Simulation code for 5 cases

2023-5-23

Case 1: Two Continuous Mean

```r
################################
## Sample size; 2-sample design, Z-approximation, compare continuous means
################################
#formula from the textbook
samp_size_2mean <- function(alpha, beta, q1, q0, Effectsize, sigma) {
  A <- 1/q1 + 1/q0
  B <- (qnorm(alpha/2)+qnorm(beta))^2
  Number <- A*B/((Effectsize/sigma)^2)
  return(Number)
}
#test the data out
samp_size_2mean(alpha=0.05, beta=0.2, q1=0.5, q0=0.5, Effectsize=0.2, sigma=1)
```

```
## [1] 784.888
```

```r
#######################################
## Same thing via simulation
######################################
#for per group we need to divide to 2
samp_size <- ceiling(samp_size_2mean(alpha=0.05, beta=0.2, q1=0.5, q0=0.5, Effectsize=0.2, sigma=1)/2)

#build a function with all the variable needed for the function
samp_size_sim_ttest <- function(n, m1, m2, s) {
    ## Simulate data from random normal data generating mechanism
  x1 <- rnorm(n, mean=m1, sd=s)
    x2 <- rnorm(n, mean=m2, sd=s)
    ## Analyze independent random normal sample using t-test
    t_test <- t.test(x=x1, y=x2)
    t_test_pval <- t_test$p.value
    t_test_tag <- ifelse(t_test_pval<0.05, 1, 0)
    return(t_test_tag)
}
#try with dummy data
samp_size_sim_ttest(n=10, m1=0, m2=0.2, s=1)
```

```
## [1] 0
```

```
#t test for testing the simulation result
t_test_sim_vec <- replicate(10000, samp_size_sim_ttest(n=samp_size, m1=0, m2=0.2, s=1))

mean(t_test_sim_vec)
```

```
## [1] 0.7971
```

Case 2: Two Binary Proportions

```
################################
## Sample size; 2-sample design, Z-approximation, compare binary proportions
################################
samp_size_2props <- function(alpha, beta, q1, q0, P1, P0){
  P <-(q1*P1) + (q0*P0)
  A <- qnorm(alpha/2)*sqrt( P*(1-P)*(1/q1 + 1/q0) )
  B <- qnorm(beta)*sqrt( P1*(1-P1)*(1/q1) + P0*(1-P0)*(1/q0) )
  C <- (P1-P0)^2
  Number <- (A+B)^2/C
  return(Number)
}
samp_size_2props(alpha=0.05, beta=0.2, q1=0.5, q0=0.5, P1=0.3, P0=0.2) #586.3
```

```
## [1] 586.3026
```

```
########################################
## Same thing via simulation
#######################################
samp_size_bi_pro <- floor(samp_size_2props(alpha=0.05, beta=0.2, q1=0.5, q0=0.5, P1=0.3, P0=0.2)/2) + 20
samp_size_bi_pro
```

```
## [1] 313
```

```
samp_size_sim_proptest <- function(n, prob1, prob2) {
    ## Simulate data from random normal data generating mechanism with fisher test
  x1 <- rbinom(n, size=1, prob = prob1)
    x2 <- rbinom(n, size=1, prob = prob2)
    g1 <- rep("group1", length(x1))
    g2 <- rep("group2", length(x2))
    x <- c(x1, x2)
    g <- c(g1, g2)
    res <- fisher.test(table(x,g))
    # prop test for proportion mean
    prop_test_pval <- res$p.value
    prop_test_tag <- ifelse(prop_test_pval<0.05, 1, 0)
    return(prop_test_tag)
}

#t test for testing the simulation result
prop_test_sim_vec <- replicate(1000, samp_size_sim_proptest(n=samp_size_bi_pro, prob1=0.3, prob2=0.2))
mean(prop_test_sim_vec)
```

```
## [1] 0.798
```

```
##alternative way

samp_size_sim_proptest <- function(n, prob1, prob2) {
    ## Simulate data from random normal data generating mechanism wiht sum function
  x1 <- rbinom(n, size=1, prob = prob1)
    x2 <- rbinom(n, size=1, prob = prob2)
    n1 <- length(x1)
    n2 <- length(x2)
    sx1 <- sum(x1)
    sx2 <- sum(x2)
    res <- prop.test(x=c(sx1, sx2), n=c(n1,n2), alternative="two.sided", correct=TRUE)
    prop_test_pval <- res$p.value
    prop_test_tag <- ifelse(prop_test_pval<0.05, 1, 0)
    return(prop_test_tag)
}

prop_test_sim_vec <- replicate(1000, samp_size_sim_proptest(n=samp_size_bi_pro, prob1=0.3, prob2=0.2))
mean(prop_test_sim_vec)
```

```
## [1] 0.806
```

Case 3: Correlation Case

```
##############################
## Sample size; 1-sample design, Z-approximation, single correlation coefficient
##############################
library(MASS)
samp_size_corr_coef <- function(alpha, beta, correlation){
  C <- 0.5*log( (1+correlation)/(1-correlation) )
  Number <- ( (qnorm(alpha/2) + qnorm(beta)) / C)^2 + 3
  return(Number)
}
samp_size_corr_coef(alpha=0.05,beta=0.1,correlation=0.3) #112
```

```
## [1] 112.6781
```

```
#######################################
## Same thing via simulation
#####################################
samp_size_cor <- ceiling(samp_size_corr_coef(alpha=0.05,beta=0.1,correlation=0.3))
samp_size_cor
```

```
## [1] 113
```

```
# MASS package provide the Multidimensional vector data
samp_size_sim_cor_test <- function(n, rho) {
  ## Mean vector for MVN data
```

```r
  mu_vec <- c(0,0)
  ## Sigma matrix for MVN data
  Sigma <- matrix(c(1,rho,rho,1), ncol=2, byrow=TRUE)
  ## Simulate MVN data with given mean/covariance structure
  x <- data.frame(mvrnorm(n=n, mu=mu_vec, Sigma=Sigma))
  ## Rename columns using perason test for determined
  names(x) <- c("x1","x2")
  res <- cor.test(x=x$x1, y=x$x2, method="pearson")
  cor_test_pval <- res$p.value
  cor_test_tag <- ifelse(cor_test_pval<0.05, 1, 0)
  return(cor_test_tag)
}
# find the simulate result
mean(replicate(1000, samp_size_sim_cor_test(n=samp_size_cor, rho=0.3)))
```

```
## [1] 0.898
```

```r
#alternative way
samp_size_cor <- ceiling(samp_size_corr_coef(alpha=0.05,beta=0.1,correlation=0.3))
samp_size_cor
```

```
## [1] 113
```

```r
samp_size_sim_cor_test <- function(n, rho) {
  ## Mean vector for MVN data
  mu_vec <- c(0,0)
  ## Sigma matrix for MVN data
  Sigma <- matrix(c(1,rho,rho,1), ncol=2, byrow=TRUE)
  ## Simulate MVN data with given mean/covariance structure
  x <- data.frame(mvrnorm(n=n, mu=mu_vec, Sigma=Sigma))
  ## Rename columns
  names(x) <- c("x1","x2")
  lm_fit <- lm(x1 ~ x2, data=x)
  lm_fit_coef <- coef(summary(lm_fit))
  lm_fit_pval <- lm_fit_coef[2,4]
  lm_fit_tag <- ifelse(lm_fit_pval<0.05, 1, 0)
  return(lm_fit_tag)
}
#test for simulation
mean(replicate(1000, samp_size_sim_cor_test(n=samp_size_cor, rho=0.3)))
```

```
## [1] 0.902
```

```r
## make the data frame find mean power
n <- 113
mu_vec <- c(0,0)
rho <- 0.3
Sigma <- matrix(c(1,rho,rho,1), ncol=2, byrow=TRUE)

dat <- expand.grid(n,mu_vec,Sigma )
names(dat) <- c("n","mu_vec","Sigma")
```

```
power_vec <- apply(dat, 1, function(x)mean(replicate(1000, samp_size_sim_cor_test(n=samp_size_cor, rho=

dat$power <- power_vec
dat
```

```
##     n mu_vec Sigma power
## 1 113      0   1.0 0.903
## 2 113      0   1.0 0.915
## 3 113      0   0.3 0.914
## 4 113      0   0.3 0.888
## 5 113      0   0.3 0.895
## 6 113      0   0.3 0.910
## 7 113      0   1.0 0.908
## 8 113      0   1.0 0.912
```

```
mean(power_vec)
```

```
## [1] 0.905625
```

Case 4: One Continueous mean

```
###############################
## Sample size; 1-sample design, Z-approximation, single continuous mean
###############################
samp_size_cont_ci <- function(alpha, width, sigma){
  Number <- 4*qnorm(alpha/2)^2*sigma^2/width^2
  return(Number)
}
samp_size_cont_ci(alpha=0.05, width=0.6, sigma=1) # 42.68288
```

```
## [1] 42.68288
```

```
#############################################
## Simulate sample size required to estimate single continuous mean
#############################################
n <- 43

## Set Seed to ensure replicability of random number generation
set.seed(12345)

#create the function
norm_ci_samp_size <- function(n, mean, sd) {
    ## Generate random data
    x <- rnorm(n=n, mean = mean, sd = sd)

    ## Analyze generated/simulated data
    res=t.test(x, mu = mean, alternative = "two.sided")

    ## Extract the estimate and CI
    p_hat <- res$estimate
    p_ll <- res$conf.int[1]
    p_ul <- res$conf.int[2]

    # Return the estimate and the CI to the user
    out <- c(p_hat=p_hat, p_ll=p_ll, p_ul=p_ul)
    return(out)

}


## Simulate n_rep copies of sample size trials
sim_out <- list()

## Number simulation replicates
n_rep <- 10000
mean <- 1
sd <- 1


## Loop over number simulation replicates, storing results in list
t0 <- Sys.time()
for (i in 1:n_rep) {
    sim_out[[i]] <- norm_ci_samp_size(n =n, mean = mean, sd= sd)
}
t1 <- Sys.time()
runtime <- t1-t0
runtime
```

```
## Time difference of 1.192147 secs
```

```
## Aggregate results into dataframe
sim_df <- do.call("rbind", sim_out)
sim_means <- apply(sim_df, 2, mean)
sim_means
```

```
## p_hat.mean of x            p_ll            p_ul
##       0.9999889       0.6935878       1.3063900
```

```
#predetermined CI given the sample mean and sd
ci_width <- sim_means[[3]] - sim_means[[2]]
ci_width
```

```
## [1] 0.6128021
```

Case 5: One Binary Proportion

```
################################
## Sample size; 1-sample design, Z-approximation, single binary proportion
```

```
################################
samp_size_bin_ci <- function(alpha, proportion, width){
  Number <- (4 * qnorm(alpha/2)^2 * proportion*(1-proportion) ) / (width^2)
  return(Number)
}
samp_size_bin_ci(alpha=0.05, proportion=0.2, width=0.1) #245
```

```
## [1] 245.8534
```

```
##############################################
## Simulate sample size required to estimate binomial CI with certain precision
##############################################

## Set Seed to ensure replicability of random number generation
set.seed(12345)

##
## Create function to estimate sample size
##
binom_ci_samp_size <- function(n_, p_) {
    ## Generate random data
    x <- rbinom(n=n_, size=1, p=p_)

    ## Analyze generated/simulated data
    res <- prop.test(table(x)[2:1])
    # str(res)

    ## Extract the estimate and CI
    p_hat <- res$estimate
    p_ll <- res$conf.int[1]
    p_ul <- res$conf.int[2]

    # Return the estimate and the CI to the user
    out <- c(p_hat=p_hat, p_ll=p_ll, p_ul=p_ul)
    return(out)

}


## Simulate n_rep copies of sample size trials
sim_out <- list()

## Number simulation replicates
n_rep <- 1000
samp_size <- 246
true_prop <- 0.8

## Loop over number simulation replicates, storing results in list
t0 <- Sys.time()
for (i in 1:n_rep) {
    sim_out[[i]] <- binom_ci_samp_size(n_=samp_size, p_=true_prop)
}
t1 <- Sys.time()
```

```r
runtime <- t1-t0
runtime
```

```
## Time difference of 0.2899079 secs
```

```r
## Aggregate results into dataframe
sim_df <- do.call("rbind", sim_out)
sim_means <- apply(sim_df, 2, mean)
sim_means
```

```
##   p_hat.p      p_ll      p_ul
## 0.8001423 0.7436927 0.8469878
```

```r
#predetermined CI given the sample proportion and alpha
ci_width <- sim_means[[3]] - sim_means[[2]]
ci_width
```

```
## [1] 0.1032951
```

```r
###################
## Compare against analytic formula
##################
binom_ci <- function(za, p, w) {(4*(za^2)*p*(1-p))/(w^2)}
binom_ci(za=1.96, p=0.8, w=.1)
```

```
## [1] 245.8624
```