

# Supplemental Material to “SimEngine: A Modular Framework for Statistical Simulations in R”

by *Avi Kenny and Charles J. Wolock*

## A. Simulation-based power calculation

Calculating statistical power is a critical step in the design of experiments. For a given study design, the statistical power is defined as the probability that a hypothesis test correctly rejects the null hypothesis (assuming it is false). Sometimes, the sample size for a study is considered fixed, and interest lies in calculating power. More often, investigators want to know what sample size is needed to reject the null hypothesis at a given power level (e.g. 80% or 90%). We assume that the reader has some familiarity with statistical hypothesis testing.

For simple study designs (e.g. an individually randomized controlled trial with two groups), formulas exist to calculate the sample size necessary to reject the null hypothesis under certain assumptions on the distribution of the outcome, the effect size, etc. For example, in an experiment comparing means between two groups, the following formula is used to calculate the necessary sample size to reject the null hypothesis of no difference with power  $1 - \beta$  while maintaining type I error rate  $\alpha$ , where the outcome variable has means  $(\mu_0, \mu_1)$  and variances  $(\sigma_0^2, \sigma_1^2)$  in the two groups and  $z_q$  denotes the  $q$ th quantile of the standard normal distribution:

$$n = \frac{(z_{\alpha/2} + z_{\beta})^2(\sigma_0^2 + \sigma_1^2)}{(\mu_0 - \mu_1)^2}. \quad (1)$$

However, for more complex study designs or analysis plans, sample size formulas may not exist. In these situations, an easier approach is to conduct a simulation-based power calculation. The basic idea is that to repeatedly simulate the entire experiment and calculate the proportion of experiments in which the null hypothesis is rejected; this is the estimated power. Simulating the entire experiment will typically involve generating a dataset and then running an analysis that involves a hypothesis test. Randomness is usually introduced into the process through the dataset generation, although sometimes a population dataset will be fixed and randomness induced by taking samples from that population (e.g., to simulate survey data analyses). Often, the most difficult aspect of a simulation-based power calculation is simulating a dataset that accurately reflects the nuances (e.g., the correlation structure) of the real dataset.

To calculate sample size at a fixed power level (e.g., 90%), a “guess and check” approach may be used. With this approach, the simulation is run with an arbitrarily-selected sample size  $n_1$  to estimate the power. If power is estimated to be lower than 90%, a new, larger value  $n_2$  is selected and the simulation is run again again. This procedure is repeated until the estimated power is roughly 90%.

We illustrate this process by simulating a randomized controlled trial and comparing the results to what we would have attained using (1). First, we declare a new simulation object and write a function to generate data:

```
R> sim <- new_sim()
R> create_rct_data <- function(n, mu_0, mu_1, sigma_0, sigma_1) {
+   group <- sample(rep(c(0,1),n))
+   outcome <- (1-group) * rnorm(n=n, mean=mu_0, sd=sigma_0) +
+     group * rnorm(n=n, mean=mu_1, sd=sigma_1)
```

```
+   return(data.frame("group"=group, "outcome"=outcome))
+ }
R> create_rct_data(n=3, mu_0=3, mu_1=4, sigma_0=0.1, sigma_1=0.1)
  group outcome
1     1 4.073832
2     0 2.917953
3     1 3.969461
4     0 3.032951
5     0 2.917953
6     1 3.969461
```

Next, we add a function that takes a dataset generated by the `create_rct_data` function and runs a statistical test to determine whether to reject the null hypothesis:

```
R> run_test <- function(data) {
+   test_result <- t.test(outcome~group, data=data)
+   return(as.integer(test_result$p.value<0.05))
+ }
```

Next, we write the simulation script and tell [SimEngine](#) to run 1,000 simulation replicates each for four sample sizes.

```
R> sim %<>% set_script(function() {
+   data <- create_rct_data(n=L$n, mu_0=17, mu_1=18, sigma_0=2, sigma_1=2)
+   reject <- run_test(data)
+   return (list("reject"=reject))
+ })
R> sim %<>% set_levels(n=c(20,40,60,80))
R> sim %<>% set_config(num_sim=1000)
```

Next, we run the simulation. After obtaining results, we calculate power by averaging the reject variable using the `summarize` function, which indicates the percentage of simulations in which the null hypothesis was rejected.

```
R> sim %<>% run()
|#####| 100%
Done. No errors or warnings detected.
R> power_sim <- sim %>% summarize(
+   list(stat="mean", name="power", x="reject")
+ )
R> print(power_sim)
  level_id  n n_reps power
1         1 20   1000 0.364
2         2 40   1000 0.578
3         3 60   1000 0.736
4         4 80   1000 0.844
```

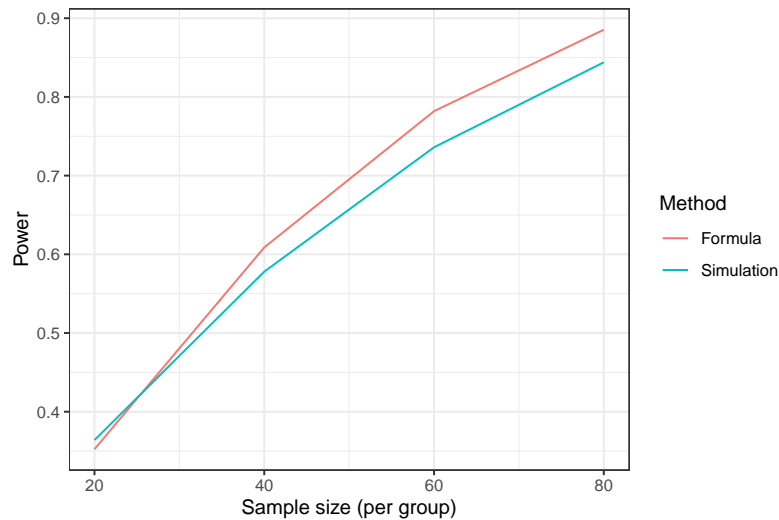
We can compare the results to what we obtain from (1).

```
R> power_formula <- sapply(c(20,40,60,80), function(n) {
+   pnorm(sqrt((n*(17-18)^2)/(2^2+2^2)) - qnorm(0.025, lower.tail=F))
+ })
R> library(ggplot2)
R> ggplot(data.frame(
+   n = rep(c(20,40,60,80), 2),
+   power = c(power_sim$power, power_formula),
```

```

+   which = rep(c("Simulation", "Formula"), each=4)
+ ), aes(x=n, y=power, color=factor(which))) +
+   geom_line() +
+   theme_bw() +
+   labs(color="Method", y="Power", x="Sample size (per group)")

```



Of course, real applications will typically involve much more complex data generating mechanisms and analyses, but the same basic principles illustrated in the code above will generally apply to simulation-based power calculations.

## B. Comparing two standard error estimators

When developing a novel statistical method, we often wish to compare our proposed method with one or more existing methods. This serves to highlight the differences between our method and whatever is used in common practice. Generally, we wish to examine realistic settings, motivated by statistical theory, in which the novel method confers some advantage over the alternatives.

In this example, we will consider the problem of estimating the variance-covariance matrix of the least-squares estimator in linear regression. We assume the reader has some familiarity with linear regression.

Suppose the dataset consists of  $n$  independent observations  $\{(Y_1, X_1), (Y_2, X_2), \dots, (Y_n, X_n)\}$ , where  $X$  and  $Y$  are both scalar variables. A general linear regression model posits that

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i,$$

where  $\epsilon_i$  is a mean-zero noise term with variance  $\sigma_i^2$ . We refer to this as a heteroskedastic model, since the variances need not be equal across all  $i$ . This is the true data-generating model. We note that a more restrictive (but misspecified) model assumes that there is a common variance  $\sigma^2$  such that  $\sigma_i^2 = \sigma^2$  for all  $i$ . We refer to this incorrect model as the homoskedastic model.

In linear regression, the least-squares method is often used to construct an estimate  $\hat{\beta}$  of the coefficient vector  $\beta$ . For the purposes of building confidence intervals and performing hypothesis tests, we may also wish to estimate the standard error of the least squares estimator. Perhaps the two most common ways to do this are:

1. Use a model-based standard error that assume homoskedasticity. This is the estimator used by default in most statistical software, including the `lm` function in R.

2. Use a so-called sandwich standard error ([White, 1980](#)). Statistical theory shows that this estimator will be consistent even under heteroskedasticity.

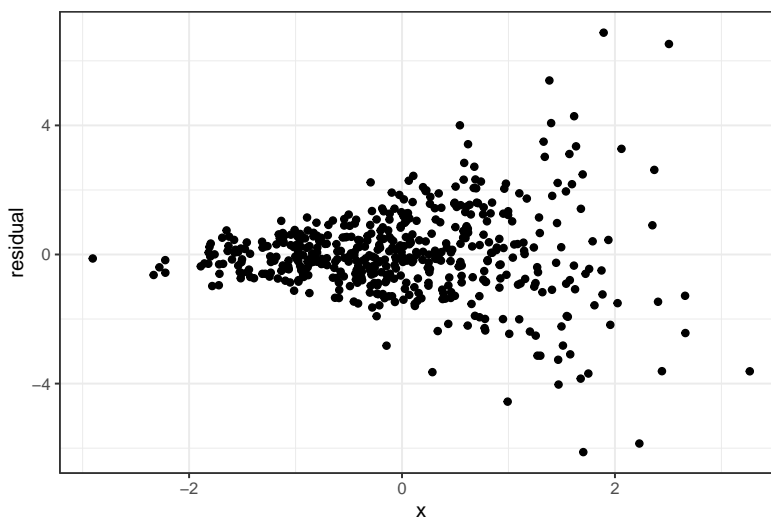
We will carry out a small simulation study to compare these two estimators.

We start by declaring a new simulation object and writing a function that generates some data according to the heteroskedastic model. In this simulation,  $\sigma_i^2$  is larger for larger values of  $X_i$ .

```
R> sim <- new_sim()
R> create_regression_data <- function(n) {
+   beta <- c(-1, 10)
+   x <- rnorm(n)
+   sigma2 <- exp(x)
+   y <- rnorm(n=n, mean=(beta[1]+beta[2]*x), sd = sqrt(sigma2))
+   return(data.frame(x=x, y=y))
+ }
```

For a sense of what this sort of heteroskedasticity looks like empirically, we generate a dataset, fit a linear regression model, and make a scatterplot of the residuals against  $X$ .

```
R> dat <- create_regression_data(n=500)
R> linear_model <- lm(y~x, data=dat)
R> dat$residuals <- linear_model$residuals
R> library(ggplot2)
R> ggplot(dat, aes(x=x, y=residuals)) +
+   geom_point() +
+   theme_bw() +
+   labs(x="x", y="residual")
```



Now, we declare two methods: one returns the least squares estimate  $\hat{\beta}$  and model-based estimate of the variance-covariance matrix of  $\hat{\beta}$ , and the other returns  $\hat{\beta}$  and the sandwich estimate from `vcovHC` in the [sandwich](#) package ([Zeileis, 2004](#); [Zeileis et al., 2020](#)).

```
R> model_vcov <- function(data) {
+   mod <- lm(y~x, data=data)
+   return(list("coef"=mod$coefficients, "vcov"=diag(vcov(mod))))
+ }
R> sandwich_vcov <- function(data) {
+   mod <- lm(y~x, data=data)
```

```
+   return(list("coef"=mod$coefficients, "vcov"=diag(vcovHC(mod))))
+ }
```

Next, we write the simulation script. This script returns a point estimate and a standard error estimate for both the intercept parameter  $\beta_0$  and the slope parameter  $\beta_1$ . We tell [SimEngine](#) to run 500 simulation replicates for each of four sample sizes. It is important to use the seed argument in `set_config` so that our results will be reproducible. In addition, we use the packages option to load the [sandwich](#) package. Loading packages via `set_config` (as opposed to running `library(sandwich)`) is required if running simulations in parallel. Finally, we run the simulation.

```
R> sim %<>% set_script(function() {
+   data <- create_regression_data(n=L$n)
+   estimates <- use_method(L$estimator, list(data))
+   return(list(
+     "beta0_est" = estimates$coef[1],
+     "beta1_est" = estimates$coef[2],
+     "beta0_se_est" = sqrt(estimates$vcov[1]),
+     "beta1_se_est" = sqrt(estimates$vcov[2])
+   ))
+ })
R> sim %<>% set_levels(
+   estimator = c("model_vcov", "sandwich_vcov"),
+   n = c(50, 100, 500, 1000)
+ )
R> sim %<>% set_config(
+   num_sim = 500,
+   seed = 24,
+   packages = c("sandwich")
+ )
R> sim %<>% run()
|#####| 100%
Done. No errors or warnings detected.
```

Now we can summarize the results using `summarize`. There are two main quantities of interest. The primary purpose of the standard error estimate for  $\hat{\beta}$  is to form confidence intervals, so we examine (1) the average half-width of the resulting interval (simply 1.96 times the average standard error estimate across simulation replicates), and (2) the estimated coverage of the interval, which is simply the proportion of simulation replicates in which the interval contains the true value of  $\beta$ . We focus on 95% confidence intervals in this simulation.

```
R> summarized_results <- sim %>% summarize(
+   list(stat="mean", name="mean_se_beta0", x="beta0_se_est"),
+   list(stat="mean", name="mean_se_beta1", x="beta1_se_est"),
+   list(stat="coverage", name="cov_beta0", estimate="beta0_est",
+     se="beta0_se_est", truth=-1),
+   list(stat="coverage", name="cov_beta1", estimate="beta1_est",
+     se="beta1_se_est", truth=10)
+ )
R> print(summarized_results)
```

	level_id	estimator	n	n_reps	mean_se_beta0	mean_se_beta1	cov_beta0
1	1	model_vcov	50	500	0.18100830	0.18233202	0.946
2	2	sandwich_vcov	50	500	0.18235693	0.24320294	0.938
3	3	model_vcov	100	500	0.12724902	0.12778192	0.946
4	4	sandwich_vcov	100	500	0.12805679	0.17341936	0.942
5	5	model_vcov	500	500	0.05704424	0.05697317	0.946

6	6	sandwich_vcov	500	500	0.05748439	0.07960900	0.948
7	7	model_vcov	1000	500	0.04055718	0.04059151	0.960
8	8	sandwich_vcov	1000	500	0.04052126	0.05721446	0.958
cov_beta1							
1					0.848		
2					0.922		
3					0.862		
4					0.946		
5					0.836		
6					0.940		
7					0.860		
8					0.958		

To visualize the results, we set up a plotting function.

```
R> library(dplyr)
R> library(tidyr)
R> plot_results <- function(summarized_results, which_graph, n_est) {
+   if (n_est == 3) {
+     values <- c("#999999", "#E69F00", "#56B4E9")
+     breaks <- c("model_vcov", "sandwich_vcov", "bootstrap_vcov")
+     labels <- c("Model-based", "Sandwich", "Bootstrap")
+   } else {
+     values <- c("#999999", "#E69F00")
+     breaks <- c("model_vcov", "sandwich_vcov")
+     labels <- c("Model-based", "Sandwich")
+   }
+   if (which_graph == "width") {
+     summarized_results %>%
+     pivot_longer(
+       cols = c("mean_se_beta0", "mean_se_beta1"),
+       names_to = "parameter",
+       names_prefix = "mean_se_"
+     ) %>%
+     mutate(value_j = jitter(value, amount = 0.01)) %>%
+     ggplot(aes(x=n, y=1.96*value_j, color=estimator)) +
+     geom_line(aes(linetype=parameter)) +
+     geom_point() +
+     theme_bw() +
+     ylab("Average CI width") +
+     xlab("Sample size") +
+     scale_color_manual(
+       values = values,
+       breaks = breaks,
+       name = "SE estimator",
+       labels = labels
+     ) +
+     scale_linetype_discrete(
+       breaks = c("beta0", "beta1"),
+       name = "Parameter",
+       labels = c(expression(beta[0]), expression(beta[1]))
+     )
+   } else {
+     summarized_results %>%
+     pivot_longer(
+       cols = c("cov_beta0", "cov_beta1"),
```

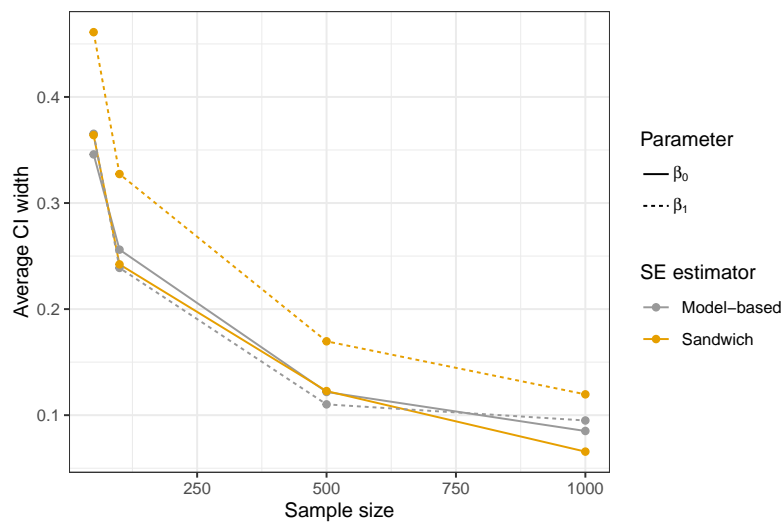
```

+   names_to = "parameter",
+   names_prefix = "cov_"
+ ) %>%
+ mutate(value_j = jitter(value, amount = 0.01)) %>%
+ ggplot(aes(x=n, y=value, color=estimator)) +
+   geom_line(aes(linetype = parameter)) +
+   geom_point() +
+   theme_bw() +
+   ylab("Coverage") +
+   xlab("Sample size") +
+   scale_color_manual(
+     values = values,
+     breaks = breaks,
+     name = "SE estimator",
+     labels = labels
+   ) +
+   scale_linetype_discrete(
+     breaks = c("beta0", "beta1"),
+     name = "Parameter",
+     labels = c(expression(beta[0]), expression(beta[1]))
+   ) +
+   geom_hline(yintercept=0.95)
+ }
+ }

```

We then use the plotting function to make figures.

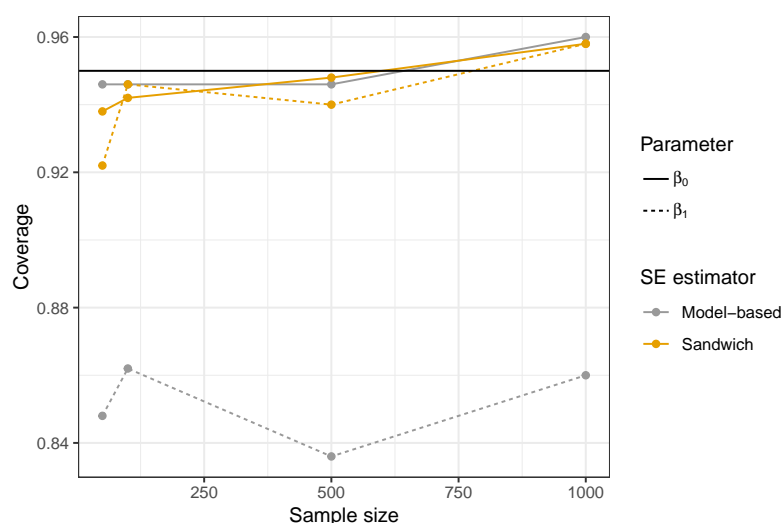
```
R> plot_results(summarized_results, "width", 2)
```



```
R> plot_results(summarized_results, "coverage", 2)
```

Looking at these plots, we see that the sandwich method results in a wider interval, on average, for  $\beta_1$ . In terms of coverage, the sandwich estimator achieves near nominal coverage for both parameters, while there is moderate undercoverage for  $\beta_1$  using the model-based estimator.

The bootstrap is another popular approach to estimating standard errors (Efron, 1979). We can add a bootstrap method and use `update_sim` to run the new simulation replicates without re-running any of the previous work. We do this by including the new estimator



in the simulation levels. Since the bootstrap can be computationally intensive, we use parallelization. This requires specifying the option 'parallel = TRUE' in `set_config`.

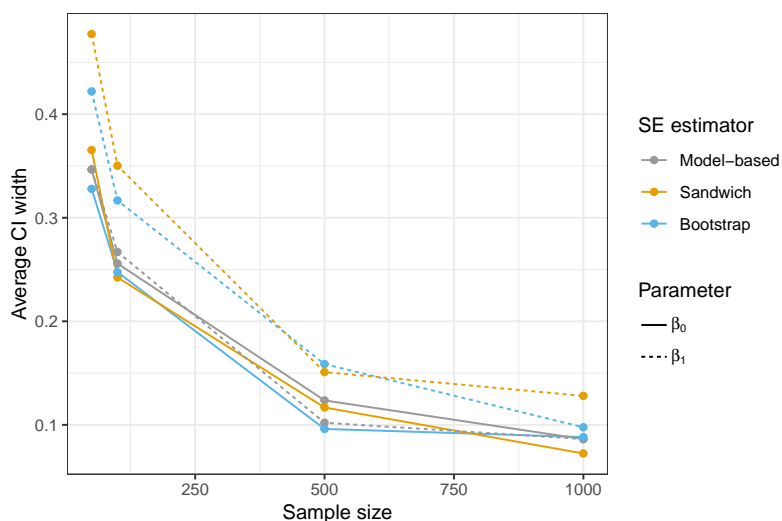
```
R> bootstrap_vcov <- function(data) {
+   mod <- lm(y~x, data=data)
+   boot_ests <- matrix(NA, nrow=100, ncol=2)
+   for (j in 1:100) {
+     indices <- sample(1:nrow(data), size=nrow(data), replace=TRUE)
+     boot_dat <- data[indices,]
+     boot_mod <- lm(y~x, data=boot_dat)
+     boot_ests[j,] <- boot_mod$coefficients
+   }
+   boot_v1 <- var(boot_ests[,1])
+   boot_v2 <- var(boot_ests[,2])
+   return(list("coef"=mod$coefficients, "vcov"=c(boot_v1, boot_v2)))
+ }
R> sim %<>% set_levels(
+   estimator = c("model_vcov", "sandwich_vcov", "bootstrap_vcov"),
+   n = c(50, 100, 500, 1000)
+ )
R> sim %<>% set_config(
+   num_sim = 500,
+   seed = 24,
+   parallel = TRUE,
+   n_cores = 2,
+   packages = c("sandwich")
+ )
R> sim %<>% update_sim()
Done. No errors or warnings detected.
```

Now that the bootstrap results are included in the simulation object, we can look at the updated results.

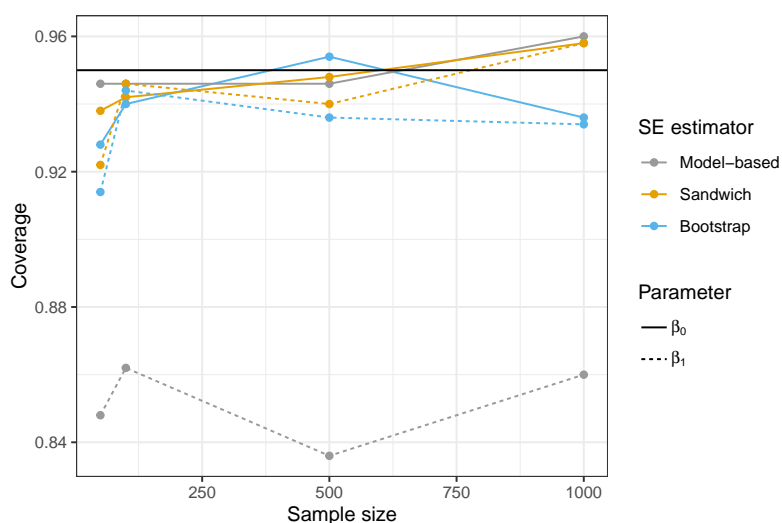
```
R> summarized_results <- sim %>% summarize(
+   list(stat="mean", name="mean_se_beta0", x="beta0_se_est"),
+   list(stat="mean", name="mean_se_beta1", x="beta1_se_est"),
+   list(stat="coverage", name="cov_beta0", estimate="beta0_est",
+     se="beta0_se_est", truth=-1),
+   list(stat="coverage", name="cov_beta1", estimate="beta1_est",
```



```
+       se="beta1_se_est", truth=10)
+ )
R> plot_results(summarized_results, "width", 3)
```



```
R> plot_results(summarized_results, "coverage", 3)
```



Like the sandwich estimator, the bootstrap results in wider intervals for  $\beta_1$ , but is much closer to achieving 95% coverage compared to the model-based estimator.

## Bibliography

- B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. doi: 10.1214/aos/1176344552. URL <https://doi.org/10.1214/aos/1176344552>. [p7]
- H. White. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: journal of the Econometric Society*, pages 817–838, 1980. [p4]
- A. Zeileis. Econometric computing with hc and hac covariance matrix estimators. *Journal of Statistical Software*, 2004. [p4]

A. Zeileis, S. Köll, and N. Graham. Various versatile variances: an object-oriented implementation of clustered covariances in r. *Journal of Statistical Software*, 95:1–36, 2020. [p4]