Simulation

Jordan Klein

3/11/2020

Simulation setup

```
# Write function to setup simulation with D, e, Y & B given N & M
simulation_setup <- function(N, M, beta) {
  # Generate D
  set.seed(08544)
  D <- rbinom(N, 1, .3)

  # Generate e
  set.seed(08544)
  e <- as.data.frame(replicate(M, rnorm(N)))
  names(e) <- c("e") %>%
    paste0(., 1:M)

  # Generate Y
  Y <- beta*D+e
  names(Y) <- c("Y") %>%
    paste0(., 1:M)

  # Create a dataframe with D, e, & Y
  df <- cbind(D, e, Y)

  # Generate B
  lapply(select(df, "Y1":dim(df)[2]), function(x) {
    D_Y <- as.data.frame(cbind(df$D, x))
    names(D_Y) <- c("D", "Y")

    Y1 <- D_Y$Y[D_Y$D==1]
    Y0 <- D_Y$Y[D_Y$D==0]

    mean_Y1 <- mean(Y1)
    mean_Y0 <- mean(Y0)
    statistic <- mean_Y1-mean_Y0

    return(statistic)
  }) %>%
    bind_rows(.) -> B
  names(B) <- c("B") %>%
    paste0(., 1:M)

  # Append B to dataframe & output results
  df2 <- cbind(df, B)
  return(df2)
}
```

P-values

```r
# Write function to compute unadjusted p-values from a simulation
unadj_p_values <- function(df, beta) {
  lapply(df[, c(grep("Y", names(df)))], function(x, beta) {
    D_Y <- as.data.frame(cbind(df$D, x))
    names(D_Y) <- c("D", "Y")

    # Create vectors for Y1 & Y0
    Y1 <- D_Y$Y[D_Y$D==1]
    Y0 <- D_Y$Y[D_Y$D==0]

    # Compute means of Y1 & Y0 and find their difference
    mean_Y1 <- mean(Y1)
    mean_Y0 <- mean(Y0)
    statistic <- mean_Y1-mean_Y0

    # Compute Y1 & Y0 standard error
    std_error <- sqrt((var(Y1)/length(Y1))+(var(Y0)/length(Y0)))

    # Calculate t-statistic
    t_statistic <- (statistic-beta)/std_error

    # Calculate degrees of freedom
    dof <- length(x)-2

    # Generate p-value
    p_value <- if(t_statistic < 0) {
      pt(t_statistic, dof)*2
    } else {
      pt(t_statistic, dof, lower.tail = F)*2
    }

    # Output results
    return(p_value)
  }, beta = beta) %>%
    unlist()
}

# Write function to compute Bonferroni-corrected p-values
p_val_bon_corr <- function(df, beta) {
  unadj_p_values(df, beta) %>%
    p.adjust(., method = "bonferroni") %>%
    return()
}

# Write function to compute Benjamini-Hochberg-corrected p-values
p_val_bh_corr <- function(df, beta) {
  unadj_p_values(df, beta) %>%
    p.adjust(., method = "BH") %>%
    return()
}
```

a.

Generate dataframes and write functions for 2a

```
# Generate matrix of sample sizes N and outcomes M
N_M_matrix <- expand.grid(N = c(100, 500, 1000), M = c(5, 10, 30))

# Generate dataframes of unajdusted pvalues, significance, & true value of beta
unadj_pvals_2a <- mapply("simulation_setup", N = N_M_matrix$N, M = N_M_matrix$M, beta = 0) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(unadj_p_values(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
    df <- mutate(df, beta = c(0))
    return(df)
  })

# Generate dataframes of Bonferroni-corrected pvalues, significance, & true value of beta
bon_pvals_2a <- mapply("simulation_setup", N = N_M_matrix$N, M = N_M_matrix$M, beta = 0) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(p_val_bon_corr(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
    df <- mutate(df, beta = c(0))
    return(df)
  })

# Generate dataframes of Benjamini-Hochberg-corrected pvalues, significance, & true value of beta
bh_pvals_2a <- mapply("simulation_setup", N = N_M_matrix$N, M = N_M_matrix$M, beta = 0) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(p_val_bh_corr(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
    df <- mutate(df, beta = c(0))
    return(df)
  })

# FWER function
FWER <- function(df_list) {
  lapply(df_list, function(x) {
      dim(filter(x, signif == T & beta == 0))[1] > 0
  })  %>%
  unlist() %>%
  sum()/length(df_list) -> P_V
  return(P_V)
}

# FDR function
FDR <- function(df_list) {
  # Calculate conditional expectation of V/R
  lapply(df_list, function(x) {
      dim(filter(x, signif == T))[1] > 0
  }) %>%
  unlist() %>%
  df_list[.] %>%
  lapply(function(x) {
    dim(filter(x, signif == T & beta == 0))[1]/dim(filter(x, signif == T))[1]
```

```
  }) %>%
  unlist() %>%
  mean() -> E

  # Calculate P(R>0)
  lapply(df_list, function(x) {
      dim(filter(x, signif == T))[1] > 0
  }) %>%
  unlist() %>%
  sum()/length(df_list) -> P_R

  return(E*P_R)
}
```

i.

```
# Compute FWER with unadjusted p-values
FWER(unadj_pvals_2a)
```

[1] 0.3333333

ii.

```
# Compute FWER with Bonferroni-corrected p-values
FWER(bon_pvals_2a)
```

[1] 0.1111111

iii.

```
# Compute FDR with unadjusted p-values
FDR(unadj_pvals_2a)
```

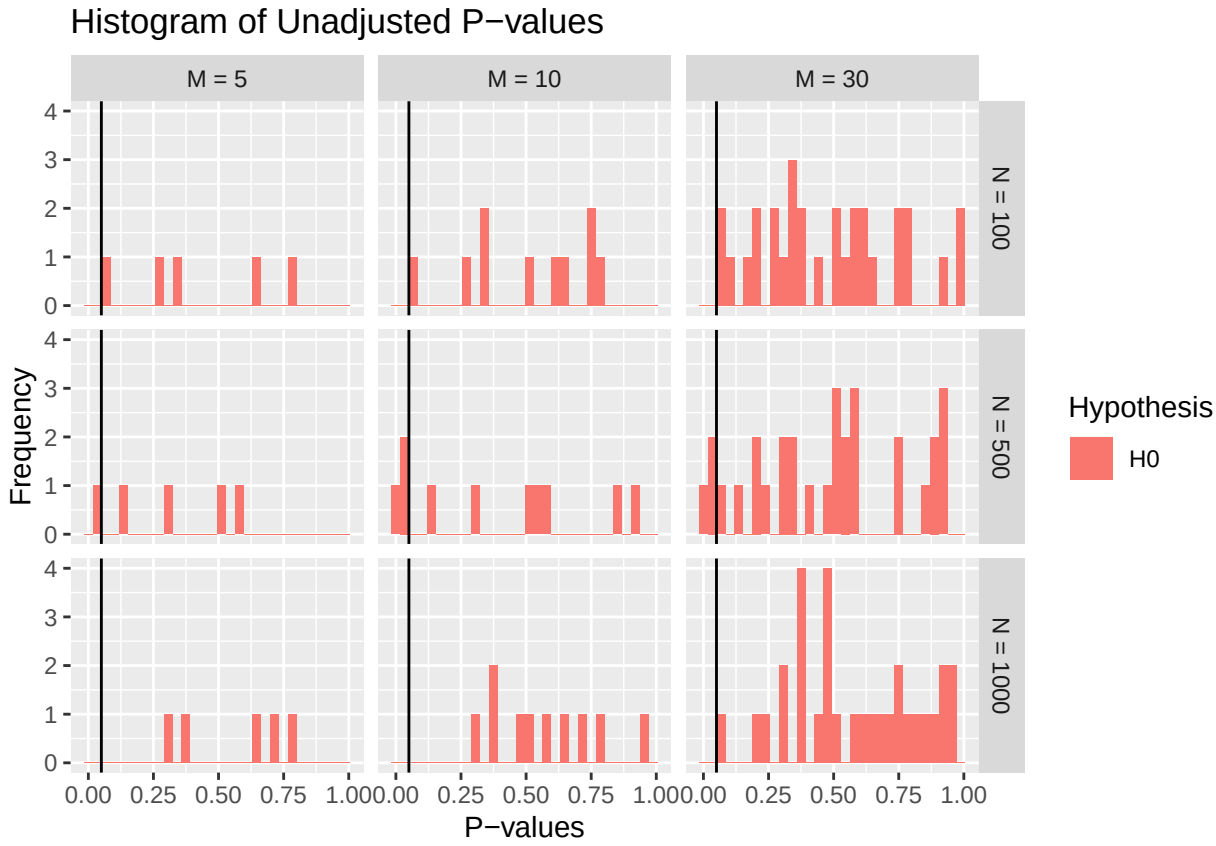
[1] 0.3333333

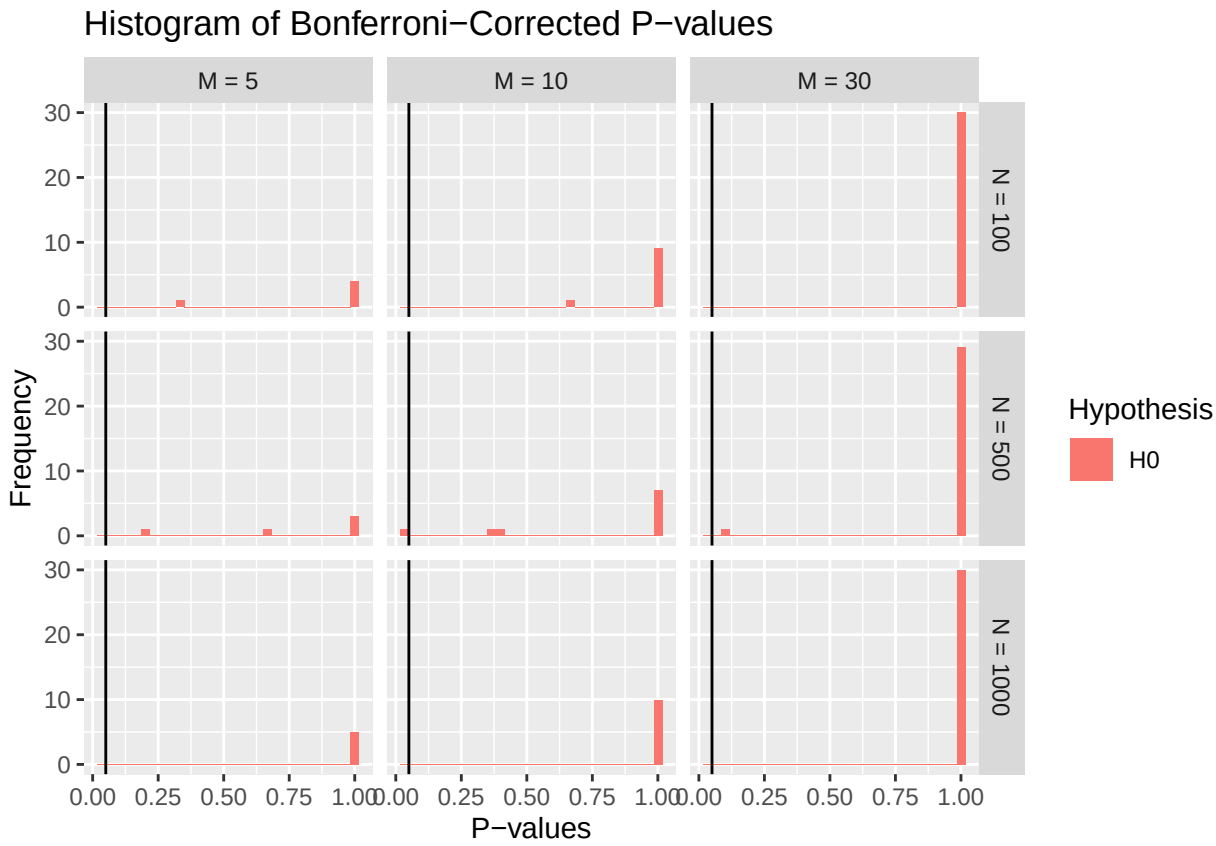
iv.

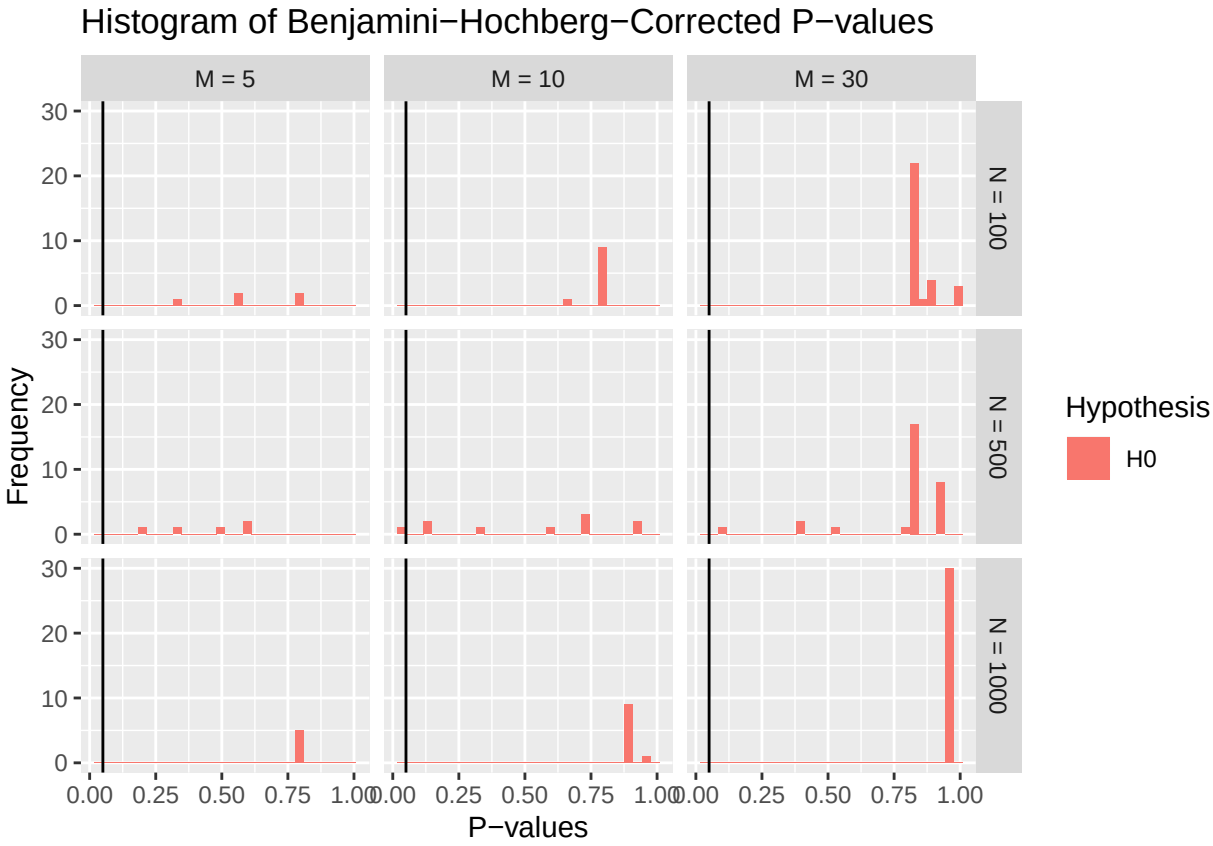
```
# Compute FDR with Benjamini-Hochberg-corrected p-values
FDR(bh_pvals_2a)
```

[1] 0.1111111

1)


Histogram of Unadjusted P−values
M = 5
M = 10
M = 30
N = 100
N = 500
N = 1000
Frequency
P−values
Hypothesis
H0


Histogram of Bonferroni−Corrected P−values
M = 5
M = 10
M = 30
N = 100
N = 500
N = 1000
Frequency
P−values
Hypothesis
H0



2)

In this simulation, with unadjusted p-values the FWER, the probability of making at least one type I error when testing many hypotheses, is 0.33. With unadjusted p-values the FDR, the expected proportion of type I errors among rejected null hypotheses, is 0.33.

3)

In this simulation, since β is always 0, the null hypothesis is always true. Therefore, every instance the null hypothesis is rejected is a type I error. Therefore, $V = R$ and $FDR = \mathbb{E}[V/R|R > 0] * P(R > 0) = \mathbb{E}[1] * P(V > 0) = FWER$, so FDR and FWER are the same in this simulation as expected.

4)

Applying the Bonferroni correction to the p-values yields a FWER of 0.11, bringing us closer to our aim of constraining the probability of making a type-I error at the $\alpha = .05$ level.

5)

Applying the Benjamini-Hochberg correction to the p-values yields a FDR of 0.11, bringing us closer to our aim of constraining the probability of making a type-I error at the $\alpha = .05$ level.

b.

Modify simulation setup function for 2b

```r
# Modify simulation setup with variable beta
simulation_setup_m <- function(N, M) {
  # Generate D
  set.seed(08544)
  D <- rbinom(N, 1, .3)

  # Generate e
  set.seed(08544)
  e <- as.data.frame(replicate(M, rnorm(N)))
  names(e) <- c("e") %>%
    paste0(., 1:M)

  # Generate beta
  beta <- data.frame(rep(1, N), rep(2, N), rep(3, N),
                     replicate(M-3, rep(0, N)))

  # Generate Y
  Y <- beta*D+e
  names(Y) <- c("Y") %>%
    paste0(., 1:M)

  # Create a dataframe with D, e, & Y
  df <- cbind(D, e, Y)

  # Generate B
  lapply(select(df, "Y1":dim(df)[2]), function(x) {
    D_Y <- as.data.frame(cbind(df$D, x))
    names(D_Y) <- c("D", "Y")

    Y1 <- D_Y$Y[D_Y$D==1]
    Y0 <- D_Y$Y[D_Y$D==0]

    mean_Y1 <- mean(Y1)
    mean_Y0 <- mean(Y0)
    statistic <- mean_Y1-mean_Y0

    return(statistic)
  }) %>%
    bind_rows(.) -> B
  names(B) <- c("B") %>%
    paste0(., 1:M)

  # Append B to dataframe & output results
  df2 <- cbind(df, B)
  return(df2)
}
```

Generate dataframes for 2b

```r
# Generate dataframes of unajdusted pvalues, significance, & true value of beta
unadj_pvals_2b <- mapply("simulation_setup_m", N = N_M_matrix$N, M = N_M_matrix$M) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(unadj_p_values(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
```

```
    df <- mutate(df, beta = c(1, 2, 3, rep(0, dim(df)[1]-3)))
    return(df)
    })

# Generate dataframes of Bonferroni-corrected pvalues, significance, & true value of beta
bon_pvals_2b <- mapply("simulation_setup_m", N = N_M_matrix$N, M = N_M_matrix$M) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(p_val_bon_corr(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
    df <- mutate(df, beta = c(1, 2, 3, rep(0, dim(df)[1]-3)))
    return(df)
    })

# Generate dataframes of Benjamini-Hochberg-corrected pvalues, significance, & true value of beta
bh_pvals_2b <- mapply("simulation_setup_m", N = N_M_matrix$N, M = N_M_matrix$M) %>%
  lapply(., function(x) {
    df <- data.frame(p_vals = c(p_val_bh_corr(x, beta = 0)))
    df <- mutate(df, signif = ifelse(p_vals < .05, T, F))
    df <- mutate(df, beta = c(1, 2, 3, rep(0, dim(df)[1]-3)))
    return(df)
    })
```

i.

```
# Compute FWER with unadjusted p-values
FWER(unadj_pvals_2b)
```

[1] 0.3333333

ii.

```
# Compute FWER with Bonferroni-corrected p-values
FWER(bon_pvals_2b)
```

[1] 0.1111111

iii.

```
# Compute FDR with unadjusted p-values
FDR(unadj_pvals_2b)
```

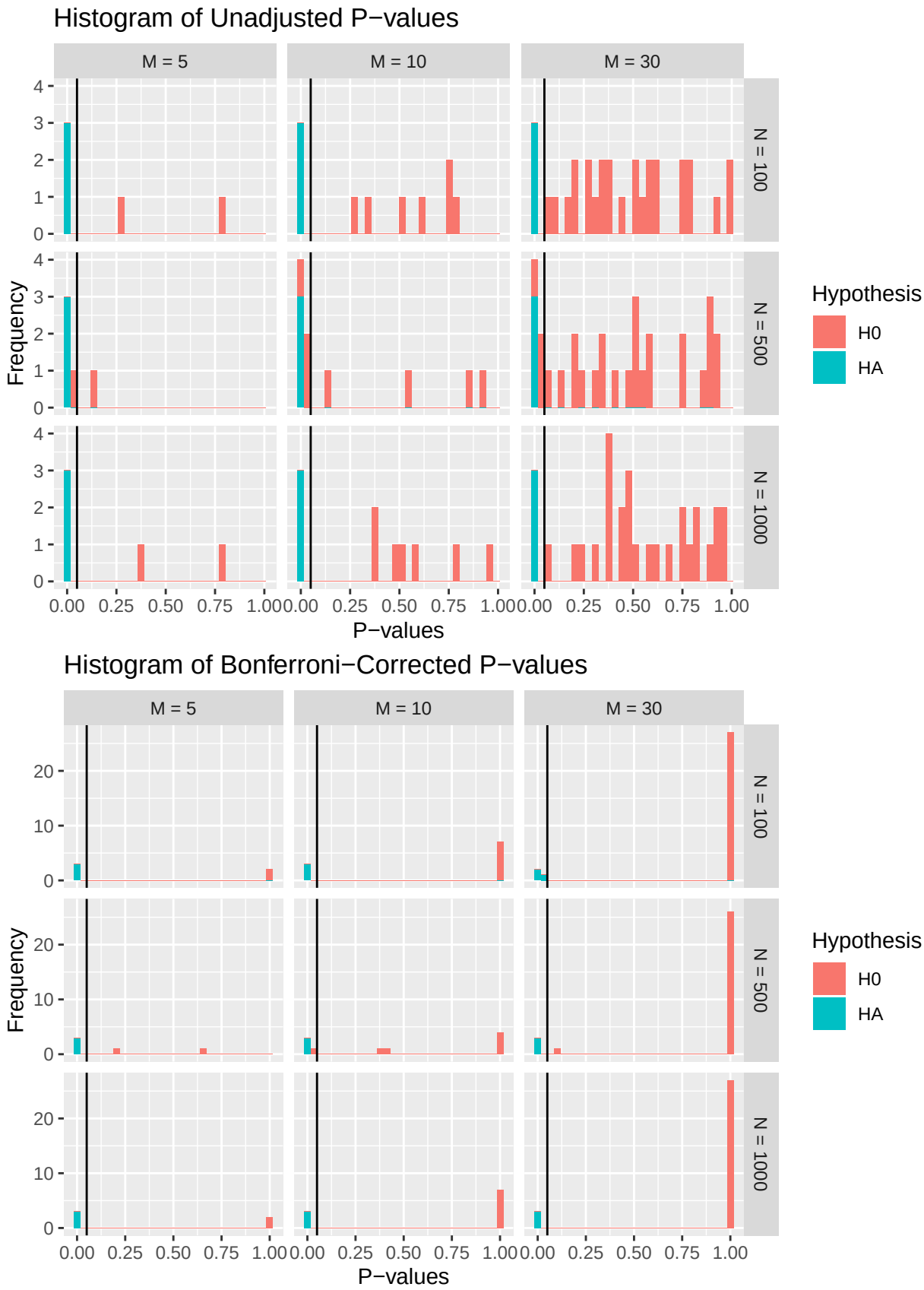
[1] 0.1388889

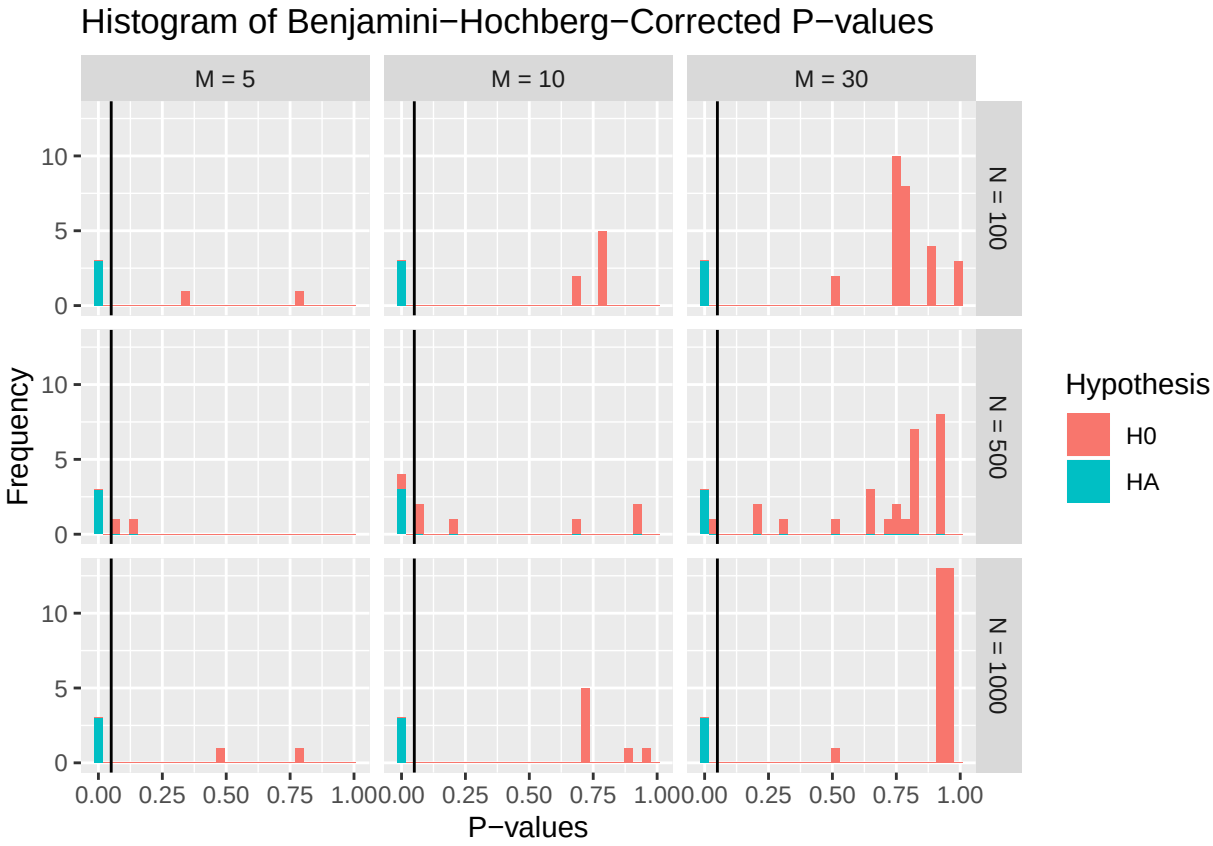
iv.

```
# Compute FDR with Benjamini-Hochberg-corrected p-values
FDR(bh_pvals_2b)
```

[1] 0.05555556

1)


Histogram of Unadjusted P−values
M = 5
M = 10
M = 30
N = 100
N = 500
N = 1000
Frequency
P−values
Hypothesis
H0
HA
Histogram of Bonferroni−Corrected P−values
M = 5
M = 10
M = 30
N = 100
N = 500
N = 1000
Frequency
P−values
Hypothesis
H0
HA



2)

In this simulation, with unadjusted p-values the FWER, the probability of making at least one type I error when testing many hypotheses, is 0.33. With unadjusted p-values the FDR, the expected proportion of type I errors among rejected null hypotheses, is 0.14.

3)

In this simulation, β varies so in some instances the null hypothesis is true while in others the alternative hypothesis is true. Therefore not every instance in which the null hypothesis is rejected is a type I error and FWER and FDR are different in this simulation as expected.

4)

Applying the Bonferroni correction to the p-values yields a FWER of 0.11, bringing us closer to our aim of constraining the probability of making a type-I error at the $\alpha = .05$ level.

5)

Applying the Benjamini-Hochberg correction to the p-values yields a FDR of 0.056, bringing us closer to our aim of constraining the probability of making a type-I error at the $\alpha = .05$ level.