HW03

2024-02-22

1. Implementation

Generate a random covariance matrix.

```
# Args:
#   dim: The dimension of the covariance matrix
#
# Returns:
#   A valid dim x dim covariance matrix
DrawCovMat <- function(dim) {
  A <- runif(dim**2) %>% matrix(dim, dim)
  return(A %*% t(A))
}
```

Generate a random matrix of regressors.

```
# Args:
#   n_obs: The number of regression observations
#   cov_mat: A dim x dim valid covariance matrix
#
# Returns:
#   A n_obs x dim matrix of normally distributed random regressors
#   where the rows have covariance cov_mat
SimulateRegressors <- function(n_obs, cov_mat) {
  # first check whether the covariance matrix is valid
  eig <- eigen(cov_mat)
  if (!(all(eig$values >= 0))) {
    stop("The covariance matrix is not PSD")
  }
  # then, find the square root of the cov matrix so that when it multiplies the
  # standard multivariate normal random vectors, it will generate a multi-normal
  # vector with the required cov matrix.
  cov_mat_root <- eig$vectors %*% diag(sqrt(eig$values)) %*% t(eig$vectors)
  dim <- ncol(cov_mat)
  rand_normal_mat <- rnorm(n_obs * dim) %>% matrix(n_obs, dim)
  # Since we want the rows to have covariance cov_mat, we can multiply it with the
  # transpose of the square root matrix on the right.
  A <- rand_normal_mat %*% t(cov_mat_root)
  return(A)
}
```

Generate the response for a linear model.

```
# Args:
#   x_mat: A n_obs x dim matrix of regressors
#   beta: A dim-length vector of true regression coefficients
#   sigma: The standard deviation of the residuals
#
# Returns:
#   A n_obs-vector of responses drawn from the regression
#   model y_n ~ x_n^T \beta + \epsilon_n, where \epsilon_n
#   is distributed N(0, sigma^2),
SimulateResponses <- function(x_mat, beta, sigma) {
    n_obs <- nrow(x_mat)
    dim <- ncol(x_mat)
    epsilon <- rnorm(n_obs, mean = 0, sd = sigma**2) %>% matrix(n_obs, 1)
    return(x_mat %*% beta + epsilon)
}
```

Estimate the regression coefficient.

```
# Args:
#   x: A n_obs x dim matrix of regressors
#   y: A n_obs-length vector of responses
#
# Returns:
#   A dim-length vector estimating the coefficient
#   for the least squares regression y ~ x.
GetBetahat <- function(x, y) {
    return(solve(t(x) %*% x) %*% t(x) %*% y)
}
```

Estimate the residual standard deviation.

```
# Args:
#   x: A n_obs x dim matrix of regressors
#   y: A n_obs-length vector of responses
#
# Returns:
#   An estimate of the residual standard deviation
#   for the least squares regression y ~ x.
GetSigmahat <- function(x, y) {
  sigma <- y - x %*% GetBetahat(x, y)
  return(sd(sigma))
}
```

2. Draw and check

We want to show that $y_{new} - \hat{y}_{new}$ approaches the true ϵ_{new}. Thus, we can show this by forming a true linear model by assuming that the ϵ_{new} has a variance of 1. Then we use the OLS estimates to see whether the estimated ϵ approaches the variance 1.

```
N <- 500000
dim <- 4
sigma <- 1
x <- SimulateRegressors(N, DrawCovMat(dim)) # a static regressor
b <- runif(dim) # the true regressor
y <- SimulateResponses(x, b, sigma)
GetSigmahat(x, y)
```

```
## [1] 1.000676
```

```
# We can see that this approaches 1 when the observations is large
```

3. Draw a training set and test set

```
set.seed(2024)
testFunc <- function(N, P, sigma, beta, N_new){
  X <- SimulateRegressors(N, DrawCovMat(P))
  Y <- SimulateResponses(X, beta, sigma)
  beta_hat <- GetBetahat(X, Y)
  sigma_hat <- GetSigmahat(X, Y)
  # Simulate new data
  X_new <- SimulateRegressors(N_new, DrawCovMat(P))
  Y_new <- SimulateResponses(X_new, beta, sigma)
  # Form 80% predictive interval:
  #   use the new regressors multiply with the beta_hat to calculated the
  # predicted responses. The upper bound is given by the predicted responses times
  # the 90% quantile of the normal distribution with a variance we set at the
  # beginning. The lower bound works similar but with the 10% quantile. The region
  # between is the approximately 80% predictive interval for each Y_new.
  up_bound <- X_new %*% beta_hat + qnorm(0.9, mean = 0, sd = sigma_hat)
  low_bound <- X_new %*% beta_hat + qnorm(0.1, mean = 0, sd = sigma_hat)
  count <- 0
  for (i in 1:500){
    if (Y_new[i] < up_bound[i] & Y_new[i] > low_bound[i]) {
      count = count + 1
    }
  }
  count / 500 # the number is approximately 80%
}
N <- 1000
P <- 3
beta <- runif(P) # set the seed to make beta static
sigma <- 2
N_new <- 500
testFunc(N, P, sigma, beta, N_new)
```

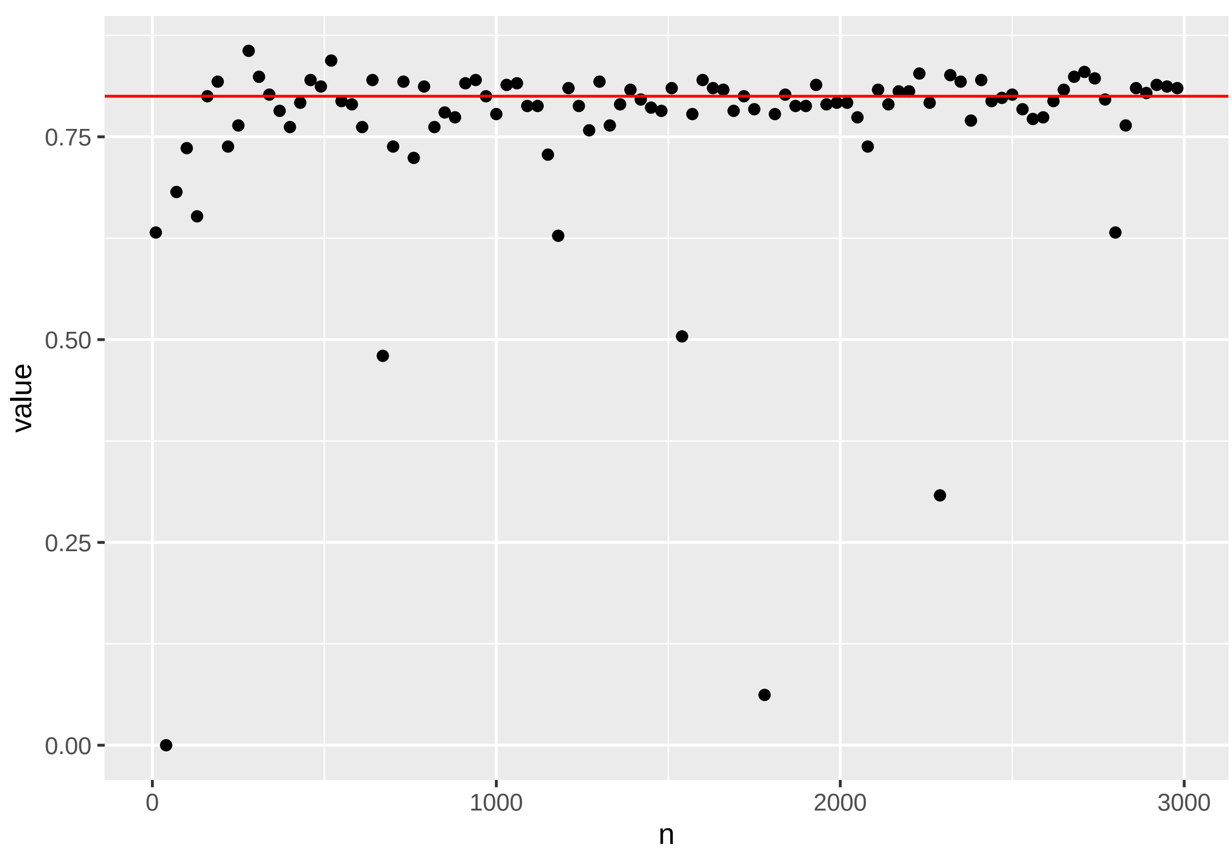
```
## [1] 0.786
```

The prediction interval performs very much in line with expectations. The reason may be that the observations are strictly i.i.d., and the number of observations is large; thus, by the law of large numbers, $\hat{\beta}$ converge to the actual β on average. Therefore, the prediction interval formed by $\hat{\sigma}$ is also approximately the true interval for the true observation y_{new}. In general, the predicted error term approches the true ϵ.

4. Vary the setting

1. N increases or decreases, all else fixed

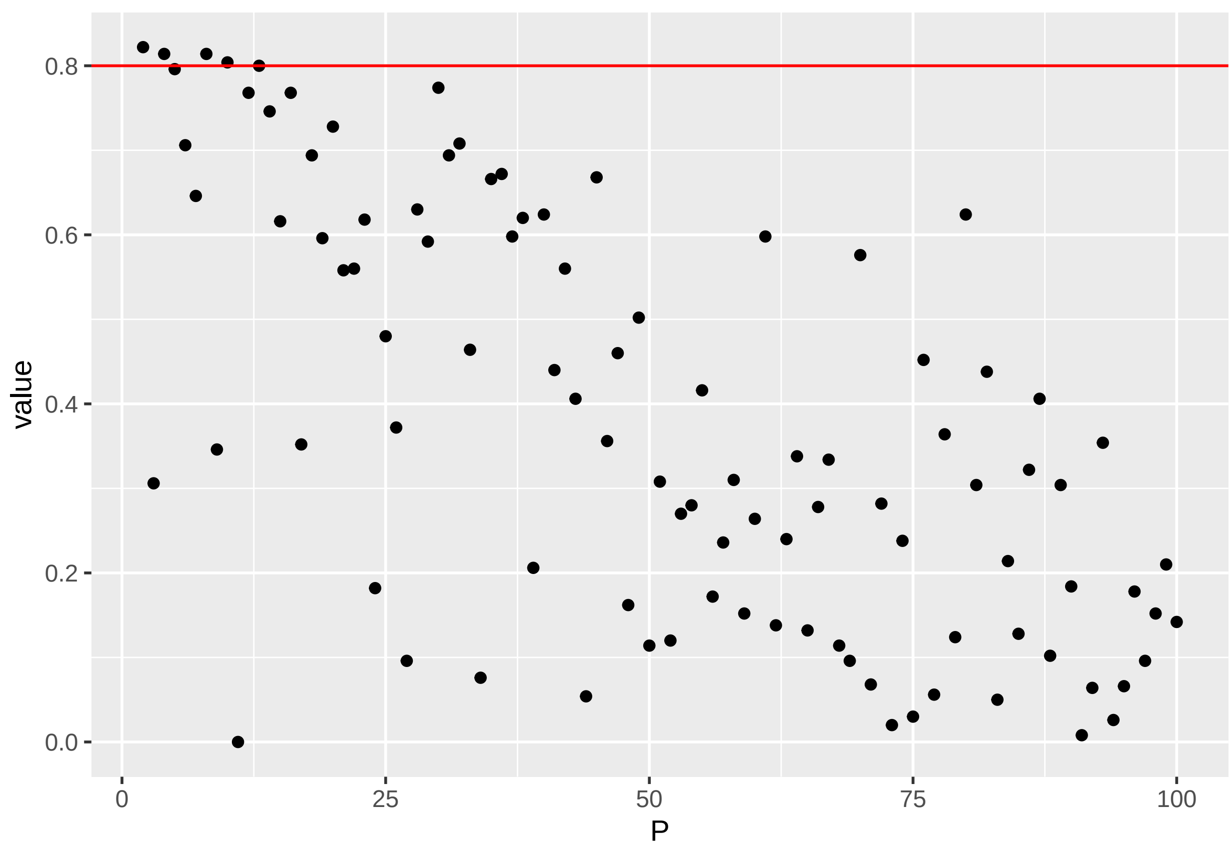
```r
set.seed(2024)
DrawFunc <- function(n_obs_vec=seq(10, 3000, by=30)) {
  result <- data.frame()
  for(n_obs in n_obs_vec) {
    result <- bind_rows(result,
                        data.frame(n=n_obs,
                                   value=testFunc(n_obs, P, sigma, beta, N_new)))
  }
  return(result)
}
result <- DrawFunc()
ggplot(result) +
  geom_point(aes(x=n, y=value), alpha=1.0) +
  geom_hline(aes(yintercept=0.8), color="red")
```



From the result, we can conclude that the number of observations does not have a significant impact on the test sets.

2. P increase and decreases (and β changes with it)

```
set.seed(2024)
DrawFunc <- function(P_vec=seq(2, 100, by=1)) {
  result <- data.frame()
  for(P_new in P_vec) {
    beta_new <- runif(P_new)
    result <- bind_rows(result, data.frame(
      P=P_new,
      value=testFunc(N, P_new, sigma, beta_new, N_new)))
  }
  return(result)
}
result <- DrawFunc()
ggplot(result) +
  geom_point(aes(x=P, y=value), alpha=1.0) +
  geom_hline(aes(yintercept=0.8), color="red")
```



We can observe from the result that as P becomes larger, the test sets becomes more likely to fail from our expectation. The number of true responses in the prediction set becomes less as P becomes larger.

3. The distribution of the residuals changes 1.t-distribution

```r
set.seed(2024)
SimulateResponses_new <- function(x_mat, beta, sigma) {
    n_obs <- nrow(x_mat)
    dim <- ncol(x_mat)
    epsilon <- rt(n_obs, df = 2 / (sigma**2 - 1)) %>% matrix(n_obs, 1)
    # Since the variance of t(v) is v/(v-2)
    return(x_mat %*% beta + epsilon)
}
testFunc_new <- function(N, P, sigma, beta, N_new){
  X <- SimulateRegressors(N, DrawCovMat(P))
  Y <- SimulateResponses(X, beta, sigma)
  beta_hat <- GetBetahat(X, Y)
  sigma_hat <- GetSigmahat(X, Y)
  # Simulate new data
  X_new <- SimulateRegressors(N_new, DrawCovMat(P))
  Y_new <- SimulateResponses_new(X_new, beta, sigma)
  # Form 80% predictive interval:
  up_bound <- X_new %*% beta_hat + qt(0.9, df = 2 / (sigma_hat**2 - 1))
  low_bound <- X_new %*% beta_hat + qt(0.1, df = 2 / (sigma_hat**2 - 1))
  count <- 0
  for (i in 1:500){
    if (Y_new[i] < up_bound[i] & Y_new[i] > low_bound[i]) {
      count = count + 1
    }
  }
  count / 500 # the number is approximately 80%
}
N <- 1000
P <- 3
beta <- runif(P) # set the seed to make beta static
sigma <- 2
N_new <- 500
df <- data.frame(dist=c("N","T"),
                 value=c(testFunc(N, P, sigma, beta, N_new),
                         testFunc_new(N, P, sigma, beta, N_new)))
df
```

```
##   dist value
## 1    N 0.786
## 2    T 1.000
```

We can see that the test sets works extremely well when $\epsilon \sim \mathrm{T}(v)$, thus, we can conclude that the prediction of $\hat{\epsilon}$ will converge more to the true ϵ when it follows a t-dist. But seems there will be risks of overfitting since the prediction lies 100% in the prediction interval sometimes.

2. chi-square distribution

```r
set.seed(2024)
SimulateResponses_new2 <- function(x_mat, beta, sigma) {
    n_obs <- nrow(x_mat)
    dim <- ncol(x_mat)
    epsilon <- rchisq(n_obs, df = sigma**2 / 2) %>% matrix(n_obs, 1)
```

```
    # Since the variance of chi(k) is 2k
    return(x_mat %*% beta + epsilon)
}
testFunc_new2 <- function(N, P, sigma, beta, N_new){
  X <- SimulateRegressors(N, DrawCovMat(P))
  Y <- SimulateResponses(X, beta, sigma)
  beta_hat <- GetBetahat(X, Y)
  sigma_hat <- GetSigmahat(X, Y)
  # Simulate new data
  X_new <- SimulateRegressors(N_new, DrawCovMat(P))
  Y_new <- SimulateResponses_new2(X_new, beta, sigma)
  # Form 80% predictive interval:
  up_bound <- X_new %*% beta_hat + qchisq(0.9, df = sigma**2 / 2)
  low_bound <- X_new %*% beta_hat + qchisq(0.1, df = sigma**2 / 2)
  count <- 0
  for (i in 1:500){
    if (Y_new[i] < up_bound[i] & Y_new[i] > low_bound[i]) {
      count = count + 1
    }
  }
  count / 500 # the number is approximately 80%
}
N <- 1000
P <- 3
beta <- runif(P) # set the seed to make beta static
sigma <- 2
N_new <- 500
df <- data.frame(dist=c("N","Chi-Square"),
                 value=c(testFunc(N, P, sigma, beta, N_new),
                         testFunc_new2(N, P, sigma, beta, N_new)))
df
```

```
##          dist value
## 1           N 0.786
## 2 Chi-Square 0.686
```

Seems that the Chi-squre distribution performs worse than the normal distribution in the prediction of the error terms.

3. uniform distribution

```
set.seed(2024)
SimulateResponses_new3 <- function(x_mat, beta, sigma) {
    n_obs <- nrow(x_mat)
    dim <- ncol(x_mat)
    epsilon <- runif(n_obs, min=0, max=sqrt(12 * sigma**2)) %>% matrix(n_obs, 1)
    # Since the variance of uniform(0,b) is b^2/12
    return(x_mat %*% beta + epsilon)
}
testFunc_new3 <- function(N, P, sigma, beta, N_new){
  X <- SimulateRegressors(N, DrawCovMat(P))
  Y <- SimulateResponses(X, beta, sigma)
```

```r
  beta_hat <- GetBetahat(X, Y)
  sigma_hat <- GetSigmahat(X, Y)
  # Simulate new data
  X_new <- SimulateRegressors(N_new, DrawCovMat(P))
  Y_new <- SimulateResponses_new3(X_new, beta, sigma)
  # Form 80% predictive interval:
  up_bound <- X_new %*% beta_hat + qunif(0.9, min=0, max=sqrt(12 * sigma**2))
  low_bound <- X_new %*% beta_hat + qunif(0.1, min=0, max=sqrt(12 * sigma**2))
  count <- 0
  for (i in 1:500){
    if (Y_new[i] < up_bound[i] & Y_new[i] > low_bound[i]) {
      count = count + 1
    }
  }
  count / 500 # the number is approximately 80%
}
N <- 1000
P <- 3
beta <- runif(P) # set the seed to make beta static
sigma <- 2
N_new <- 500
df <- data.frame(dist=c("N","uniform"),
                 value=c(testFunc(N, P, sigma, beta, N_new),
                         testFunc_new3(N, P, sigma, beta, N_new)))
df
```

```
##      dist value
## 1       N 0.786
## 2 uniform 0.736
```

We can see that the test results are similar when using uniform and normal distribution. Thus, uniform disribution is also a candidates for the ϵ model.