Pool vs Individual

Guanshengrui Hao

12/4/2018

Research question

We compare two strategies of conducting multiple hypothesis testing. The first strategy is conducting hypothesis testing for each single site to obtain p-values. Then make decision by adjusting p-values to control for FDR across all sites. We will refer this strategy as individual strategy hereafter. The second strategy is to form a reference distribution first using the test statistics of each site under all (or a random sample when the number of samples in each group is large) possible permutation of group labels. Then evaluate the test statistic under original grouping based on the reference distribution to obtain p-values and adjust to control for FDR. We will refer this strategy as pooled strategy hereafter.

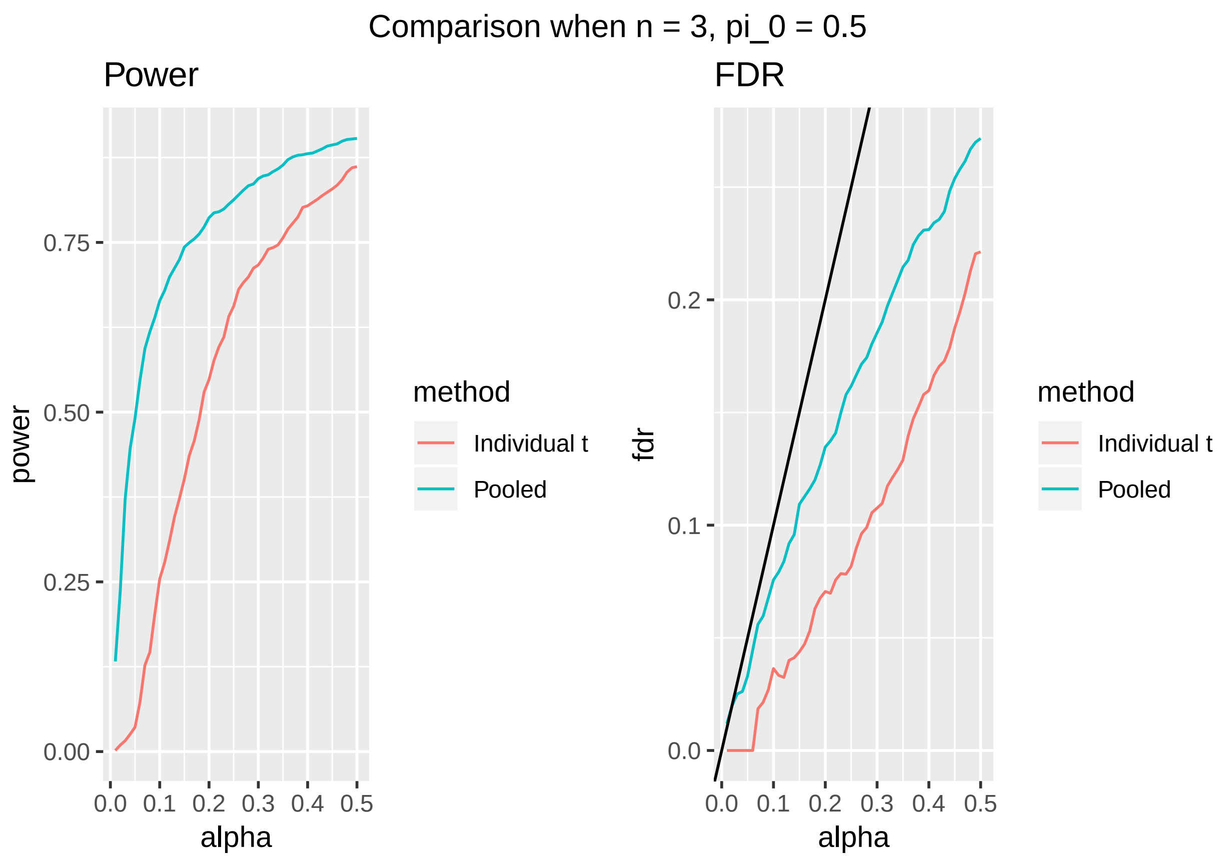
Comparison under homogeneous normal setup

In this comparison, all samples under true null are drawn from $N(0, 1)$ distribution. Under true non-null, samples from control group are drawn from $N(0, 1)$ while samples from treatment group are drawn from $N(\mu, \sigma^2)$, where $\mu \sim \text{Unif}(1, 4)$, $\sigma \sim \text{Exp}(1)$.

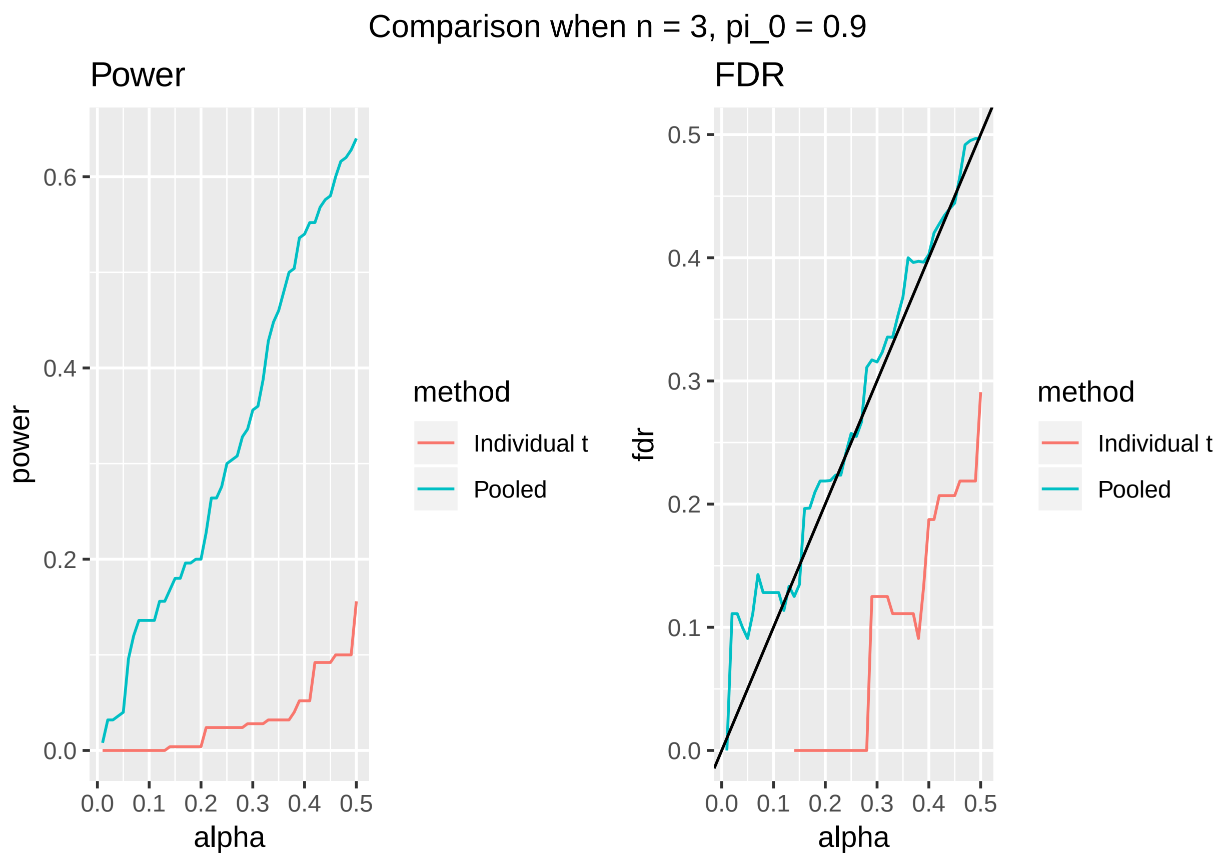
```r
# Total number of tests
m <- 2500
# number of samples in each group
n_vec <- c(3, 5)
# Proportion of null
pi_0_vec <- c(0.5, 0.9)

for (n in n_vec) {
  for (pi_0 in pi_0_vec) {
    #----------Simulation setup----------#
    # Index of true null
    true_null_idx <- 1:(m*pi_0)
    non_null_idx <- (1:m)[-true_null_idx]

    # Mean of each test in control group and treatment group
    mu_control <- numeric(m)
    mu_treat <- numeric(m)

    mu_treat[true_null_idx] <- mu_control[true_null_idx]
    mu_treat[non_null_idx] <- runif(length(non_null_idx), 1, 4)

    # Standard deviation of each test in both groups
    sigma_control <- rep(1, m)
    sigma_treat <- sigma_control
    sigma_treat[non_null_idx] <- rexp(length(non_null_idx), rate = 1)

    # Generate data
    control <- replicate(n, rnorm(m, mu_control, sigma_control))
    treat <- replicate(n, rnorm(m, mu_treat, sigma_treat))
```

```r
dat <- cbind(control, treat)

#----------Conduct multiple testing----------#
perms <- combn(2*n, n)
perms <- perms[, 1:(ncol(perms)/2)]
stat <- matrix(0, nrow = m, ncol = ncol(perms))

for (i in 1:ncol(perms)) {
  perm <- perms[, i]
  control <- dat[, perm]
  treat <- dat[, -perm]

  stat[, i] <- abs(unlist(lapply(1:nrow(treat),
                              FUN = function(x, treat, control)
                                t.test(treat[x, ], control[x, ])$statistic,
                              treat = treat,
                              control = control)))
}

alpha <- seq(0.01, 0.5, by = 0.01)
#------------------Individual t-test------------------#
p_val_t <- apply(dat,
                 1,
                 FUN = function(x)
                   t.test(x[1:n], x[n + (1:n)])$p.value)
q_val_t <- p.adjust(p_val_t, method = "BH")
power_t <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                         q_val = q_val_t,
                         non_null_idx = non_null_idx))
fdr_t <- unlist(lapply(alpha,
                       FUN = function(x, q_val, non_null_idx)
                         sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                       q_val = q_val_t,
                       non_null_idx = non_null_idx))

#------------------Pooled permutation test------------------#
ref_cdf <- ecdf(as.vector(stat[, 2:ncol(stat)]))
p_val_pool <- 1 - ref_cdf(stat[, 1])
q_val_pool <- p.adjust(p_val_pool, method = "BH")
# pi_0_hat <- sum(p_val_pool > 0.5)/m/(0.5)
# q_val_pool_storey <- unlist(lapply(p_val_pool,
#                                    FUN = function(x, pi_0, p_val)
#                                      m*pi_0*x/max(sum(p_val <= x), 1),
#                                    pi_0 = pi_0_hat,
#                                    p_val = p_val_pool))

power_pool <- unlist(lapply(alpha,
                     FUN = function(x, q_val, non_null_idx)
                       sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                     q_val = q_val_pool,
                     non_null_idx = non_null_idx))
```

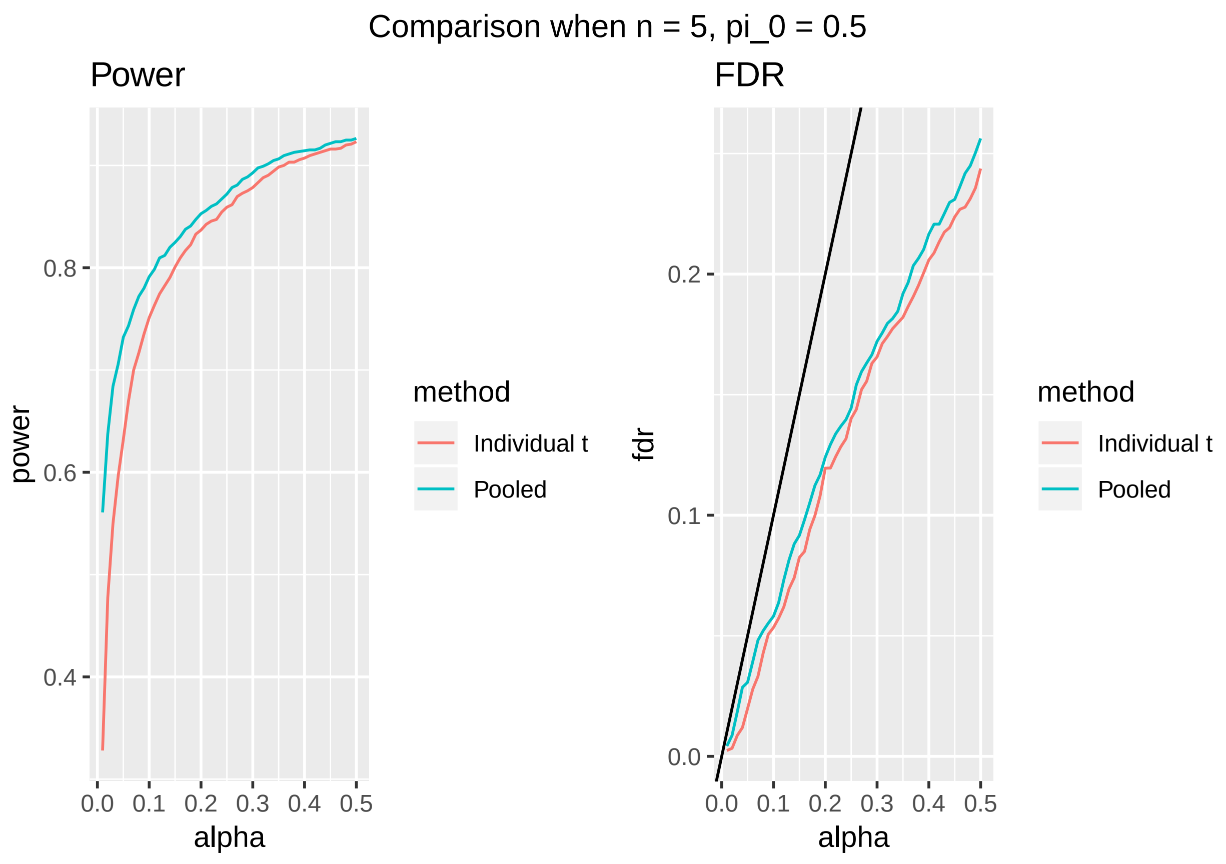
```
    fdr_pool <- unlist(lapply(alpha,
                          FUN = function(x, q_val, non_null_idx)
                            sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                          q_val = q_val_pool,
                          non_null_idx = non_null_idx))

    # power_pool_storey <- unlist(lapply(alpha,
    #                               FUN = function(x, q_val, non_null_idx)
    #                                 sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
    #                               q_val = q_val_pool_storey,
    #                               non_null_idx = non_null_idx))
    # fdr_pool_storey <- unlist(lapply(alpha,
    #                             FUN = function(x, q_val, non_null_idx)
    #                               sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
    #                             q_val = q_val_pool_storey,
    #                             non_null_idx = non_null_idx))

    #----------Plot----------#
    dat_plot <- cbind.data.frame(c(power_pool, power_t),
                                 c(fdr_pool, fdr_t),
                                 c(alpha, alpha),
                                 c(rep("Pooled", length(alpha)), rep("Individual t", length(alpha))))
    colnames(dat_plot) <- c("power", "fdr", "alpha", "method")
    power_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = power, color = method)) + ggti
    fdr_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = fdr, color = method)) + geom_abl
    grid.arrange(power_plot, fdr_plot, nrow = 1, top = paste0("Comparison when n = ", n, ", pi_0 = ", p
    print(paste0("The step for n = ", n, ", pi_0 = ", pi_0, " has completed!"))
  }
}
```



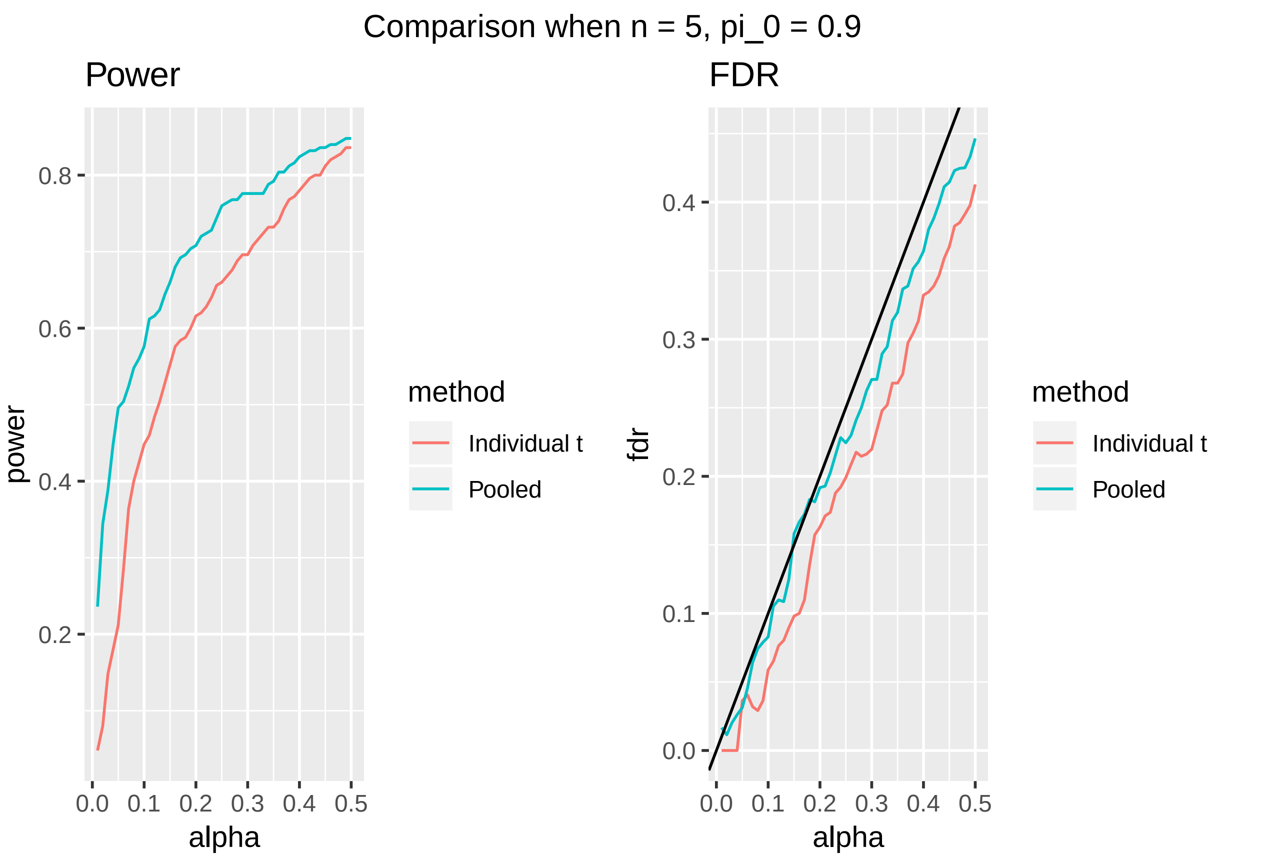
```
## [1] "The step for n = 3, pi_0 = 0.5 has completed!"

## Warning: Removed 13 rows containing missing values (geom_path).
```



```
## [1] "The step for n = 3, pi_0 = 0.9 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.9 has completed!"
```

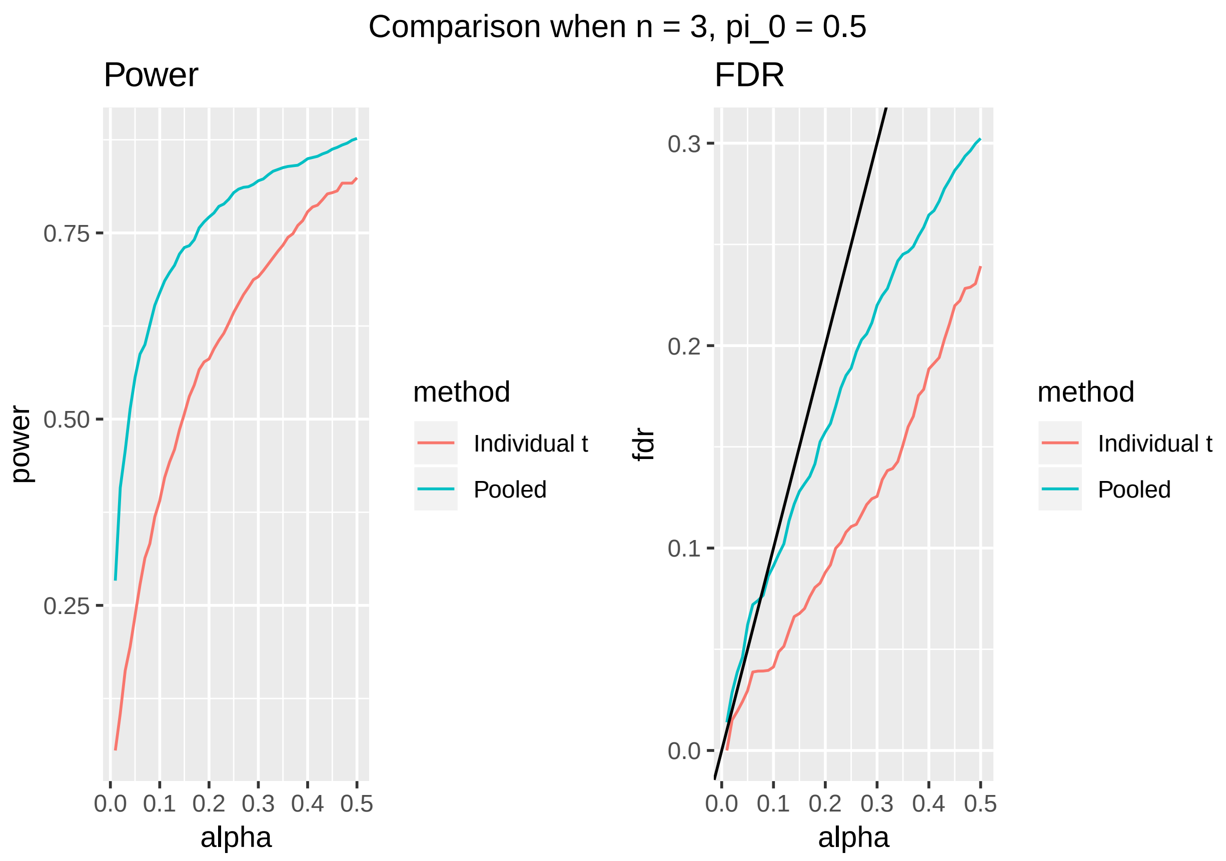
Comparison under heterogeneous normal setup

In this comparison, all samples from control group are drawn from $N(\mu_i^c, \sigma_i^c)$, where μ_i^c's are drawn from $N(0, 5)$ and σ_i^c's are drawn from $\text{Exp}(1)$. All samples from treatment group are drawn from $N(\mu_i^t, \sigma_i^t)$. Under true null, $\mu_i^t = \mu_i^c$ and $\sigma_i^t = \sigma_i^c$, while under true non-null, $\mu_i^t = \mu_i^c + \delta$, where $\delta \sim \text{Sign}(\mu_i^c) * \text{Unif}(1, 4)$, and $\sigma_i^t \sim \text{Exp}(1)$ are redrawn independently from σ_i^c.

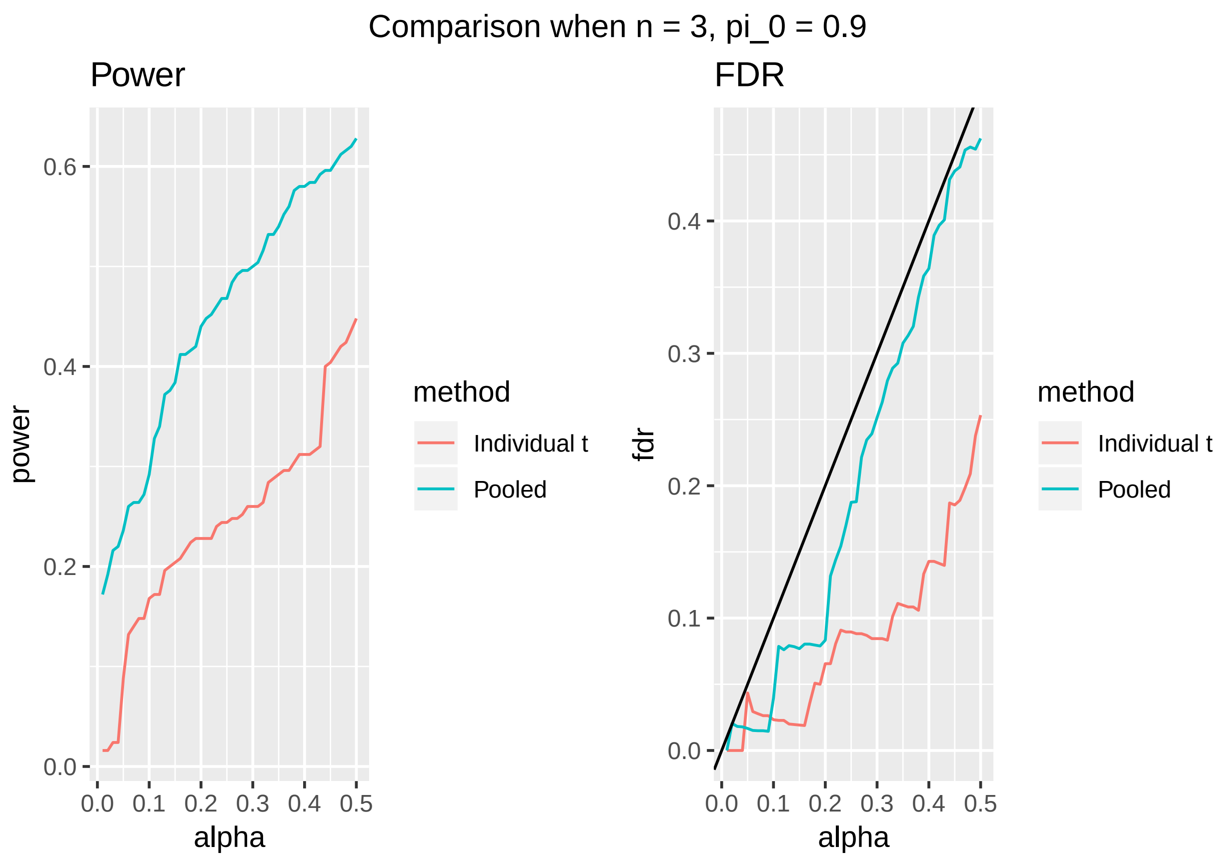
```
# Total number of tests
m <- 2500
# number of samples in each group
n_vec <- c(3, 5)
# Proportion of null
pi_0_vec <- c(0.5, 0.9)

for (n in n_vec) {
  for (pi_0 in pi_0_vec) {
    #----------Simulation setup----------#
    # Index of true null
    true_null_idx <- 1:(m*pi_0)
    non_null_idx <- (1:m)[-true_null_idx]

    # Mean of each test in control group and treatment group
    mu_control <- numeric(m)
    mu_treat <- numeric(m)
```

```r
mu_control <- rnorm(m, 0, 5)
mu_treat[true_null_idx] <- mu_control[true_null_idx]
mu_treat[non_null_idx] <- (abs(mu_control[non_null_idx]) + runif(length(non_null_idx), 1, 4))*sign(

# Standard deviation of each test in both groups
sigma_control <- rexp(m, rate = 1)
sigma_treat <- sigma_control
sigma_treat[non_null_idx] <- rexp(length(non_null_idx), rate = 1)

# Generate data
control <- replicate(n, rnorm(m, mu_control, sigma_control))
treat <- replicate(n, rnorm(m, mu_treat, sigma_treat))

dat <- cbind(control, treat)

#----------Conduct multiple testing----------#
perms <- combn(2*n, n)
perms <- perms[, 1:(ncol(perms)/2)]
stat <- matrix(0, nrow = nrow(dat), ncol = ncol(perms))

for (i in 1:ncol(perms)) {
  perm <- perms[, i]
  control <- dat[, perm]
  treat <- dat[, -perm]

  stat[, i] <- abs(unlist(lapply(1:nrow(treat),
                            FUN = function(x, treat, control)
                              t.test(treat[x, ], control[x, ])$statistic,
                            treat = treat,
                            control = control)))
}

alpha <- seq(0.01, 0.5, by = 0.01)
#------------------Individual t-test------------------#
p_val_t <- apply(dat,
                 1,
                 FUN = function(x)
                   t.test(x[1:n], x[n + (1:n)])$p.value)
q_val_t <- p.adjust(p_val_t, method = "BH")
power_t <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                         q_val = q_val_t,
                         non_null_idx = non_null_idx))
fdr_t <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                         q_val = q_val_t,
                         non_null_idx = non_null_idx))

#------------------Pooled permutation test------------------#
ref_cdf <- ecdf(as.vector(stat[, 2:ncol(stat)]))
p_val_pool <- 1 - ref_cdf(stat[, 1])
```

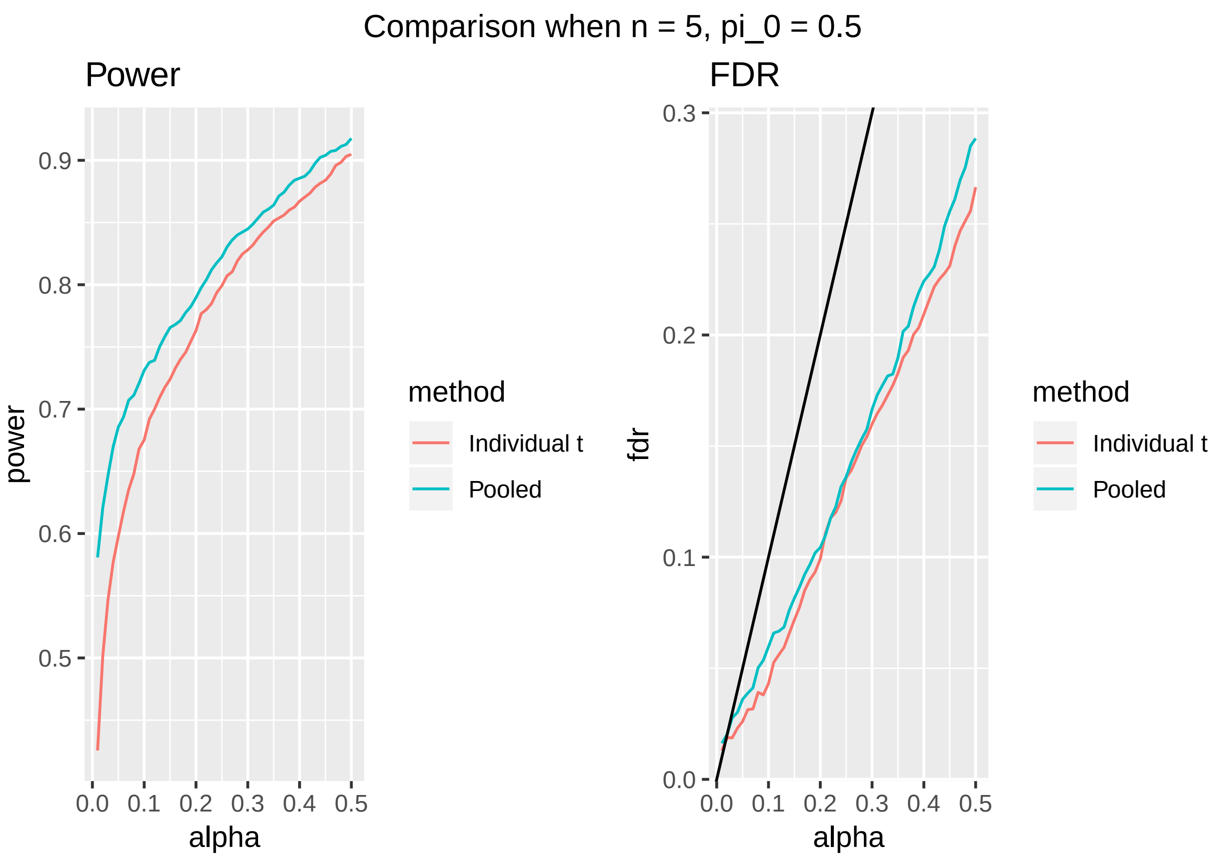
```r
    q_val_pool <- p.adjust(p_val_pool, method = "BH")
    # pi_0_hat <- sum(p_val_pool > 0.5)/m/(0.5)
    # q_val_pool_storey <- unlist(lapply(p_val_pool,
    #                                   FUN = function(x, pi_0, p_val)
    #                                     m*pi_0*x/max(sum(p_val <= x), 1),
    #                                   pi_0 = pi_0_hat,
    #                                   p_val = p_val_pool))

    power_pool <- unlist(lapply(alpha,
                        FUN = function(x, q_val, non_null_idx)
                          sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                        q_val = q_val_pool,
                        non_null_idx = non_null_idx))
    fdr_pool <- unlist(lapply(alpha,
                          FUN = function(x, q_val, non_null_idx)
                            sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                          q_val = q_val_pool,
                          non_null_idx = non_null_idx))

    # power_pool_storey <- unlist(lapply(alpha,
    #                                 FUN = function(x, q_val, non_null_idx)
    #                                   sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
    #                                 q_val = q_val_pool_storey,
    #                                 non_null_idx = non_null_idx))
    # fdr_pool_storey <- unlist(lapply(alpha,
    #                               FUN = function(x, q_val, non_null_idx)
    #                                 sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
    #                               q_val = q_val_pool_storey,
    #                               non_null_idx = non_null_idx))

    #----------Plot----------#
    dat_plot <- cbind.data.frame(c(power_pool, power_t),
                              c(fdr_pool, fdr_t),
                              c(alpha, alpha),
                              c(rep("Pooled", length(alpha)), rep("Individual t", length(alpha))))
    colnames(dat_plot) <- c("power", "fdr", "alpha", "method")
    power_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = power, color = method)) + ggti
    fdr_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = fdr, color = method)) + geom_abl
    grid.arrange(power_plot, fdr_plot, nrow = 1, top = paste0("Comparison when n = ", n, ", pi_0 = ", p
    print(paste0("The step for n = ", n, ", pi_0 = ", pi_0, " has completed!"))
  }
}
```



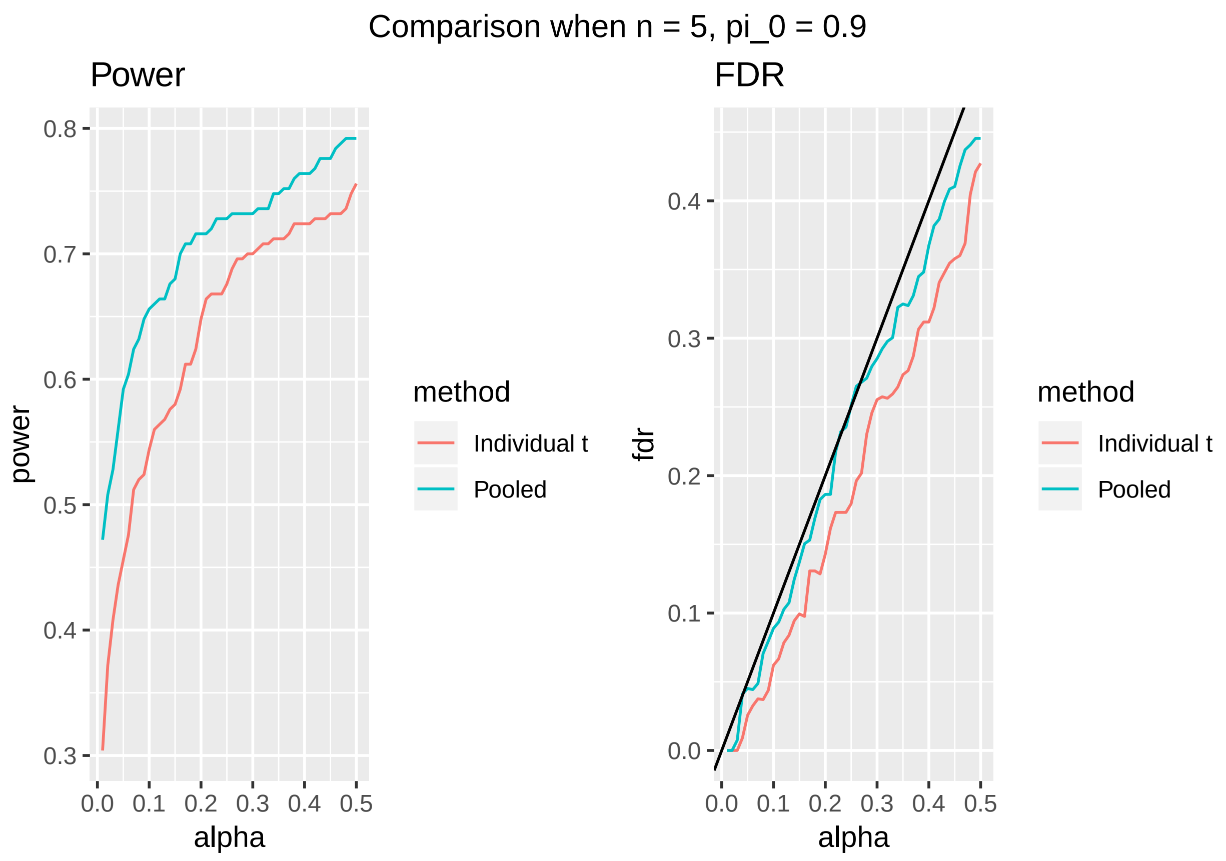
```
## [1] "The step for n = 3, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 3, pi_0 = 0.9 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.9 has completed!"
```

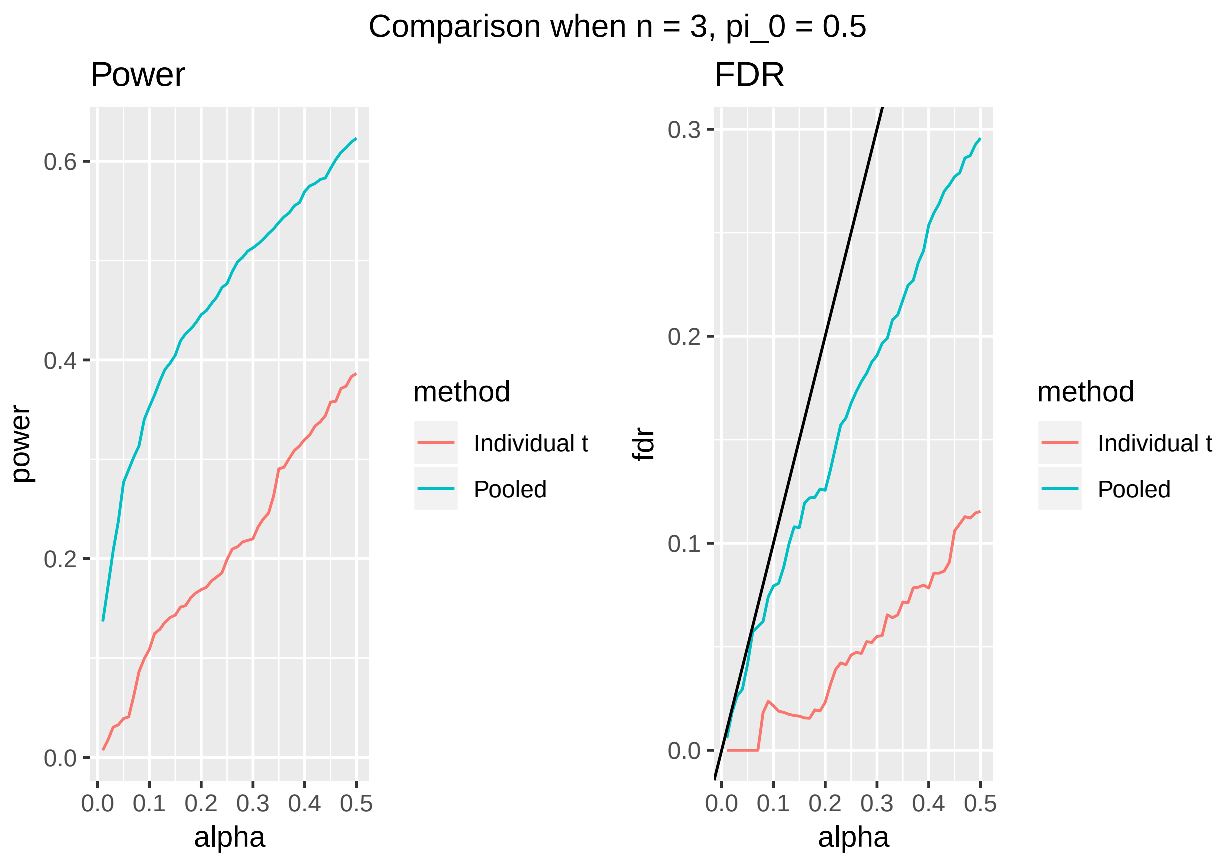
Comparison under homogeneous cauchy setup

In this comparison, all samples under true null are drawn from $\text{Cauch}(0, 1)$. Under true non-null, samples from control group are drawn from $\text{Cauch}(0, 1)$, while samples from treatment group are drawn from $\text{Cauch}(l, s)$, where $l \sim \text{Unif}(-4, 4)$, $s \sim \text{Exp}(1)$.

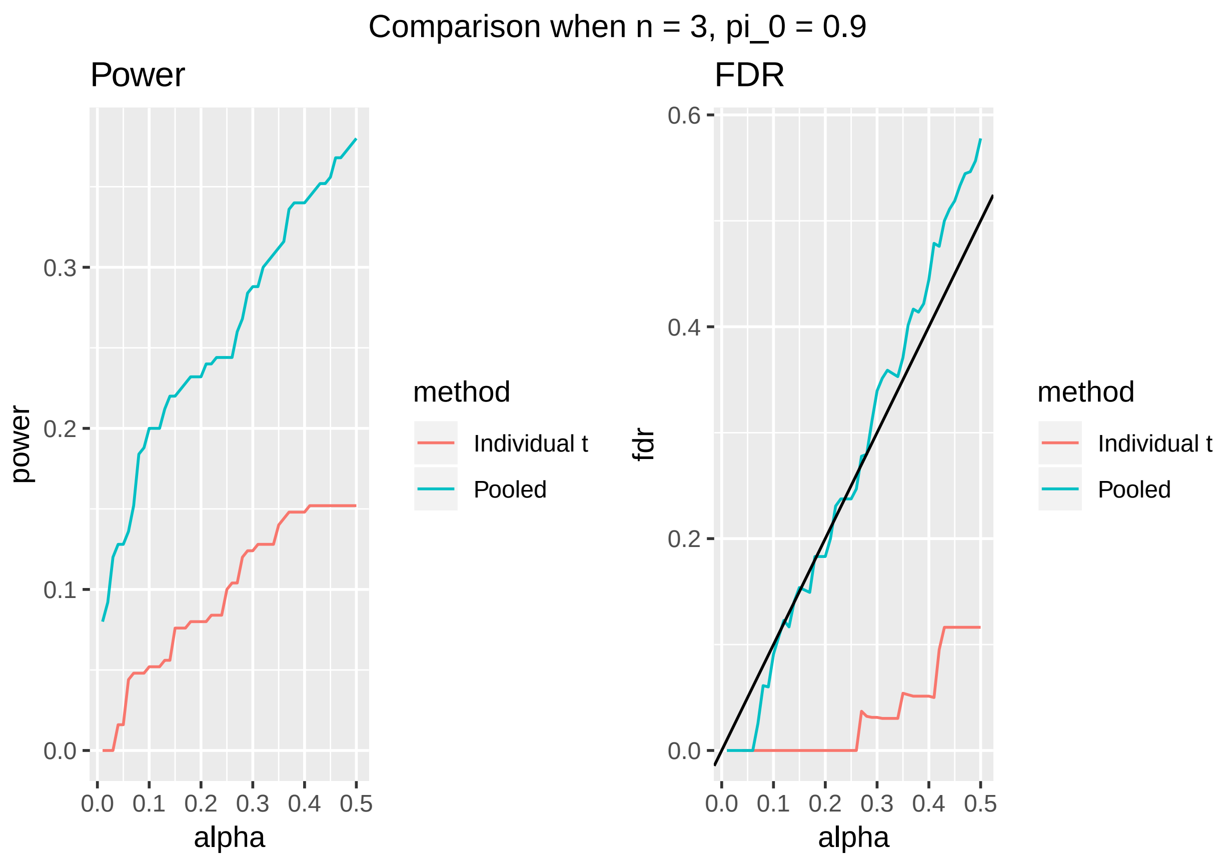
```
# Total number of tests
m <- 2500
# number of samples in each group
n_vec <- c(3, 5)
# Proportion of null
pi_0_vec <- c(0.5, 0.9)

for (n in n_vec) {
  for (pi_0 in pi_0_vec) {
    #----------Simulation setup----------#
    # Index of true null
    true_null_idx <- 1:(m*pi_0)
    non_null_idx <- (1:m)[-true_null_idx]

    # Mean of each test in control group and treatment group
    loc_control <- numeric(m)
    loc_treat <- numeric(m)
```

```r
loc_treat[true_null_idx] <- loc_control[true_null_idx]
loc_treat[non_null_idx] <- runif(length(non_null_idx), -4, 4)

# Standard deviation of each test in both groups
#scale_control <- rep(1, m)
scale_control <- rexp(m, rate = 1)
scale_treat <- scale_control
scale_treat[non_null_idx] <- rexp(length(non_null_idx), rate = 1)

# Generate data
control <- replicate(n, rcauchy(m, loc_control, scale_control))
treat <- replicate(n, rnorm(m, loc_treat, scale_treat))

dat <- cbind(control, treat)

#----------Conduct multiple testing----------#
perms <- combn(2*n, n)
perms <- perms[, 1:(ncol(perms)/2)]
stat <- matrix(0, nrow = m, ncol = ncol(perms))

for (i in 1:ncol(perms)) {
  perm <- perms[, i]
  control <- dat[, perm]
  treat <- dat[, -perm]

  stat[, i] <- abs(unlist(lapply(1:nrow(treat),
                             FUN = function(x, treat, control)
                               t.test(treat[x, ], control[x, ])$statistic,
                             treat = treat,
                             control = control)))
}

alpha <- seq(0.01, 0.5, by = 0.01)
#------------------Individual t-test------------------#
p_val_t <- apply(dat,
                 1,
                 FUN = function(x)
                   t.test(x[1:n], x[n + (1:n)])$p.value)
q_val_t <- p.adjust(p_val_t, method = "BH")
power_t <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                         q_val = q_val_t,
                         non_null_idx = non_null_idx))
fdr_t <- unlist(lapply(alpha,
                       FUN = function(x, q_val, non_null_idx)
                         sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                       q_val = q_val_t,
                       non_null_idx = non_null_idx))

#------------------Pooled permutation test------------------#
ref_cdf <- ecdf(as.vector(stat[, 2:ncol(stat)]))
p_val_pool <- 1 - ref_cdf(stat[, 1])
```

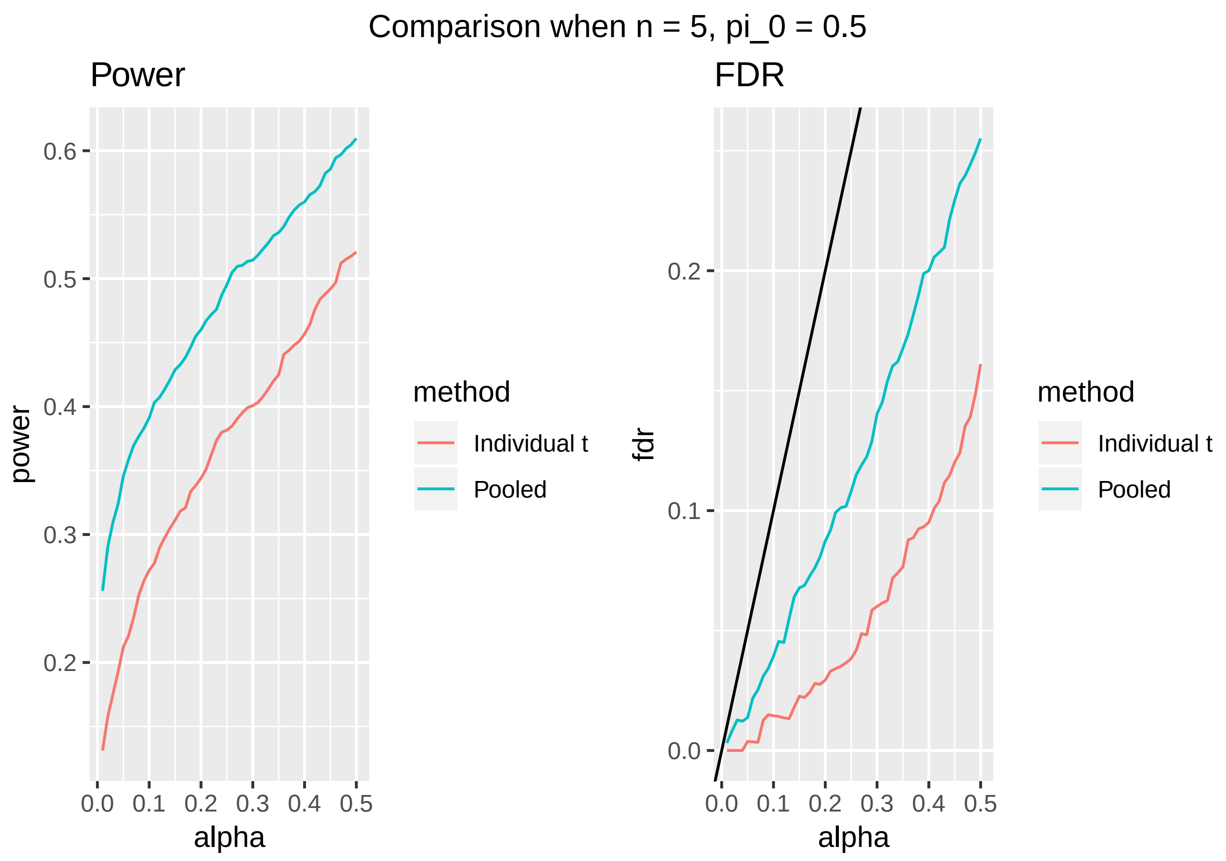
```r
    q_val_pool <- p.adjust(p_val_pool, method = "BH")
    # pi_0_hat <- sum(p_val_pool > 0.5)/m/(0.5)
    # q_val_pool_storey <- unlist(lapply(p_val_pool,
    #                                    FUN = function(x, pi_0, p_val)
    #                                      m*pi_0*x/max(sum(p_val <= x), 1),
    #                                    pi_0 = pi_0_hat,
    #                                    p_val = p_val_pool))

    power_pool <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                         q_val = q_val_pool,
                         non_null_idx = non_null_idx))
    fdr_pool <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
                           sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                         q_val = q_val_pool,
                         non_null_idx = non_null_idx))

    # power_pool_storey <- unlist(lapply(alpha,
    #                               FUN = function(x, q_val, non_null_idx)
    #                                 sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
    #                               q_val = q_val_pool_storey,
    #                               non_null_idx = non_null_idx))
    # fdr_pool_storey <- unlist(lapply(alpha,
    #                             FUN = function(x, q_val, non_null_idx)
    #                               sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
    #                             q_val = q_val_pool_storey,
    #                             non_null_idx = non_null_idx))

    #----------Plot----------#
    dat_plot <- cbind.data.frame(c(power_pool, power_t),
                                 c(fdr_pool, fdr_t),
                                 c(alpha, alpha),
                                 c(rep("Pooled", length(alpha)), rep("Individual t", length(alpha))))
    colnames(dat_plot) <- c("power", "fdr", "alpha", "method")
    power_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = power, color = method)) + ggti
    fdr_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = fdr, color = method)) + geom_abl
    grid.arrange(power_plot, fdr_plot, nrow = 1, top = paste0("Comparison when n = ", n, ", pi_0 = ", pi
    print(paste0("The step for n = ", n, ", pi_0 = ", pi_0, " has completed!"))
  }
}
```



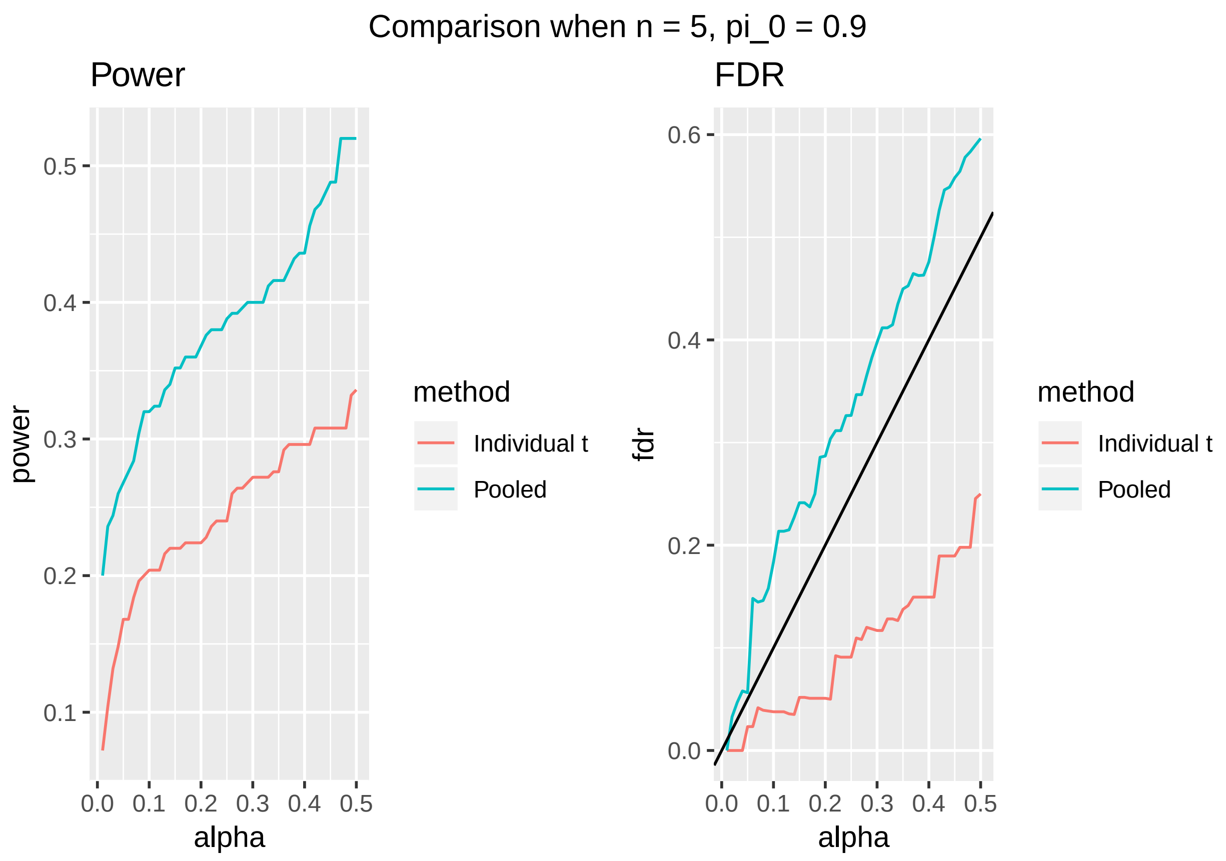
```
## [1] "The step for n = 3, pi_0 = 0.5 has completed!"

## Warning: Removed 3 rows containing missing values (geom_path).
```



```
## [1] "The step for n = 3, pi_0 = 0.9 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.9 has completed!"
```

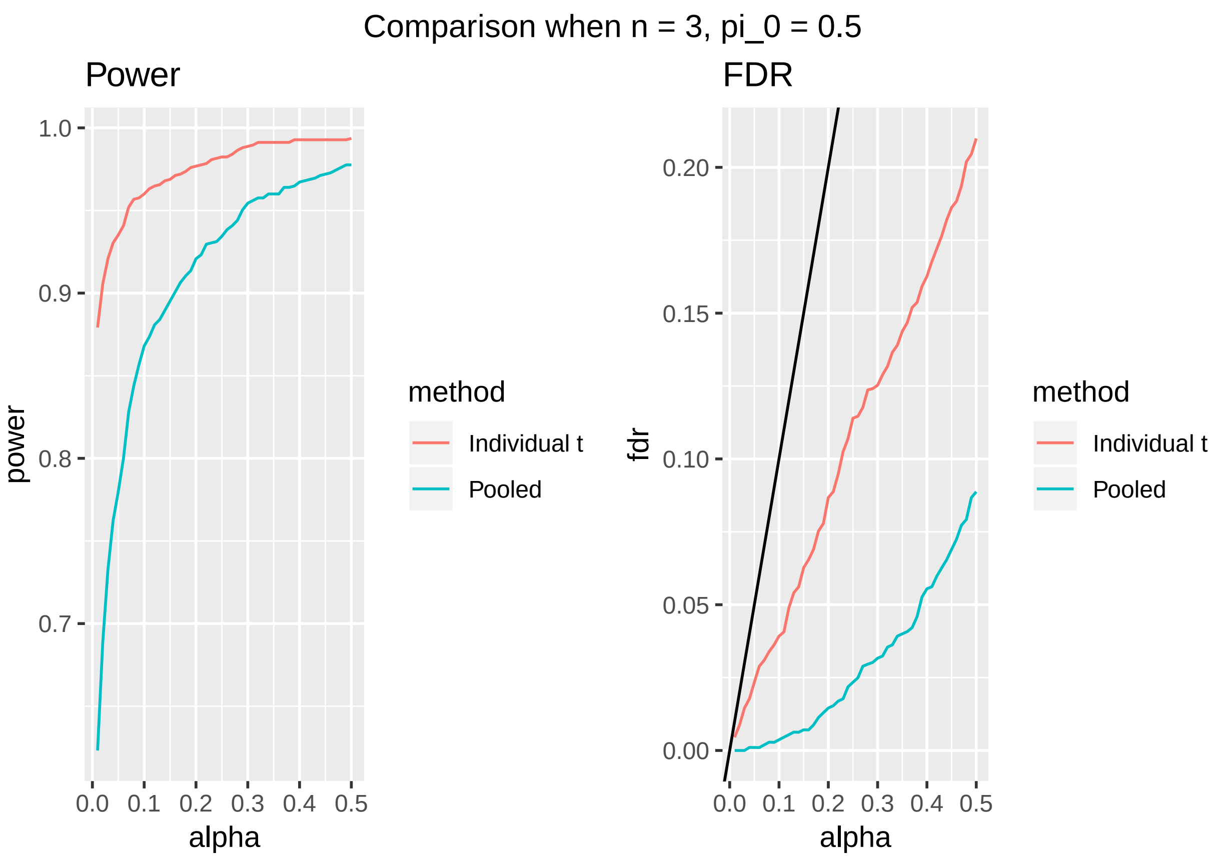
Comparison under heterogeneous Binomial setup

Follow the simulation setup in the book chapter, number of trials are drawn independently from Poisson(25). Under true null, Binomial parameter $p_i^c = p_i^t$'s are drawn from Unif$(0,1)$. Under true non-null, p_i^c's are drawn from Unif$(0,0.5)$ while p_i^t's are drawn as $p_i^t \sim p_i^c + \text{Unif}(0.2, 0.5)$.

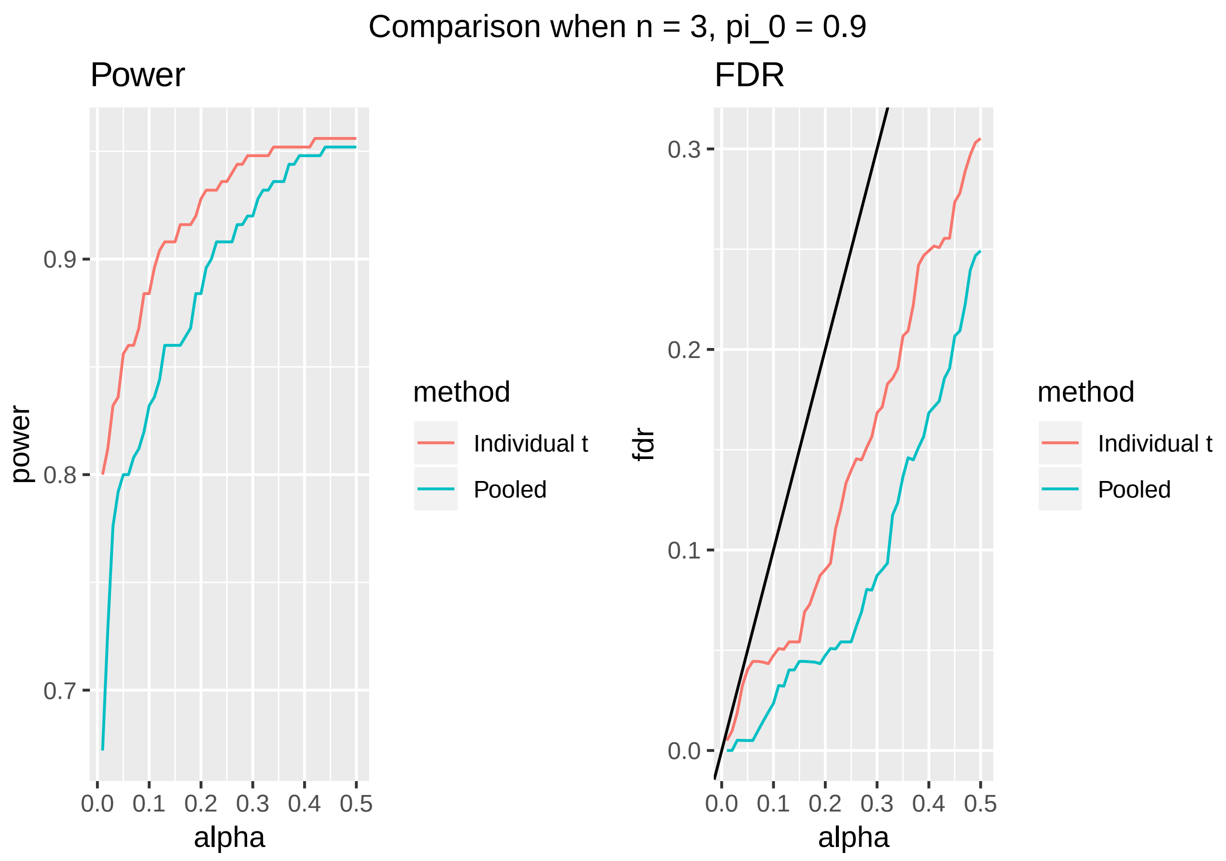
```
# Total number of tests
m <- 2500
# Poisson mean
mu_pois <- 25
# number of samples in each group
n_vec <- c(3, 5)
# Proportion of null
pi_0_vec <- c(0.5, 0.9)

for (n in n_vec) {
  for (pi_0 in pi_0_vec) {
    #----------Simulation setup----------#
    # Index of true null
    true_null_idx <- 1:(m*pi_0)
    non_null_idx <- (1:m)[-true_null_idx]

    # Binomial mean
    p_1 <- rep(0, m)
```

```
p_2 <- rep(0, m)

p_1[true_null_idx] <- runif(length(true_null_idx))
p_2[true_null_idx] <- p_1[true_null_idx]
p_1[non_null_idx] <- runif(length(non_null_idx), 0, 0.5)
p_2[non_null_idx] <- p_1[non_null_idx] + runif(length(non_null_idx), 0.2, 0.5)

# Number of trials
n_trial_1 <- replicate(n, rpois(m, mu_pois))
n_trial_2 <- replicate(n, rpois(m, mu_pois))
n_trial <- cbind(n_trial_1, n_trial_2)

# Binomial draws
bin_1 <- apply(n_trial_1, 2, FUN = function(x, m, p) rbinom(m, x, p), m = m, p = p_1)
bin_2 <- apply(n_trial_2, 2, FUN = function(x, m, p) rbinom(m, x, p), m = m, p = p_2)
binom <- cbind(bin_1, bin_2)


#----------Conduct multiple testing----------#
perms <- combn(2*n, n)
perms <- perms[, 1:(ncol(perms)/2)]
stat <- matrix(0, nrow = nrow(dat), ncol = ncol(perms))

for (i in 1:ncol(perms)) {
  perm <- perms[, i]
  control_binom <- apply(binom[, perm], 1, sum)
  treat_binom <- apply(binom[, -perm], 1, sum)
  control_n_trial <- apply(n_trial[, perm], 1, sum)
  treat_n_trial <- apply(n_trial[, -perm], 1, sum)

  stat[, i] <- unlist(lapply(1:m,
                             FUN = function(x, c_binom, t_binom, c_trial, t_trial)
                               fisher.test(matrix(c(c_binom[x], c_trial[x] - c_binom[x], t_binom[x]
                                                    nrow = 2))$p.value,
                             c_binom = control_binom,
                             t_binom = treat_binom,
                             c_trial = control_n_trial,
                             t_trial = treat_n_trial))
}

alpha <- seq(0.01, 0.5, by = 0.01)

#------------------Individual FET------------------#
p_val_fet <- stat[, 1]
q_val_fet <- p.adjust(p_val_fet, method = "BH")

power_fet <- unlist(lapply(alpha,
                           FUN = function(x, q_val, non_null_idx)
                             sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                           q_val = q_val_fet,
                           non_null_idx = non_null_idx))
fdr_fet <- unlist(lapply(alpha,
                         FUN = function(x, q_val, non_null_idx)
```

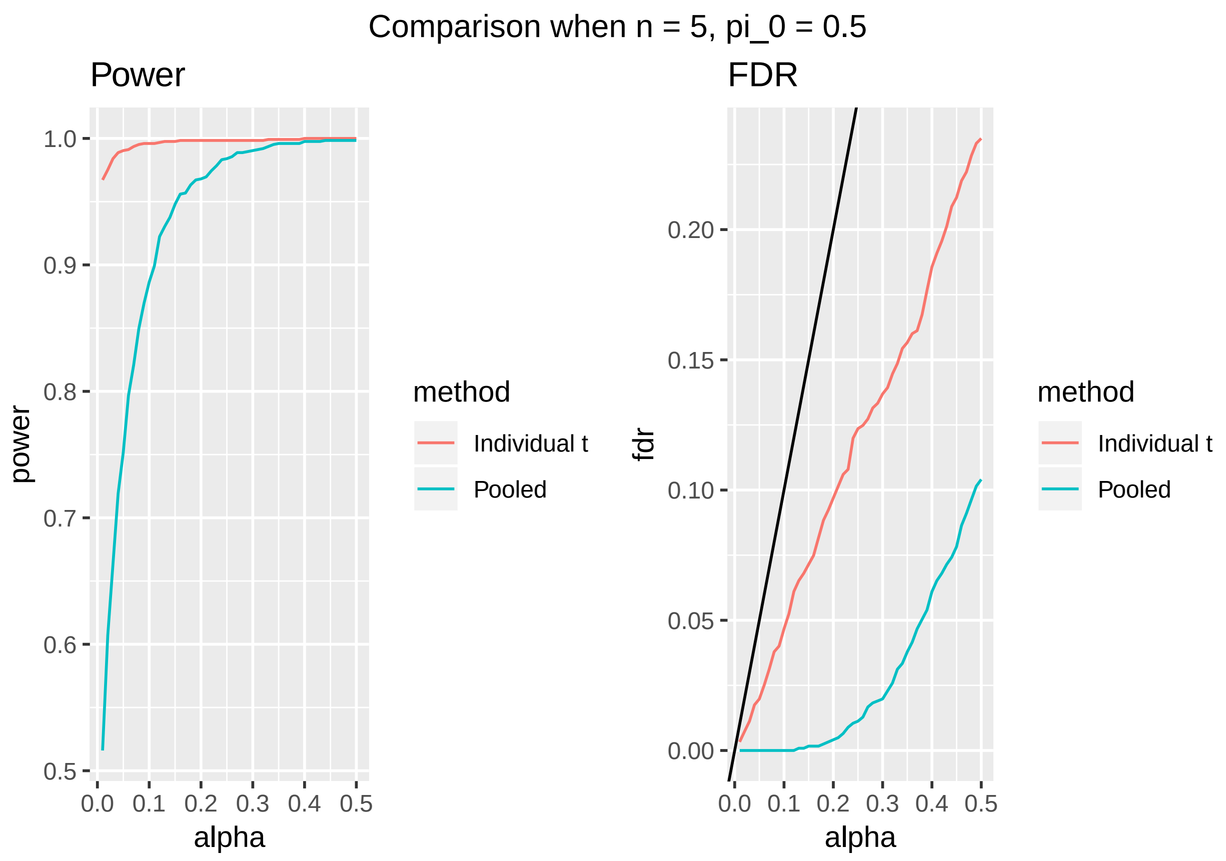
```
                              sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                            q_val = q_val_fet,
                            non_null_idx = non_null_idx))

    #------------------Pooled permutation test------------------#
    ref_cdf <- ecdf(as.vector(stat[, 2:ncol(stat)]))
    p_val_pool <- ref_cdf(stat[, 1])
    q_val_pool <- p.adjust(p_val_pool, method = "BH")

    power_pool <- unlist(lapply(alpha,
                                FUN = function(x, q_val, non_null_idx)
                                  sum(which(q_val <= x) %in% non_null_idx)/length(non_null_idx),
                                q_val = q_val_pool,
                                non_null_idx = non_null_idx))
    fdr_pool <- unlist(lapply(alpha,
                              FUN = function(x, q_val, non_null_idx)
                                sum(!(which(q_val <= x) %in% non_null_idx))/sum(q_val <= x),
                              q_val = q_val_pool,
                              non_null_idx = non_null_idx))

    #----------Plot----------#
    dat_plot <- cbind.data.frame(c(power_pool, power_fet),
                                 c(fdr_pool, fdr_fet),
                                 c(alpha, alpha),
                                 c(rep("Pooled", length(alpha)), rep("Individual t", length(alpha))))
    colnames(dat_plot) <- c("power", "fdr", "alpha", "method")
    power_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = power, color = method)) + ggti
    fdr_plot <- ggplot(data = dat_plot) + geom_line(aes(x = alpha, y = fdr, color = method)) + geom_abl
    grid.arrange(power_plot, fdr_plot, nrow = 1, top = paste0("Comparison when n = ", n, ", pi_0 = ", pi
    print(paste0("The step for n = ", n, ", pi_0 = ", pi_0, " has completed!"))
  }
}
```



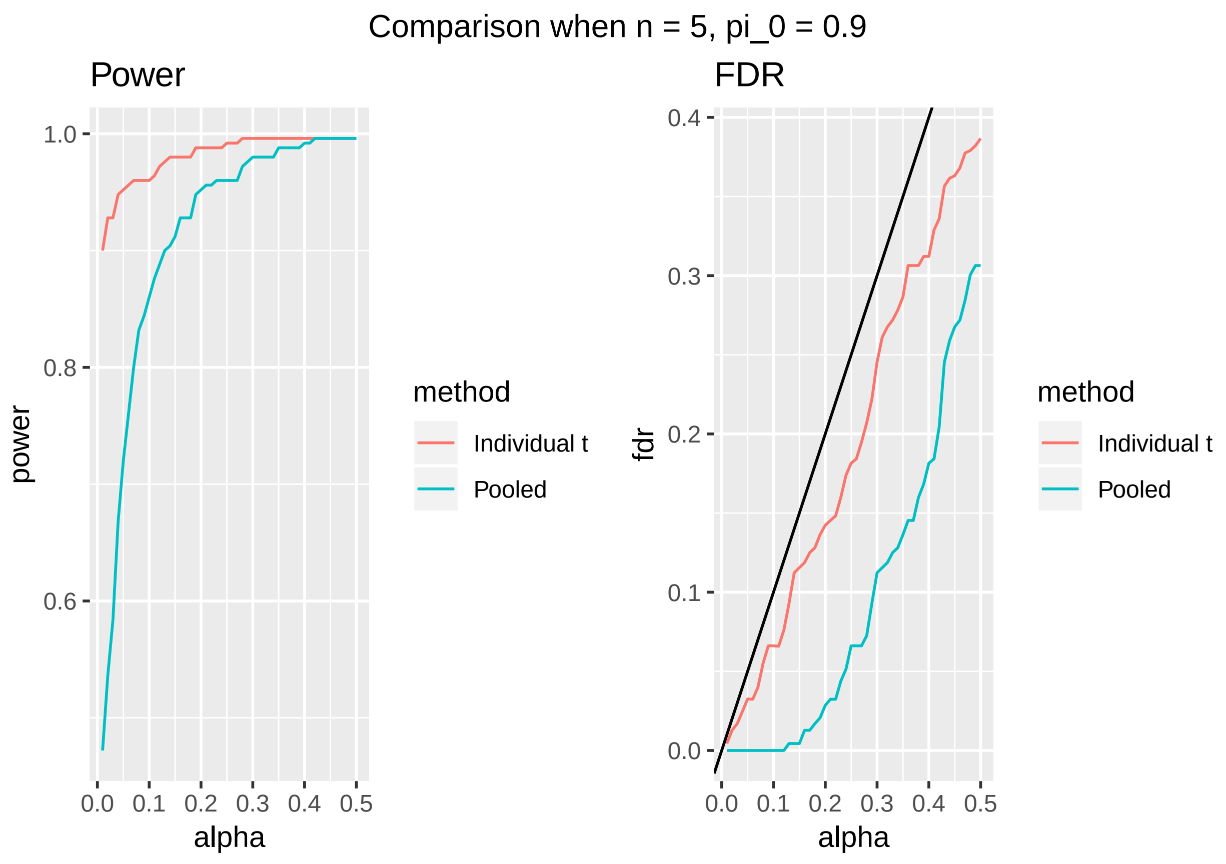
```
## [1] "The step for n = 3, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 3, pi_0 = 0.9 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.5 has completed!"
```



```
## [1] "The step for n = 5, pi_0 = 0.9 has completed!"
```