Introduction to simulation studies

Nora Wickelmaier[1]

Last modified: 2025-09-15

[1]Parts of this slide set are a (modified) version of slides accompanying the book by Strobl, Henninger, Rothacher, and Debelak (2024)

What are simulation studies used for?

- Examining the properties of statistical methods
- Various applications of simulation studies
  - Illustration or didactic explanation of properties of statistical methods (e. g. with Shiny Apps)
  - Investigation of newly developed statistical methods for which theoretical properties are not yet known
  - Investigation of properties of statistical methods that only hold asymptotically for realistic sample sizes
  - Investigation of the impact of violated assumptions on statistical methods
  - Power analyses for sample size planning
- All these applications of simulation studies have as a common core that a simulation study represents an – ideally well-planned – experiment

Strobl et al. (2024)

Advantages of using simulated data

- All properties of the data sets can be controlled
- Extreme scenarios can be examined, which are rare in real data
- Forces us to define (and think about) data generating process
- Data simulation (repeated sampling) is at the heart of hypothesis testing in the frequentist framework

Example simple data simulation

- Let us look at a simple example and remind ourselves of the underlying concept of hypothesis testing after Neyman and Pearson (1933)
- We fit a simple regression line to predict stopping distance (in feet) of oldtimer cars with driving speed (in miles per hour)
- We fit a null model that only predicts the mean distance

$$y = \beta_0 + \varepsilon, \quad \text{with } \varepsilon \sim N(0, \sigma_\varepsilon^2)$$

  an a model that predicts distance with speed

$$y = \beta_0 + \beta_1 \cdot x + \varepsilon, \quad \text{with } \varepsilon \sim N(0, \sigma_\varepsilon^2)$$

- We want to simulate the type I error rate for fitting a model including the slope β_1 when data were generated from the null model

Type I error rate

- The type I error rate corresponds to how often a test obtains a significant result for a given α level (e. g. 5%), given that no effect is truly present

$$\widehat{Type\text{-}I\text{-}Err}_s = \frac{\sum_{i=1}^{niter} I(p\text{-}value_{is} < \alpha | H_0)}{niter}$$

- For a test with a given α level, the empirical type I error rate should be close to α
- A test is called conservative if the observed type I error rate is lower than the given α
- A test is called liberal if the observed type I error rate exceeds the given α

Demonstration

Hypothesis testing after Neyman and Pearson

```
lm0 <- lm(dist ~ 1, cars)          # H0
lm1 <- lm(dist ~ speed, cars)      # H1

nsim <- 1000
pval <- numeric(nsim)

for (i in 1:nsim) {
  sim <- simulate(lm0)$sim_1
  fit <- lm(sim ~ speed, cars)
  pval[i] <- summary(fit)$coef["speed", "Pr(>|t|)"]
}

# Type I error
mean(pval < 0.05)
```

Structure of simulation studies

- Strobl et al. (2024) recommend dividing code for simulation studies into three functions
  1. `dgp` = data generating process
  2. `one_simulation` = a single simulation run
  3. `simulation_study` = complete simulation design
- This modular structure allows to easily extend the code and use it for different tasks
- Functions allow to combine individual work steps (promotes simplicity and clarity of the code)
- Functions are useful if the same work step is to be executed multiple times (e. g. data generation)
- Set of functions as a "modular system"

Example of a simulation study

- We are interested in estimating the slope coefficient β_1 in the simple linear regression model
- Possible questions we could ask are:
  1. How much do the estimated slope coefficients deviate from the true slope coefficient for a given sample size?
  2. How does the accuracy of the estimation change when we alter the sample size?
- The model equation for a person p, with $p = 1, \ldots, npers$, is:

$$y_p = \beta_0 + \beta_1 \cdot x_p + \varepsilon_p, \text{ where } \varepsilon_p \sim N(0, \sigma_\varepsilon^2)$$

- The model equation serves as a "recipe" for simulating data

Data generating process

To generate data from the simple linear regression model, we need to specify:

- The parameters β_0 and β_1
- The variance σ_ε^2 or standard deviation σ_ε of the error distribution
- The distribution from which we draw the x values (commonly: uniform distribution or normal distribution)
- The sample size *npers*

```r
npers <- 100
s_err <- 5
beta  <- c(1.5, 2.5)
```

Data generating process

- Generating the random errors
  - The simple linear regression model assumes $\varepsilon_p \sim N(0, \sigma_\varepsilon^2)$
  - Random, normally distributed values can be generated in R using `rnorm()`

```
err <- rnorm(n = npers, mean = 0, sd = s_err)
```

- Generating the x and y values

```
x <- runif(n = npers, min = 0, max = 5)
y <- beta[1] + beta[2] * x + err
```

- Saving as a data frame

```
dat <- data.frame(x = x, y = y)
```

Data generating process

- Model estimation to control data generation

```
model <- lm(y ~ x, data = dat)
summary(model)
coef(model)
```

- We will now combine these steps into a function

```
dgp <- function(npers, beta, s_err){
  x <- runif(n = npers, min = 0, max = 5)
  err <- rnorm(n = npers, mean = 0, sd = s_err)
  y <- beta[1] + beta[2] * x + err
  dat <- data.frame(x = x, y = y)
  return(dat)
}
```

Data generation with the dgp function

- If we execute the dgp function multiple times, different datasets are generated

```r
dat <- dgp(npers = 100,
           beta = c(1.5, 2.5),
           s_err = 5)
head(dat)

#           x         y
# 1 3.0105702 12.444386
# 2 0.9752196  9.033958
# 3 4.8322937 12.378318
# 4 3.2545276 18.024507
# 5 1.8353595  8.134339
# 6 4.9442961  9.418030
```

```r
dat <- dgp(npers = 100,
           beta = c(1.5, 2.5),
           s_err = 5)
head(dat)

#          x         y
# 1 2.081702 10.797706
# 2 1.272501  4.375452
# 3 1.508982  7.909477
# 4 4.474560 12.664724
# 5 3.103152 10.400254
# 6 4.588655  7.442873
```

A single simulation run

- We then create a function to run a single run of our simulation
- We want to simulate the estimated slope parameter β_1

```
one_simulation <- function(npers, beta, s_err){
  dat <- dgp(npers = npers, beta = beta, s_err = s_err)
  model <- lm(y ~ x, data = dat)
  slope_est <- coef(model)[2]
  return(slope_est)
}

# Run one simulation
one_simulation(npers = 100, beta = c(1.5, 2.5), s_err = 5)
```

500 simulation runs

- We can now use a for-loop to repeat our data simulation

```
niter <- 500
slope_est <- rep(NA, niter)
for(i in 1:niter) {
  slope_est[i] <- one_simulation(npers = 100,
                                 beta = c(1.5, 2.5),
                                 s_err = 5)
}
```

- As expected, the estimated slopes scatter around the specified true value

```
boxplot(slope_est)
abline(h = 2.5, lty = 2)
```

Complete simulation design

- The last step is to write a function for a complete simulation study design
- The factors that we want to investigate (e. g. different sample sizes, different variation in the data) are arguments for this function

```
simulation_study <- function(niter, npers, beta, s_err){
  prs <- expand.grid(i = 1:niter, npers = npers, s_err = s_err)
  slope_est <- rep(NA, nrow(prs))
  for(i in 1:nrow(prs)) {
    slope_est[i] <- one_simulation(npers = prs$npers[i],
                                   beta = beta,
                                   s_err = prs$s_err[i])
  }
  return(cbind(prs, slope_est))
}
```

The `expand.grid` function

```
niter <- 2
factor_1 <- c("a", "b")
factor_2 <- c(5, 10)
expand.grid(i = 1:niter, variant = factor_1, quantity = factor_2)
#   i variant quantity
# 1 1       a        5
# 2 2       a        5
# 3 1       b        5
# 4 2       b        5
# 5 1       a       10
# 6 2       a       10
# 7 1       b       10
# 8 2       b       10
```

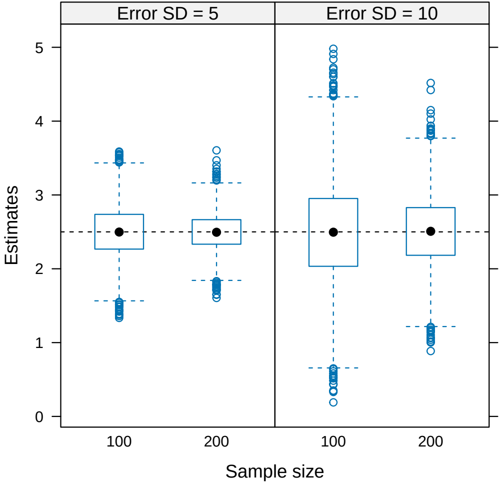
Conducting the simulation

- For each combination of factor levels of `npers` and `s_err` we perform 500 simulation runs ($500 \times 2 \times 2 = 2000$ runs)

```
sim_results <- simulation_study(niter = 500,
                                npers = c(100, 200),
                                beta = c(1.5, 2.5),
                                s_err = c(5, 10))
head(sim_results)
#   i npers s_err slope_est
# 1 1   100     5  2.552509
# 2 2   100     5  2.477369
# 3 3   100     5  3.509792
# 4 4   100     5  3.464987
# 5 5   100     5  3.154455
# 6 6   100     5  3.027992
```

Graphical presentation of results

Distribution of estimated slope coefficients around the true value 2.5




```
library("lattice")
bwplot(slope_est ~ npers | s_err,
       data = sim_results,
       xlab = "Sample size",
       ylab = "Estimates")
```

References

Neyman, J., & Pearson, E. S. (1933). On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London, Series A*, *231*(694–706), 289–337.

Strobl, C., Henninger, M., Rothacher, Y., & Debelak, R. (2024). *Simulationsstudien in R: Design und praktische Durchführung*. Springer Berlin.