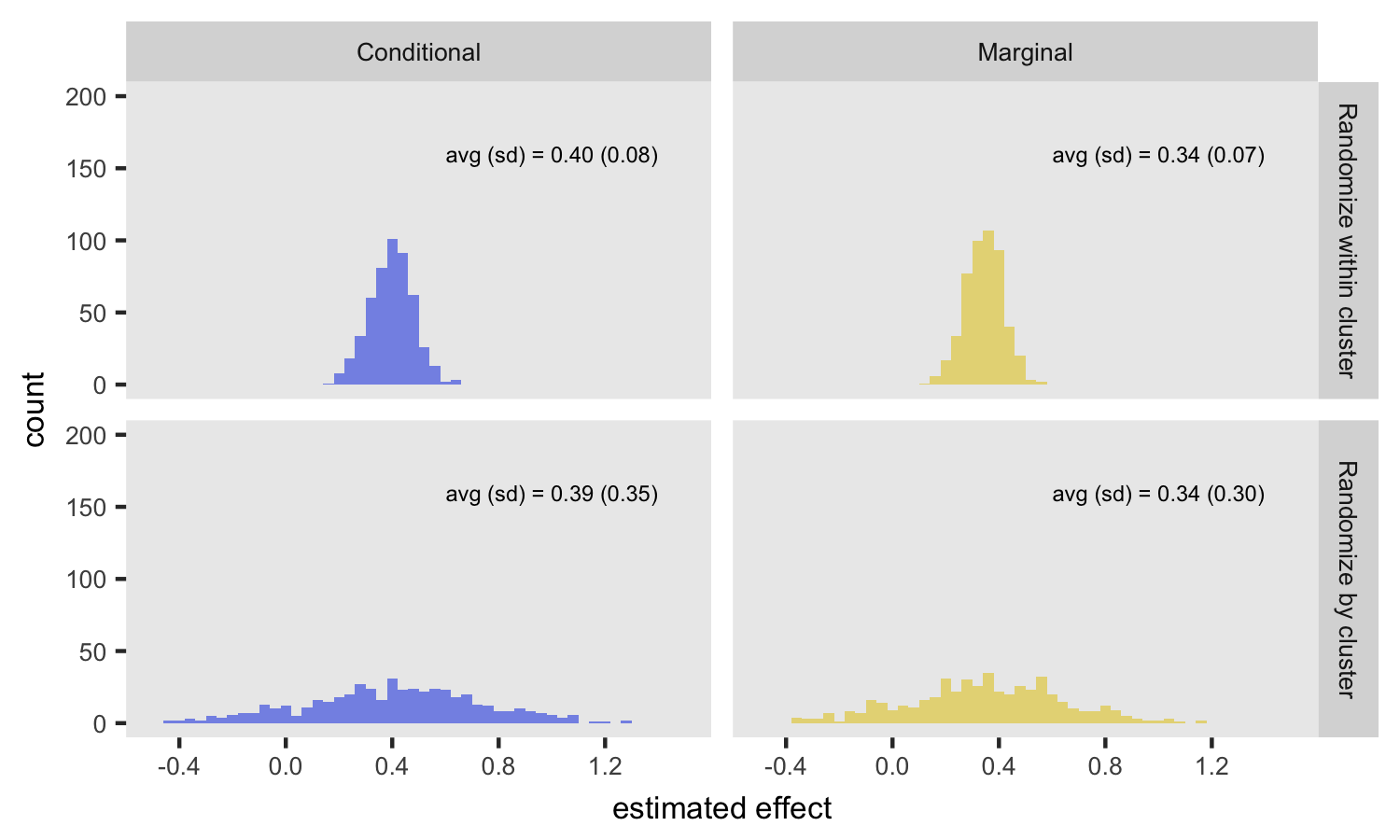
## 1 (Intercept) -0.56 0.13 18.99 0.00

## 2 rx 0.21 0.17 1.46 0.23

While the within- and by-site randomization estimates are quite different, we haven’t really learned anything, since those differences could have been due to chance. So, I created 500 data sets under different assumptions to see what the expected estimate would be as well as the variability of the estimate.

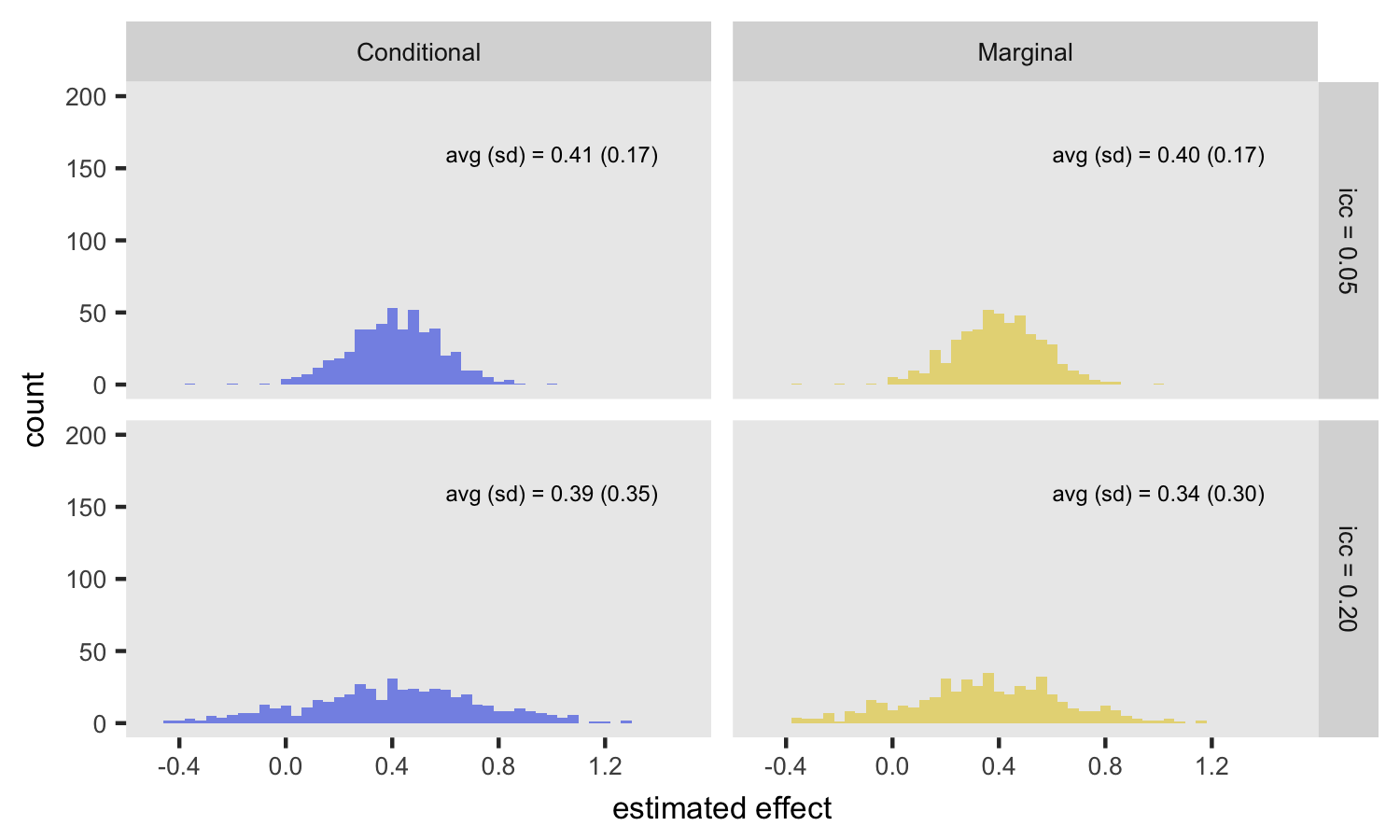
**Fixed ICC, varied randomization**

From this first set of simulations, the big take away is that randomizing *within* clusters provides an unbiased estimate of the conditional effect, but so does randomizing *by* site. The big disadvantage of randomizing *by* site is the added variability of the conditional estimate. The attenuation of the marginal effect estimates under both scenarios has nothing to do with randomization, but results from intrinsic variability across sites.



**Fixed randomization, varied ICC**

This next figure isolates the effect of across-site variability on the estimates. In this case, randomization is only *by* site (i.e. no within site randomization), but the ICC is set at 0.05 and 0.20. For the conditional model, the ICC has no impact on the expected value of the log-odds ratio, but when variability is higher (ICC = 0.20), the standard error of the estimate increases. For the marginal model, the ICC has an impact on *both* the expected value and standard error of the estimate. In the case with a low ICC (top row in plot), the marginal and condition estimates are quite similar.



Here, I will try to rectify the shortcomings of that post by actually simulating data from a traditional stepped-wedge design and two variations on that theme with the aim of seeing which approach might be preferable.

The key differences in the various designs are how many sites are exposed to the intervention and what the phase-in schedule looks like. In the examples that follow, I am assuming a study that lasts 24 weeks and with 50 total sites. Each site will include six patients per week. That means if we are collecting data for all sites over the entire study period, we will have \(24 \times 6 \times 50 = 7200\) outcome measurements.

The most important assumption I am making, however, is that the investigators can introduce the intervention at a small number of sites during each time period (for example, because the intervention involves extensive training and there is a limited number of trainers.) In this case, I am assuming that at most 10 sites can start the intervention at any point in time, and we must wait at least 4 weeks until the next wave can be started. (We can proceed slower than 4 weeks, of course, which surprisingly may be the best option.)

I am going to walk through the data generation process for each of the variations and then present the results of a series of power analyses to compare and contrast each design.

**Simulating from a Stepped Wedge Design**

## Simulating from a stepped wedge design

In this example, we are assuming a 3-year study with four groups of clusters randomized to start an intervention at either 12 months, 18 months, 24 months, or 30 months (i.e. every 6 months following the 1st baseline year). The study would enroll patients at baseline in each of the clusters, and a measurement of a binary outcome (say diabetes under control, or not) would be collected at that time. Those patients would be followed over time and the same measurement would be collected every 6 months, concluding with the 7th measurement in the 36th month of the study. (It is totally possible to enroll new patients as the study progresses and have a different follow-up scheme, but this approximates the actual study I was working on.)

The data are generated based on a mixed effects model where there are group level effects (b\_j*bj*​ in the model) as well as individual level effects (b\_i*bi*​). The model also assumes a very slight time trend before the intervention (e.g. diabetes control is improving slightly over time for an individual), an intervention effect, and an almost non-existent change in the time trend after the intervention. The outcome in each period is generated based on this formula:

logit(Y\_{ijt}) = 0.8 + .01 \* period + 0.8 \* I\_{jt} + 0.001 \* I\_{jt} \* (period-s\_j) + b\_i + b\_j,*logit*(*Yijt*​)=0.8+.01∗*period*+0.8∗*Ijt*​+0.001∗*Ijt*​∗(*period*−*sj*​)+*bi*​+*bj*​,

where period*period* goes from 0 to 6 (period 0 is the baseline, period 1 is the 6 month follow, etc.), I\_{jt}*Ijt*​ is 1 if cluster j*j* is in the intervention in period t*t*, s\_j*sj*​ is the period where the intervention starts for cluster j*j*, and logit(Y\_{ijt})*logit*(*Yijt*​) is the log odds of the outcome Y*Y* for individual i*i* in cluster j*j* during period t*t*.

We start by defining the data structure using simstudy “data def”" commands. We are assuming that there will be 100 individuals followed at each site for the full study. (We are not assuming any dropout, though we could easily do that.) In this particular case, we are assuming an effect size of 0.8 (which is a log odds ratio):

**library**(simstudy)

starts <- "rep(c(2 : 5), each = 10)"

siteDef <- defData(varname = "bj", dist = "normal", formula = 0,

variance = .01, id="site")

siteDef <- defData(siteDef, varname = "sj", dist = "nonrandom",

formula = starts)

siteDef <- defData(siteDef, varname = "ips", dist = "nonrandom",

formula = 100)

indDef <- defDataAdd(varname = "bi", dist = "normal", formula = 0,

variance = 0.01)

trtDef <- defDataAdd(varname = "Ijt" ,

formula = "as.numeric(period >= sj)",

dist = "nonrandom")

f = "0.8 + .01 \* period + 0.8 \* Ijt + 0.001 \* Ijt \* (period-sj) + bi + bj"

trtDef <- defDataAdd(trtDef, varname = "Yijt", formula = f,

dist = "binary", link = "logit")

To generate 40 clusters of data, we use the following code:

set.seed(6789)

dtSite <- genData(40, siteDef)

dtSite <- genCluster(dtSite, cLevelVar = "site", numIndsVar = "ips",

level1ID = "id")

dtSite <- addColumns(indDef, dtSite)

dtSiteTm <- addPeriods(dtSite, nPeriods = 7, idvars = "id")

dtSiteTm <- addColumns(trtDef, dtSiteTm)

dtSiteTm

## id period site bj sj ips bi timeID Ijt Yijt

## 1: 1 0 1 -0.1029785 2 100 0.08926153 1 0 1

## 2: 1 1 1 -0.1029785 2 100 0.08926153 2 0 1

## 3: 1 2 1 -0.1029785 2 100 0.08926153 3 1 1

## 4: 1 3 1 -0.1029785 2 100 0.08926153 4 1 1

## 5: 1 4 1 -0.1029785 2 100 0.08926153 5 1 1

## ---

## 27996: 4000 2 40 0.1000898 5 100 0.18869371 27996 0 1

## 27997: 4000 3 40 0.1000898 5 100 0.18869371 27997 0 0

## 27998: 4000 4 40 0.1000898 5 100 0.18869371 27998 0 1

## 27999: 4000 5 40 0.1000898 5 100 0.18869371 27999 1 1

## 28000: 4000 6 40 0.1000898 5 100 0.18869371 28000 1 1

And to visualize what the study data might looks like under these assumptions:

# summary by site

dt <- dtSiteTm[, .(Y = mean(Yijt)), keyby = .(site, period, Ijt, sj)]

ggplot(data = dt, aes(x=period, y=Y, group=site)) +

geom\_hline(yintercept = c(.7, .83), color = "grey99") +

geom\_line(aes(color=factor(site))) +

geom\_point(data = dt[sj == period], color="grey50") +

theme(panel.background = element\_rect(fill = "grey90"),

panel.grid = element\_blank(),

plot.title = element\_text(size = 10, hjust = 0),

panel.border = element\_rect(fill = NA, colour = "gray90"),

legend.position = "none",

axis.title.x = element\_blank()

) +

ylab("Proportion controlled") +

scale\_x\_continuous(breaks = seq(0, 10, by = 2),

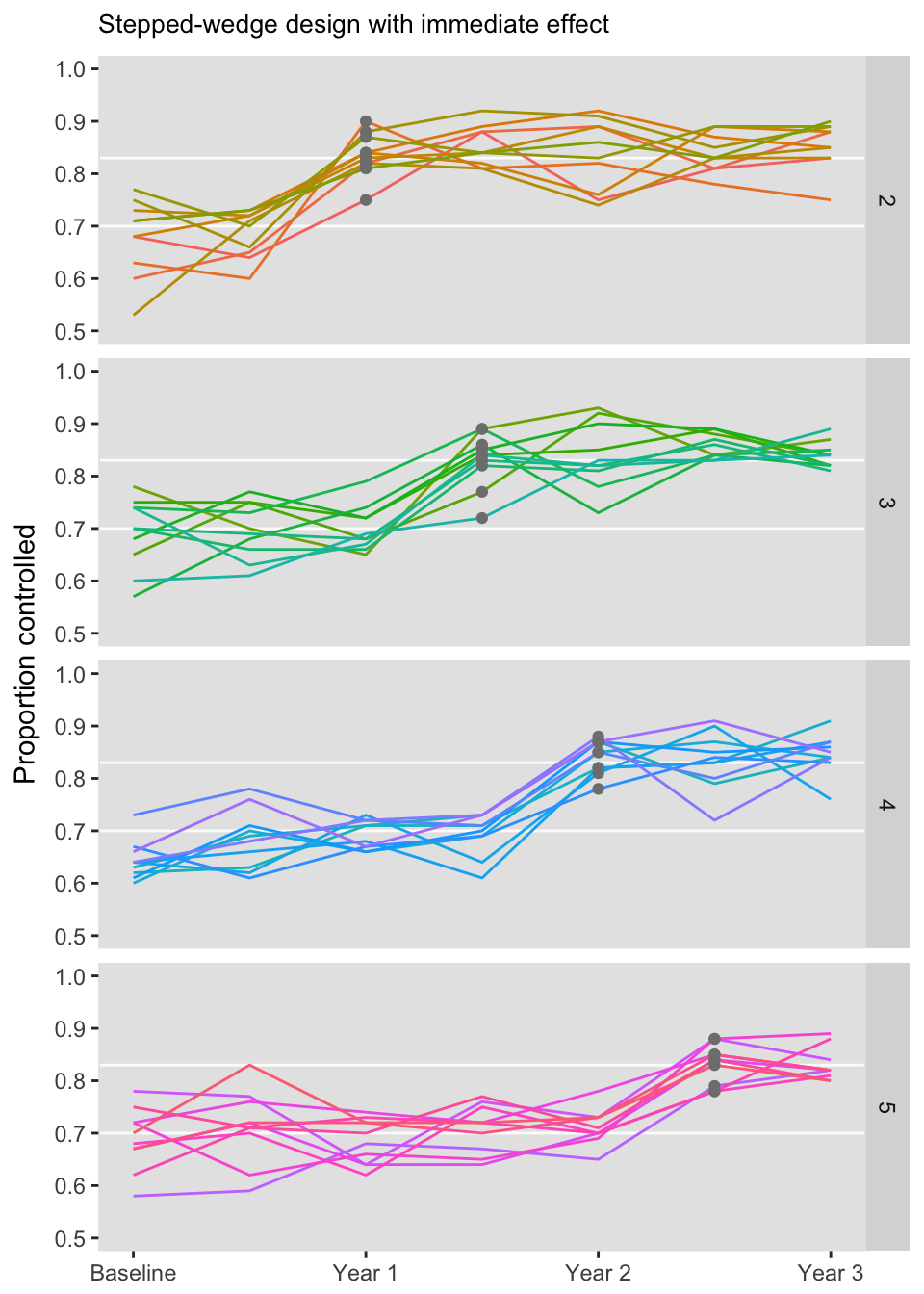
labels = c("Baseline", paste("Year", c(1:5)))) +

scale\_y\_continuous(limits = c(.5, 1),

breaks = c(.5, .6, .7, .8, .9, 1)) +

ggtitle("Stepped-wedge design with immediate effect") +

facet\_grid(sj~.)



## Estimating power

We are going to estimate power using only 20 clusters and effect size of 0.25. (Assuming 40 clusters and a large effect size was useful for visualizing the data, but not so interesting for illustrating power, since under those assumptions we are virtually guaranteed to find an effect.)

After generating the data (code not shown) for one iteration, we fit a generalized mixed effects model to show the effect estimate. In this case, the effect estimate is 1.46 (95% CI 1.21-1.77) on the odds ratio scale or 0.37 (95% CI 0.19-0.57) on the log odds ratio scale.

**library**(lme4)

**library**(sjPlot)

glmfit <- glmer(data = dtSiteTm,

Yijt ~ period + Ijt + I(Ijt\*(period - sj)) + (1|id) + (1|site),

family="binomial" )

sjt.glmer(glmfit, show.icc = FALSE, show.dev = FALSE)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Yijt | | |
|  |  | *Odds Ratio* | *CI* | *p* |
| **Fixed Parts** | | | | |
| (Intercept) |  | 2.15 | 1.90 – 2.44 | <.001 |
| period |  | 1.00 | 0.95 – 1.06 | .959 |
| Ijt |  | 1.46 | 1.21 – 1.77 | <.001 |
| I(Ijt \* (period - sj)) |  | 0.99 | 0.91 – 1.07 | .759 |
| **Random Parts** | | | | |
| τ00, id |  | 0.011 | | |
| τ00, site |  | 0.029 | | |
| Nid |  | 1000 | | |
| Nsite |  | 20 | | |
| Observations |  | 7000 | | |

In order to estimate power, we need to generate a large number of replications. I created a simple function that generates a new data set every iteration based on the definitions. If we want to vary the model assumptions across different replications, we can write code to modify the data definition part of the process. In this way we could look at power across different sample size, effect size, or variance assumptions. Here, I am only considering a single set of assumptions.

gData <- **function**() {

dtSite <- genData(nsites, siteDef)

dtSite <- genCluster(dtSite, cLevelVar = "site",

numIndsVar = "ips", level1ID = "id")

dtSite <- addColumns(indDef, dtSite)

dtSiteTm <- addPeriods(dtSite, nPeriods = 7, idvars = "id")

dtSiteTm <- addColumns(trtDef, dtSiteTm)

**return**(dtSiteTm)

}

And finally, we iterate through a series of replications, keeping track of each hypothesis test in the variable result. Typically, it would be nice to replicate a large number of times (say 1000), but this can sometimes take a long time. In this case, each call to glmer is very resource intensive - unfortunately, I know of know way to speed this up (please get in touch if you have thoughts on this) - so for the purposes of illustration, I’ve only used 99 iterations. Note also that I check to see if the model converges in each iteration, and only include results from valid estimates. This can be an issue with mixed effects models, particularly when sample sizes are small. To estimate the power (which in this case is 78%), calculate the proportion of successful iterations with a p-value smaller than 0.05, the alpha-level threshold we have used in our hypothesis test:

result <- NULL

i=1

**while** (i < 100) {

dtSite <- gData()

glmfit <- **tryCatch**(glmer(data = dtSite,

Yijt ~ period + Ijt + I(Ijt\*(period - sj)) + (1|id) + (1|site),

family="binomial" ),

**warning** = **function**(w) { "warning" }

)

**if** (! is.character(glmfit)) {

pvalue <- coef(summary(glmfit))["Ijt", "Pr(>|z|)"]

result <- c(result, pvalue)

i <- i + 1

}

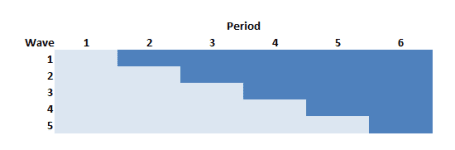
}

mean(result < .05)

## [1] 0.7812

To explore the sensitivity of the power estimates to changing underlying assumptions of effect size, sample size, variation, and time trends, we could vary those parameters and run a sequence of iterations. The code gets a little more complicated (essentially we need to change the “data defs” for each set of iterations), but it is still quite manageable. Of course, you might want to plan for fairly long execution times, particularly if you use 500 or 1000 iterations for each scenario, rather than the 100 I used here.

**Stepped-wedge design**



In the stepped-wedge design, all clusters in a trial will receive the intervention at some point, but the start of the intervention will be staggered. The amount of time in each state (control or intervention) will differ for each site (or group of sites if there are waves of more than one site starting up at the same time).

In this design (and in the others as well) time is divided into discrete data collection/phase-in periods. In the schematic figure, the light blue sections are periods during which the sites are in a control state, and the darker blue are periods during which the sites are in the intervention state. Each period in this case is 4 weeks long.

Here, the rollout period includes periods two through five.

First, we define the data, which will largely be the same across the designs: 6 individual patients per week, an intervention effect of 0.33, and a weekly time effect (which unfortunately is parameterized as "periodâ€�) of 0.02, and standard deviation within each cluster of 3.

library(simstudy)

defS <- defData(varname = "n", formula = 6,

dist = "nonrandom", id = "site")

defS <- defData(defS, varname = "siteInt", formula = 0,

variance = 1, dist = "normal")

defP <- defDataAdd(varname = "rx",

formula = "(start <= period) \* everTrt",

dist = "nonrandom")

defI <- defDataAdd(varname = "Y",

formula = "10 + rx \* 0.33 + period \* 0.02 + siteInt",

variance = 9, dist = "normal")

Now, we actually generate the data, starting with the site level data, then the period data, and then the individual patient level data. Note that the intervention is phased in every 4 weeks so that by the end of the 24 weeks all 5 waves are operating under the intervention:

set.seed(111)

dS <- genData(50, defS)

dS[, start := rep((1:5)\*4, each = 10)]

dS[, everTrt := 1]

dS[site %in% c(1, 2, 11, 12, 49, 50)] # review a subset

## site n siteInt start everTrt

## 1: 1 6 0.2352207 4 1

## 2: 2 6 -0.3307359 4 1

## 3: 11 6 -0.1736741 8 1

## 4: 12 6 -0.4065988 8 1

## 5: 49 6 2.4856616 20 1

## 6: 50 6 1.9599817 20 1

# weekly data

dP <- addPeriods(dtName = dS, nPeriods = 24, idvars = "site")

dP <- addColumns(defP, dP)

dP[site %in% c(3, 17) & period < 5] # review a subset

## site period n siteInt start everTrt timeID rx

## 1: 3 0 6 -0.31162382 4 1 49 0

## 2: 3 1 6 -0.31162382 4 1 50 0

## 3: 3 2 6 -0.31162382 4 1 51 0

## 4: 3 3 6 -0.31162382 4 1 52 0

## 5: 3 4 6 -0.31162382 4 1 53 1

## 6: 17 0 6 -0.08585101 8 1 385 0

## 7: 17 1 6 -0.08585101 8 1 386 0

## 8: 17 2 6 -0.08585101 8 1 387 0

## 9: 17 3 6 -0.08585101 8 1 388 0

## 10: 17 4 6 -0.08585101 8 1 389 0

# patient data

dI <- genCluster(dtClust = dP, cLevelVar = "timeID", numIndsVar = "n",

level1ID = "id")

dI <- addColumns(defI, dI)

dI

## site period n siteInt start everTrt timeID rx id Y

## 1: 1 0 6 0.2352207 4 1 1 0 1 10.810211

## 2: 1 0 6 0.2352207 4 1 1 0 2 14.892854

## 3: 1 0 6 0.2352207 4 1 1 0 3 12.977948

## 4: 1 0 6 0.2352207 4 1 1 0 4 11.311097

## 5: 1 0 6 0.2352207 4 1 1 0 5 10.760508

## ---

## 7196: 50 23 6 1.9599817 20 1 1200 1 7196 11.317432

## 7197: 50 23 6 1.9599817 20 1 1200 1 7197 7.909369

## 7198: 50 23 6 1.9599817 20 1 1200 1 7198 13.048293

## 7199: 50 23 6 1.9599817 20 1 1200 1 7199 17.625904

## 7200: 50 23 6 1.9599817 20 1 1200 1 7200 7.147883

Here is a plot of the site level averages at each time point:

library(ggplot2)

dSum <- dI[, .(Y = mean(Y)), keyby = .(site, period, rx, everTrt, start)]

ggplot(data = dSum, aes(x = period, y = Y, group = interaction(site, rx))) +

geom\_line(aes(color = factor(rx))) +

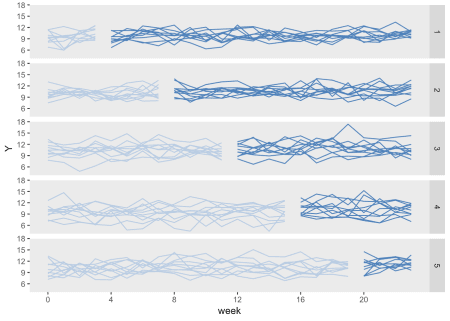
facet\_grid(factor(start, labels = c(1 : 5)) ~ .) +

scale\_x\_continuous(breaks = seq(0, 23, by = 4), name = "week") +

scale\_color\_manual(values = c("#b8cce4", "#4e81ba")) +

theme(panel.grid = element\_blank(),

legend.position = "none")



Finally, we can fit a linear mixed effects model to estimate the treatment effect:

library(lme4)

library(broom)

tidy(lmer(Y ~ rx + period + (1|site), data = dI))

## term estimate std.error statistic group

## 1 (Intercept) 9.78836231 0.184842722 52.955086 fixed

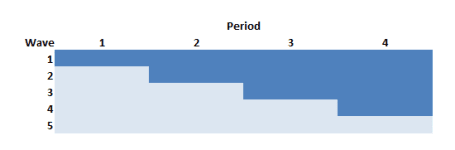
## 2 rx 0.35246094 0.122453829 2.878317 fixed

## 3 period 0.02110481 0.007845705 2.689983 fixed

## 4 sd\_(Intercept).site 1.21303055 NA NA site

## 5 sd\_Observation.Residual 2.99488532 NA NA Residual

**Stepped-wedge using "rolloutâ€� stage only**



The Thompson et al.Â paper argued that if we limit the study to the rollout period only (periods 2 through 5 in the example above) but increase the length of the periods (here, from 4 to 6 weeks), we can actually increase power. In this case, there will be one wave of 10 sites that never receives the intervention.

The data generation process is exactly the same as above, except the statement defining the length of periods (6 weeks instead of 4 weeks) and starting point (week 0 vs.Â week 4) is slightly changed:

dS[, start := rep((0:4)\*6, each = 10)]

So the site level data set with starting points at 0, 6, 12, and 18 weeks for each of the four waves that ever receive treatment looks like this:

## site n siteInt start everTrt

## 1: 1 6 0.2352207 0 1

## 2: 2 6 -0.3307359 0 1

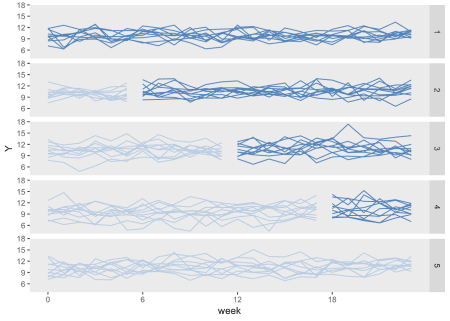
## 3: 11 6 -0.1736741 6 1

## 4: 12 6 -0.4065988 6 1

## 5: 49 6 2.4856616 24 1

## 6: 50 6 1.9599817 24 1

And the data generated under this scenario looks like:



Here is the model estimation:

tidy(lmer(Y ~ rx + period + (1|site), data = dI))

## term estimate std.error statistic group

## 1 (Intercept) 9.79022407 0.185294936 52.835897 fixed

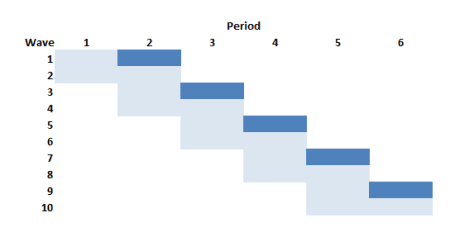
## 2 rx 0.30707559 0.122414620 2.508488 fixed

## 3 period 0.02291619 0.006378367 3.592800 fixed

## 4 sd\_(Intercept).site 1.21153700 NA NA site

## 5 sd\_Observation.Residual 2.99490926 NA NA Residual

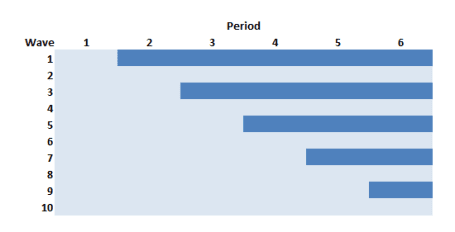
**Staggered cluster randomized trial**



If we wanted to conduct a cluster randomized trial but were able to phase in the intervention over time as we have been assuming, this design is the closest we could get. In this example with 50 sites and five phase-in periods, the intervention waves (in this example 1, 3, 5, 7, and 9) would each include five clusters. The respective control waves (2, 4, 6, 8, and 10) would also have five clusters each. And since we are assuming five waves, each wave will be in the study for eight: the first four weeks comprise "preâ€� measurement period, and the second four week period is the "postâ€� measurement period.

The problem with this design relative to all the others discussed here is that the amount of data collected for each site is considerably reduced. As a result, this design is going to be much less efficient (hence less powerful) than the others. So much so, that I do not even generate data for this design (though I did actually confirm using simulations not shown here.)

**Staggered cluster randomized trial with continued measurement**

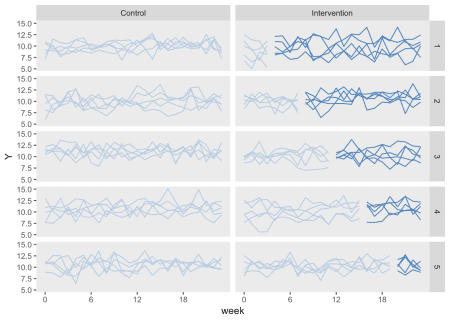


This is the staggered CRT just described, but we collect data for all 24 weeks for all of the sites. In this case, we are not at disadvantage with respect to the number of measurements, so it might be a competitive design. This version of staggered CRT could also be viewed as a traditional stepped-wedge design with controls.

The data generation is identical to the traditional stepped-wedge design we started with, except the only half of the sites are "ever treatedâ€�:

dS[, everTrt := rep(0:1)]

Here is the plot, with the control arm on the left, and the intervention arm on the right. The control arm is never introduced to the intervention.



**Conducting a power analysis using simulation**

We are ultimately interested in assessing how much information each study design can provide. Power analyses under different conditions are one way to measure this.

Since one of my missions here is to illustrate as much R code as possible, here is how I do conduct the power analysis of the traditional stepped-wedge design:

powerStepWedge1 <- function(x) {

# generate data

dS <- genData(50, defS)

dS[, start := rep((1:5)\*4, each = 10)]

dS[, everTrt := 1]

dP <- addPeriods(dtName = dS, nPeriods = 24, idvars = "site")

dP <- addColumns(defP, dP)

dI <- genCluster(dtClust = dP, cLevelVar = "timeID",

numIndsVar = "n", level1ID = "id")

dI <- addColumns(defI, dI)

# fit model

data.frame(summary(lmer(Y ~ rx + period + (1|site), data = dI))$coef)

}

res <- vector("list", length = 5)

i <- 0

for (icc in seq(0, 0.04, .01)) {

i <- i + 1

# update data definition based on new ICC

between.var <- iccRE(ICC = icc, dist = "normal", varWithin = 9)

defS <- updateDef(defS, changevar = "siteInt", newvariance = between.var)

# generate 200 data sets and fit models

resSW1<- lapply(1:200, FUN = powerStepWedge1)

# estimate and store power

pSW1 <- mean( unlist(lapply(resSW1, `[`, 2, 3 )) >= 1.96)

res[[i]] <- data.table(icc, pSW1)

}

rbindlist(res)

## icc pSW1

## 1: 0.00 0.940

## 2: 0.01 0.855

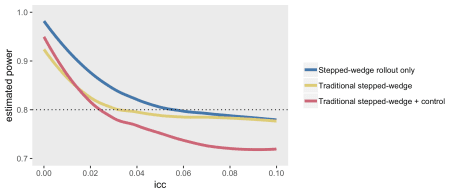
## 3: 0.02 0.850

## 4: 0.03 0.830

## 5: 0.04 0.780

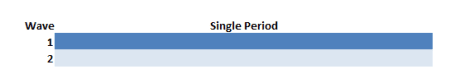
**Comparing power of three different designs**

The next figure shows the estimated power for all three designs based on the same effect size and a range of ICC's. The SW rollout only design consistently equals or outperforms the others. When the ICC is moderate to large (in this case > 0.06), the traditional SW design performs equally well. The design that comes closest to a staggered cluster randomized trial, the SW + controls performs well here on the lower range of ICCs, but is less compelling with more between site variation.

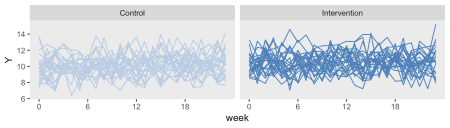


All of this has been done in the context of a normally distributed outcome. At some point, I will certainly re-do this comparison with a binary outcome.

**Addendum: cluster randomized trial**



A traditional cluster randomized trial was not really under consideration because we declared that we could only deliver the intervention to 10 sites at any one time. However, it is illustrative to compare this design to make it clear that CRT is really best used when variability across sites is at its lowest (i.e.Â when the ICC is at or very close to zero). In this example, 25 sites are randomized to receive the intervention starting in the first week and 25 sites never receive the intervention. Data are collected for all 24 weeks for each of the 50 clusters.



The simulations confirm findings that the CRT is more efficient than stepped-wedge designs when the ICC is close to zero, but pales in comparison even with ICCs as low as 0.01:

