#slide_48

se <- qt(1 - alp/2, n-1) * sqrt(apply(Xlist, 1, var)/n)

#slide_48

moe <- qnorm(1 - alp/2) * sdlist/sqrt(n)

#slide_49

moe <- qnorm(1 - alp/2) * sdlist/sqrt(n)

```
#2)
CovProb2 <- function(n, mu=2, alp=0.05, dist="exp", ...) {
  if (dist == "exp") {
    Xlist <- matrix(rexp(10000 * n, rate=1/mu), 10000, n)
  } else if (dist == "unif") {
    Xlist <- matrix(runif(10000 * n, ...), 10000, n)
  } else if (dist == "gamma") {
    Xlist <- matrix(rgamma(10000 * n, ...), 10000, n)
  } else {
    stop("Unsupported distribution")
  }

  Xbarlist <- rowMeans(Xlist)
  sdlist <- apply(Xlist, 1, sd)
  moe <- qnorm(1 - alp/2) * sdlist/sqrt(n)

  CI <- cbind(Xbarlist - moe, Xbarlist + moe)
  is_cover <- apply(CI, 1, function(x) mu > x[1] & mu < x[2])
  mean(is_cover)
}

set.seed(5400)
CovProb2(10, mu=1, alp=0.01, dist="unif", min=0, max=2)
```

```
## [1] 0.9636
```

```
CovProb2(30, mu=3, alp=0.05, dist="gamma", shape=2, scale=1.5)
```

```
## [1] 0.9273
```

```
CovProb2(50, mu=0.5, alp=0.1, dist="exp", rate=2)
```

```
## [1] 0.875
```

```
CovProb2(200, mu=2, alp=0.05, dist="exp", rate=0.5)
```

```
## [1] 0.9473
```

```
#3.1)
set.seed(5400)
```

```r
n_simulations = 10^5
n = 15
mu = 0

# Store all estimates
estimates = numeric(n_simulations)

for (i in 1:n_simulations) {
  sample = rt(n, df=4)

  # Calculate trimmed mean
  trimmed_mean = mean(sample[-c(which.min(sample), which.max(sample))])
  # Store estimate
  estimates[i] = trimmed_mean
}

bias = mean(estimates) - mu

variance = var(estimates)

mse = mean((estimates - mu)^2)

cat("Bias:", bias, "\n")
```

```
## Bias: -0.001876811
```

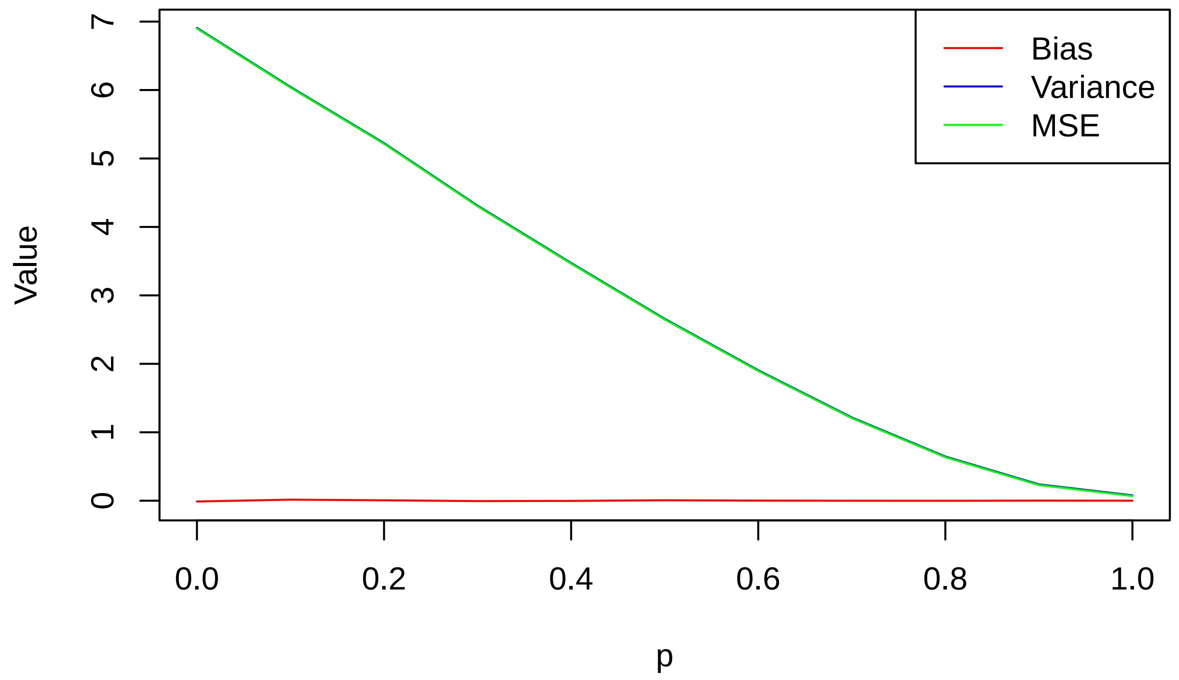
```r
cat("Variance:", variance, "\n")
```

```
## Variance: 0.1030176
```

```r
cat("MSE:", mse, "\n")
```

```
## MSE: 0.1030201
```

```r
#3.2)
set.seed(5400)

n_simulations = 10^5
n = 15
mu = 0

p_values = seq(0, 1, by=0.1)

bias_values = numeric(length(p_values))
variance_values = numeric(length(p_values))
mse_values = numeric(length(p_values))

for (j in 1:length(p_values)) {
  p = p_values[j]

  # Store all estimates for this p
  estimates = numeric(n_simulations)

  for (i in 1:n_simulations) {
    # Generate a sample
    sample = ifelse(runif(n) < p, rnorm(n, 0, 1), rnorm(n, 0, 10))

    # Calculate trimmed mean
    trimmed_mean = mean(sample[-c(which.min(sample), which.max(sample))])

    # Store estimate
    estimates[i] = trimmed_mean
  }

  bias_values[j] = mean(estimates) - mu
  variance_values[j] = var(estimates)
  mse_values[j] = mean((estimates - mu)^2)
}

plot(p_values, bias_values, type="l", col="red", ylim=c(min(c(bias_values, variance_values, mse_values))
lines(p_values, variance_values+0.01, col="blue")
lines(p_values, mse_values, col="green")
legend("topright", legend=c("Bias", "Variance", "MSE"), col=c("red", "blue", "green"), lty=1)
```



Here, we can observe that Variance and MSE are overlapped. It suggests that the bias of your estimator is very small. This is because MSE is the sum of the variance and the square of the bias. If the bias is close to zero, then the MSE will be approximately equal to the variance.

```r
#4.1)
set.seed(5400)
dat = rpois(20, 5)

sample_mean = mean(dat)
sample_sd = sd(dat)

standard_error = sample_sd / sqrt(length(dat))

z_score = qnorm(0.975)
confidence_interval = c(sample_mean - z_score*standard_error, sample_mean + z_score*standard_error)

confidence_interval
```

```
## [1] 4.32291 6.77709
```

```r
#4.2)
set.seed(5400)

n_simulations = 10^5
n = 20
lambda = 5
successes = 0

for (i in 1:n_simulations) {
  sample = rpois(n, lambda)
  sample_mean = mean(sample)
  sample_sd = sd(sample)
  standard_error = sample_sd / sqrt(n)
  z_score = qnorm(0.975)
  confidence_interval = c(sample_mean - z_score*standard_error, sample_mean + z_score*standard_error)
  # Check if the true mean is within the confidence interval
  if (lambda >= confidence_interval[1] && lambda <= confidence_interval[2]) {
    successes = successes + 1
  }
}

coverage_probability = successes / n_simulations
coverage_probability
```

```
## [1] 0.93226
```

```r
#4.3)
set.seed(5400)
dat = rpois(20, 5)
s = sum(dat)
n = length(dat)

lower_bound = 1/(2*n) * qchisq(0.025, 2*s)
upper_bound = 1/(2*n) * qchisq(0.975, 2*(s+1))

confidence_interval_exact = c(lower_bound, upper_bound)
confidence_interval_exact
```

```
## [1] 4.565666 6.683632
```

```
#4.4)
set.seed(5400)

n_simulations = 10^5
n = 20
lambda = 5
successes = 0
for (i in 1:n_simulations) {
  sample = rpois(n, lambda)
  s = sum(sample)

  lower_bound = 1/(2*n) * qchisq(0.025, 2*s)
  upper_bound = 1/(2*n) * qchisq(0.975, 2*(s+1))

  # Check if the true mean is within the confidence interval
  if (lambda >= lower_bound && lambda <= upper_bound) {
    successes = successes + 1
  }
}
coverage_probability_exact = successes / n_simulations
coverage_probability_exact
```

```
## [1] 0.95419
```

```
#install.packages("binom")
library(binom)
```

```
#5)
set.seed(5400)
theta <- 0.6
n <- 40
B <- 10000
alp <- 0.05
# generate data
Xbarlist <- rbinom(B, size=n, prob=theta) / n

moe_Wald <- qnorm(1-alp/2) *
sqrt(Xbarlist * (1 - Xbarlist)/n)
CI_Wald <- cbind(Xbarlist - moe_Wald,
Xbarlist + moe_Wald)
## simple conservative interval
moe_simple <- qnorm(1-alp/2) *
sqrt(0.5 * (1 - 0.5)/n)
CI_simple <- cbind(Xbarlist - moe_simple,
Xbarlist + moe_simple)

CI_score <- CI_CP <- CI_AC <- CI_Bayes <-
matrix(NA, B, 2)
for (i in seq(B)) {
CI_score[i, ] <- prop.test(x=(n*Xbarlist[i]), n=n,
correct=FALSE, conf.level=(1-alp))$conf.int
```

```r
CI_CP[i, ] <- binom.test(x=(n*Xbarlist[i]), n=n,
conf.level=(1-alp))$conf.int
ret <- binom.confint(x=(n*Xbarlist[i]), n=n,
conf.level=(1-alp), methods="ac")
CI_AC[i,] <- c(ret$lower, ret$upper)
ret2 <- binom.bayes(x=(n*Xbarlist[i]), n=n,
type="central", conf.level=(1-alp))
CI_Bayes[i,] <- c(ret2$lower, ret2$upper)
}

WidthCov <- function(CI) {
# get coverage probability
is_cov <- function(x) theta > x[1] & theta < x[2]
# get standard error
se <- function(x) sd(x)/sqrt(nrow(CI))
allwidth <- CI[,2] - CI[,1] # width
allcov <- apply(CI, 1, is_cov) # cov prob
# returns:
c(mean(allwidth), se(allwidth),
mean(allcov), se(allcov))
}

col.nm <- paste0("CI_", c('simple', 'Wald', 'score', 'CP', 'AC', 'Bayes'))
res <- sapply(lapply(col.nm, get), WidthCov)
colnames(res) <- col.nm
rownames(res) <- c('exp.width', 'se', 'cov.prob', 'se')
round(res, 3)
```

```
##           CI_simple CI_Wald CI_score CI_CP CI_AC CI_Bayes
## exp.width     0.310   0.300    0.287 0.314 0.288    0.291
## se            0.000   0.000    0.000 0.000 0.000    0.000
## cov.prob      0.964   0.946    0.964 0.964 0.964    0.946
## se            0.002   0.002    0.002 0.002 0.002    0.002
```