Causal Inference with Continuous Exposures: A Tutorial with Application to ICU Data: Simulation

M Ehsan Karim

1 Simulate Data for Analysis

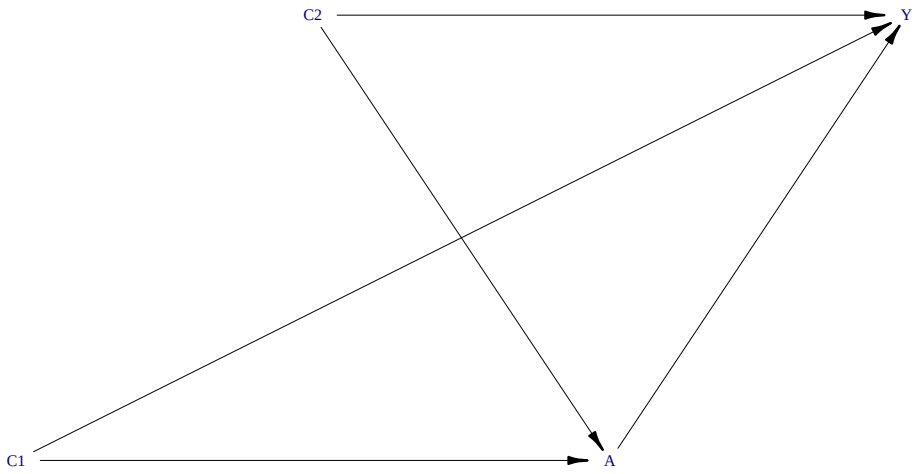
We begin by simulating data with known causal relationships. This allows us to assess how well our methods estimate the true effects. We use the `simcausal` package, which defines data generation based on a Directed Acyclic Graph (DAG). We simulate a dataset where:

- `A` is the continuous exposure, generated as a function of confounders `C1`, `C2`
- `Y` is the binary outcome, generated as a function of `A`, `C1`, and `C2`
- `C1`, `C2` are measured confounders

The true conditional effect of A on Y (log-odds ratio) is set to 0.4 in the simulation.

```
library(simcausal)
set.seed(123)
D <- DAG.empty() +
  node("C1", distr = "rnorm", mean = 0, sd = 1) +
  node("C2", distr = "rbern", prob = 0.5) +
  node("A", distr = "rnorm", mean = 0.5 * C1 + 0.3 * C2, sd = exp(.8*C1)) +
  # Exposure depends on confounders (Heteroscedastic)
  # alternative could be to simulate from a gamma distribution:
  # node("A", distr = "rgamma", shape = 2 + C1, scale = 1) +
  node("Y", distr = "rbern", prob = plogis(0.4 * A + 0.5 * C1 - 0.3 * C2))
  # Known true effect of A on Y
Dset <- set.DAG(D, n.test = 1e6)

# Plot DAG
plotDAG(Dset)
```

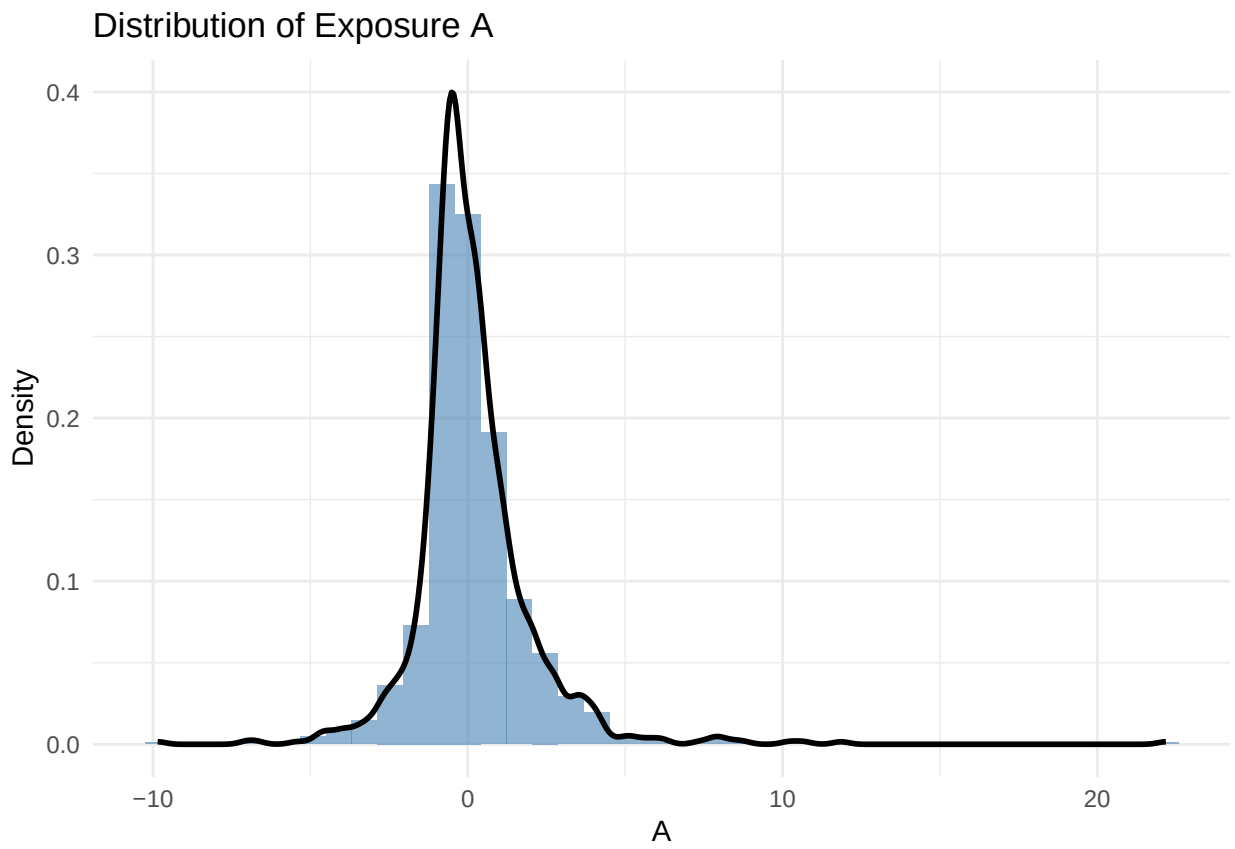



```r
# Simulate data
simdata <- sim(Dset, n = 1000)
```

Let's visualize the distribution of our continuous exposure A.

```r
library(ggplot2)

# Histogram with density overlay
ggplot(simdata, aes(x = A)) +
  geom_histogram(aes(y = ..density..), bins = 40, fill = "steelblue", alpha = 0.6) +
  geom_density(color = "black", linewidth = 1) +
  labs(title = "Distribution of Exposure A", x = "A", y = "Density") +
  theme_minimal()
```



1.1 Method 1: Normal Model for Exposure

This method assumes that the exposure is normally distributed with constant variance (homoscedastic).

- Fit a marginal model for numerator (A ~ 1)
- Fit a conditional model for denominator (A ~ C1 + C2)
- Use normal density to compute stabilized weights

```
# Marginal model
marg_model <- lm(A ~ 1, data = simdata)
mu_num <- predict(marg_model)
sd_num <- sd(residuals(marg_model))

# Conditional model
cond_model <- lm(A ~ C1 + C2, data = simdata)
mu_denom <- predict(cond_model)
sd_denom <- sd(residuals(cond_model))

# Compute densities
f_num <- dnorm(simdata$A, mean = mu_num, sd = sd_num)
f_denom <- dnorm(simdata$A, mean = mu_denom, sd = sd_denom)

# Compute stabilized weights
simdata$sw_normal <- f_num / f_denom
```

1.2 Method 2: Quantile Binning Approach

This approach does not assume a distributional form and is robust to outliers and heteroscedasticity.

- Discretize the exposure into quantiles
- Estimate conditional probability of each quantile using a multinomial logistic model
- Use inverse of predicted conditional probabilities to compute weights

```
# Categorize A into 10 quantiles
simdata$qbin <- cut(simdata$A,
                    breaks = quantile(simdata$A,
                                      probs = seq(0, 1, 0.1)),
                    include.lowest = TRUE, labels = FALSE)

# Fit multinomial logistic regression for P(qbin | C)
library(nnet)
denom_model <- multinom(factor(qbin) ~ C1 + C2,
                        data = simdata, trace = FALSE)
p_denom <- predict(denom_model, type = "probs")
row_idx <- cbind(1:nrow(simdata), simdata$qbin)
p_denom_val <- p_denom[row_idx]

# Numerator is uniform across quantiles
p_num_val <- 1 / 10

# Stabilized weights
simdata$sw_qbin <- p_num_val / p_denom_val
```

1.3 Weighted Outcome Model

With IPW weights, we estimate the marginal effect of A on Y using a weighted logistic regression model: $logit(P(Y = 1)) = \beta_0 + \beta_A A$. The weights create a pseudo-population where confounding between A and (C_1, C_2) is removed. The coefficient β_A is our estimate of the marginal log-odds ratio.

```
# Weighted outcome model using logistic regression
model_normal <- glm(Y ~ A,
                    family = binomial(),
                    data = simdata,
                    weights = sw_normal)
model_qbin <- glm(Y ~ A,
                  family = binomial(),
                  data = simdata,
                  weights = sw_qbin)

# Summary of models
require(Publish)
publish(model_normal)
```

```
##  Variable Units OddsRatio       CI.95 p-value
##         A            1.43 [1.36;1.51] < 1e-04
```

```r
publish(model_qbin)
```

```
##  Variable Units OddsRatio       CI.95 p-value
##         A            1.92 [1.71;2.17] < 1e-04
```

1.4 Method 3: TMLE

TMLE is a doubly robust, semi-parametric method. Doubly robust means it provides unbiased estimates if either the outcome model $E[Y|A,W]$ or the exposure mechanism model $P(A|W)$ is correctly specified. We use the `tmle3` package suite. TMLE involves:

1. Initial estimation of the outcome mechanism ($Q(A,W)$) and exposure mechanism ($g(A|W)$).
2. A "targeting" step that updates $Q(A,W)$ to optimize the bias-variance tradeoff for the parameter of interest.

We'll estimate the effect of a small shift, $\delta = 0.1$, in exposure A. The parameter of interest from tmle_shift with a binary outcome is often related to a coefficient in a marginal structural model like $logit P(Y(a)) = \alpha_0 + \alpha_1 a$. The raw TMLE estimate (psi) will be for the effect of this δ shift (i.e., $\alpha_1 \times \delta$). To get α_1 (the log-odds ratio per 1-unit increase in A), we divide psi by δ.

```r
{
  library(tmle3)
  library(tmle3shift)
  library(sl3)
  library(data.table)

  # Define nodes
  node_list <- list(
    W = c("C1", "C2"),
    A = "A",
    Y = "Y"
  )

  # Define trained learners (scoped inside this block)
  glm_learner <- make_learner(Lrnr_glm)
  learner_list <- list(
    Y = glm_learner,
    A = glm_learner
  )

  # Shift function using tmle_task
  tmle_spec <- tmle_shift(
    delta = 0.1,
    shift_fxn = function(tmle_task, delta, ...) {
      a <- tmle_task$get_tmle_node("A")
      a + delta
    },
    max_shift = 1
  )

  future::plan(future::sequential)
```

```
  # Run TMLE
  tmle_fit <- tmle3(
    tmle_spec,
    data = as.data.table(simdata),
    node_list = node_list,
    learner_list = learner_list
  )

  # Print estimate
  print(exp(tmle_fit$estimates[[1]]$psi))
}
```

```
## [1] 1.761697
```

```
# Extract psi and IC
psi <- tmle_fit$estimates[[1]]$psi
IC <- tmle_fit$estimates[[1]]$IC

# Compute standard error
se <- sd(IC) / sqrt(length(IC))

# 95% confidence interval
ci <- psi + c(-1.96, 1.96) * se

# Show summary
data.frame(
  logOR = psi,
  OR = exp(psi),
  se = se,
  lower = exp(ci[1]),
  upper = exp(ci[2])
)
```

```
##       logOR       OR         se    lower    upper
## 1 0.5662776 1.761697 0.01281936 1.717984 1.806522
```

2 Simulation

2.1 True Marginal Effect (for comparison)

The parameter 0.4 in our data generating process is the conditional log-odds ratio for A. IPW aims to estimate the marginal log-odds ratio. These can differ in logistic models. To obtain a "true" marginal logOR for our simulation setup, we simulate a very large dataset and fit a simple $Y\tilde{A}$ model. This provides a benchmark for our IPW estimates. For the simulation performance assessment later, we will use the conditional parameter 0.4 as the "true value" for simplicity.

```
# Estimate marginal OR (unadjusted)
set.seed(123)
data_big <- sim(Dset, n = 1e6)
model_marginal <- glm(Y ~ A, family = binomial(), data = data_big)
exp(coef(model_marginal)["A"])  # This is the true marginal OR
```

```
##        A
## 1.590596
```

2.2 Monte Carlo Simulation

To assess the average performance and variability of our estimators, we conduct a simulation study: repeat the data generation and estimation process many times (e.g., 1000) and analyze the distribution of the estimates. We use the **rsimsum** package to compare performance (Bias, MSE, Coverage). The true value for comparison is the logOR of 0.4.

```r
library(simcausal)
library(tibble)
library(nnet)
library(furrr)
library(future)
plan(multisession)

nsim <- 1000

run_sim <- function(i) {
  simdata <- sim(Dset, n = 1000)

  # Normal model
  marg_model <- lm(A ~ 1, data = simdata)
  mu_num <- predict(marg_model)
  sd_num <- sd(residuals(marg_model))

  cond_model <- lm(A ~ C1 + C2, data = simdata)
  mu_denom <- predict(cond_model)
  sd_denom <- sd(residuals(cond_model))

  f_num <- dnorm(simdata$A, mean = mu_num, sd = sd_num)
  f_denom <- dnorm(simdata$A, mean = mu_denom, sd = sd_denom)
  sw_normal <- f_num / f_denom

  # Quantile binning
  simdata$qbin <- cut(simdata$A, breaks = quantile(simdata$A, probs = seq(0, 1, 0.1)),
                      include.lowest = TRUE, labels = FALSE)
  denom_model <- nnet::multinom(factor(qbin) ~ C1 + C2, data = simdata, trace = FALSE)
  p_denom <- predict(denom_model, type = "probs")
  row_idx <- cbind(1:nrow(simdata), simdata$qbin)
  p_denom_val <- p_denom[row_idx]
  p_num_val <- 1 / 10
  sw_qbin <- p_num_val / p_denom_val

  # Outcome models
  model_normal <- glm(Y ~ A, family = binomial(), data = simdata, weights = sw_normal)
  model_qbin <- glm(Y ~ A, family = binomial(), data = simdata, weights = sw_qbin)

  tibble(
  normal = coef(model_normal)["A"],
  se_normal = sqrt(diag(vcov(model_normal))["A"]),
  qbin = coef(model_qbin)["A"],
```

```
  se_qbin = sqrt(diag(vcov(model_qbin))["A"])
  )
}

results <- future_map_dfr(1:nsim, run_sim,
                          .progress = TRUE,
                          .options = furrr_options(seed = TRUE))

print(dim(results))
print(head(results))
saveRDS(results, "results.rds")
```

To simulate for TMLE, we follow the exact same process as above:

```
run_sim <- function(i) {
  library(simcausal)
  library(sl3)
  library(tmle3)
  library(tmle3shift)
  library(data.table)
  library(tibble)
  simdata <- sim(Dset, n = 1000)

  # TMLE estimation block
  node_list <- list(W = c("C1", "C2"), A = "A", Y = "Y")

  glm_learner <- make_learner(Lrnr_glm)
  learner_list <- list(Y = glm_learner, A = glm_learner)

  tmle_spec <- tmle_shift(
    delta = 0.1,
    shift_fxn = function(tmle_task, delta, ...) {
      a <- tmle_task$get_tmle_node("A")
      a + delta
    },
    max_shift = 1
  )

  tmle_fit <- tmle3(
    tmle_spec,
    data = as.data.table(simdata),
    node_list = node_list,
    learner_list = learner_list
  )

  # Extract and return estimate + SE
  psi <- tmle_fit$estimates[[1]]$psi
  IC <- tmle_fit$estimates[[1]]$IC
  se <- sd(IC) / sqrt(length(IC))

  return(tibble(
    logOR_tmle = psi,
    se_tmle = se
```

```
  ))
}

library(furrr)
plan(multisession)
results_tmle <- future_map_dfr(1:nsim, run_sim, .options = furrr_options(seed = TRUE))
str(results_tmle)
saveRDS(results_tmle, "results_tmle.rds")
```

2.3 Detailed simulation results

```
results <- readRDS("results.rds")
results_tmle <- readRDS("results_tmle.rds")
library(rsimsum)
# Combine all estimates and SEs into one data.frame for rsimsum
results_long <- data.frame(
  sim = rep(1:nrow(results), times = 3),
  method = rep(c("normal", "qbin", "tmle"), each = nrow(results)),
  theta = c(results$normal, results$qbin, results_tmle$logOR_tmle),
  se = c(results$se_normal, results$se_qbin, results_tmle$se_tmle),
  true = 0.4
)

sumobj <- simsum(
  data = results_long,
  estvarname = "theta",
  se = "se",
  true = 0.4,
  methodvar = "method"
)

ss1 <- summary(sumobj)

results_long$method <- factor(results_long$method,
  levels = c("normal", "qbin", "tmle"),
  labels = c("Normal Model", "Quantile Binning", "TMLE")
)


library(dplyr)
library(tidyr)
library(knitr)
library(kableExtra) # Optional, for nicer table styling

stat_map <- list(
  "thetamean"     = list(label = "Average Estimate", is_percent = FALSE, est_digits = 4, mcse_digits = 4
  "bias"          = list(label = "Bias", is_percent = FALSE, est_digits = 4, mcse_digits = 4),
  "empse"         = list(label = "Empirical SE", is_percent = FALSE, est_digits = 4, mcse_digits = 4),
  "modelse"       = list(label = "Model SE (avg)", is_percent = FALSE, est_digits = 4, mcse_digits = 4)
  "mse"           = list(label = "MSE", is_percent = FALSE, est_digits = 4, mcse_digits = 4),
  "becover"       = list(label = "Bias-Eliminated Coverage (95% CI)", is_percent = FALSE, est_digits = 4
```

```r
  "relerror"      = list(label = "Rel Error in SE (%)", is_percent = TRUE, est_digits = 2, mcse_digits =
)
desired_stats_internal_names <- names(stat_map)

if (exists("ss1") && "summ" %in% names(ss1) && is.data.frame(ss1$summ)) {
  summary_filtered <- ss1$summ %>%
    filter(stat %in% desired_stats_internal_names) %>%
    # Apply row-wise operations
    rowwise() %>%
    mutate(
      is_extreme = (method == "normal" & stat %in% c("thetamean", "bias", "empse", "mse") & (abs(est) >

      display_value = {
        # Get info for the CURRENT row's stat
        s_info <- stat_map[[stat]] # 'stat' here will be the value for the current row

        # Default to N/A if s_info is somehow NULL (should not happen if filter worked)
        if (is.null(s_info)) {
          "N/A"
        } else if (is_extreme) {
          if (is.infinite(est) | is.nan(est)) "Failed/Extreme"
          else if (est > 0) ">10^6"
          else if (est < 0) "<-10^6"
          else as.character(est)
        } else if (is.infinite(est) | is.nan(est)) {
          "N/A (Inf/NaN)"
        }
        else {
          est_formatted <- sprintf(paste0("%.", s_info$est_digits, "f"), est)
          val_to_return <- if (s_info$is_percent) paste0(est_formatted, "%") else est_formatted

          if (!is.na(mcse) && !(is.infinite(mcse) | is.nan(mcse))) {
            mcse_formatted <- sprintf(paste0("%.", s_info$mcse_digits, "f"), mcse)
            val_to_return <- if (s_info$is_percent) {
                               paste0(est_formatted, "% (", mcse_formatted, "%)")
                             } else {
                               paste0(est_formatted, " (", mcse_formatted, ")")
                             }
          }
          val_to_return
        }
      },
      # Map internal stat names to pretty display labels
      Statistic = factor(stat, levels = desired_stats_internal_names, labels = sapply(stat_map, functio
    ) %>%
    ungroup() %>%
    select(Statistic, method, display_value)

  wide_table <- summary_filtered %>%
    pivot_wider(
      names_from = method,
      values_from = display_value
    )
```

```
  method_levels_from_factor <- levels(ss1$summ$method) # Get original method levels
  desired_cols <- c("Statistic", intersect(method_levels_from_factor, names(wide_table)))
  wide_table <- wide_table %>% select(all_of(desired_cols))


  if (nrow(wide_table) > 0) {
    kable(wide_table, caption = "Simulation Performance Metrics (True Value = 0.4)", align = c('l', rep
      kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                    full_width = FALSE,
                    font_size = 10)
  } else {
    print("No data to display after filtering and pivoting. Check 'stat_map' and 'ss1$summ' contents.")
  }

} else {
  print("Error: 'ss1$summ' is not available or not a data frame. Please ensure 'ss1' is a valid rsimsum
}
```

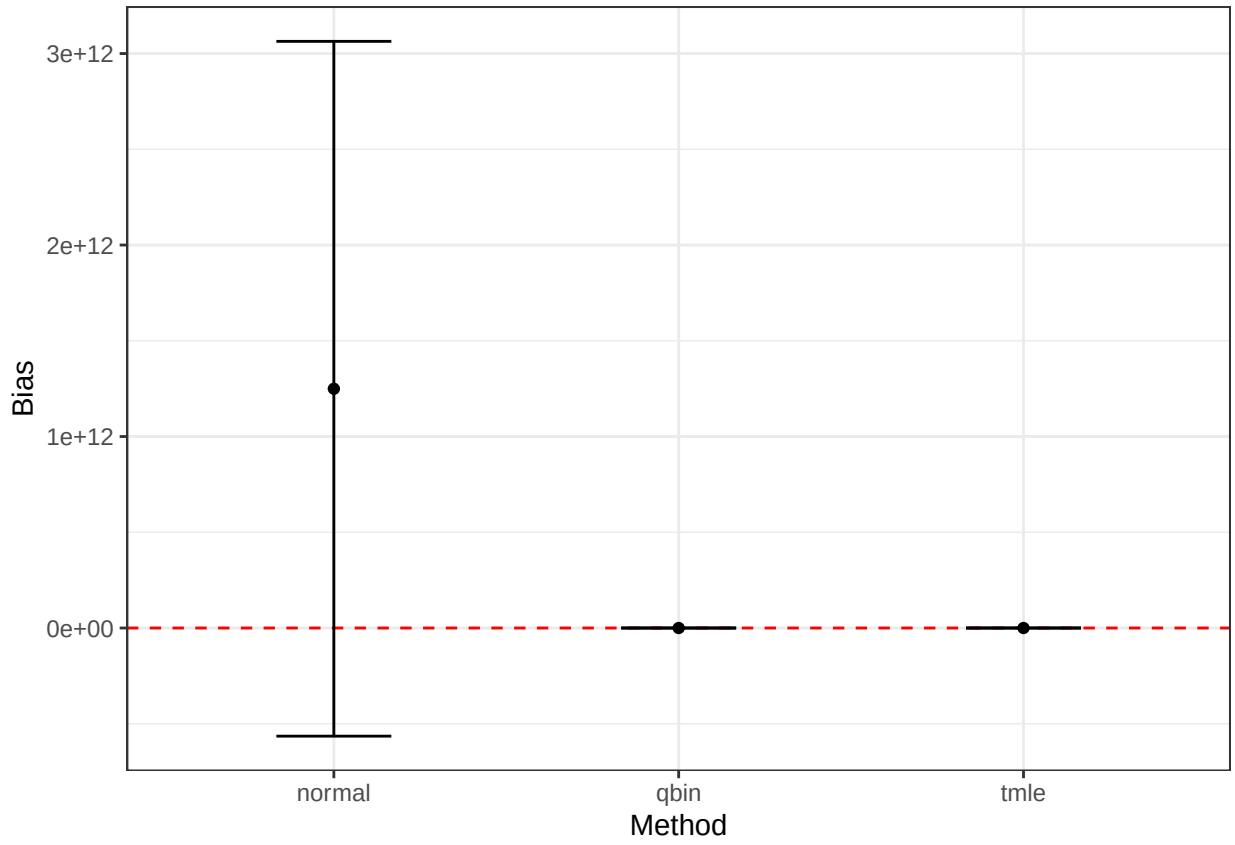
Table 1: Simulation Performance Metrics (True Value = 0.4)

Statistic	normal	qbin	tmle
Average Estimate	>10^6	0.7063	0.5542
Bias	>10^6	0.3063 (0.0137)	0.1542 (0.0006)
Empirical SE	>10^6	0.4326 (0.0097)	0.0197 (0.0004)
MSE	>10^6	0.2808 (0.0158)	0.0242 (0.0002)
Model SE (avg)	76743.8584 (7827.4084)	0.0562 (0.0002)	0.0131 (0.0000)
Rel Error in SE (%)	-100.00% (0.00%)	-87.01% (0.29%)	-33.42% (1.50%)
Bias-Eliminated Coverage (95% CI)	0.0000 (0.0000)	0.2410 (0.0135)	0.8150 (0.0123)

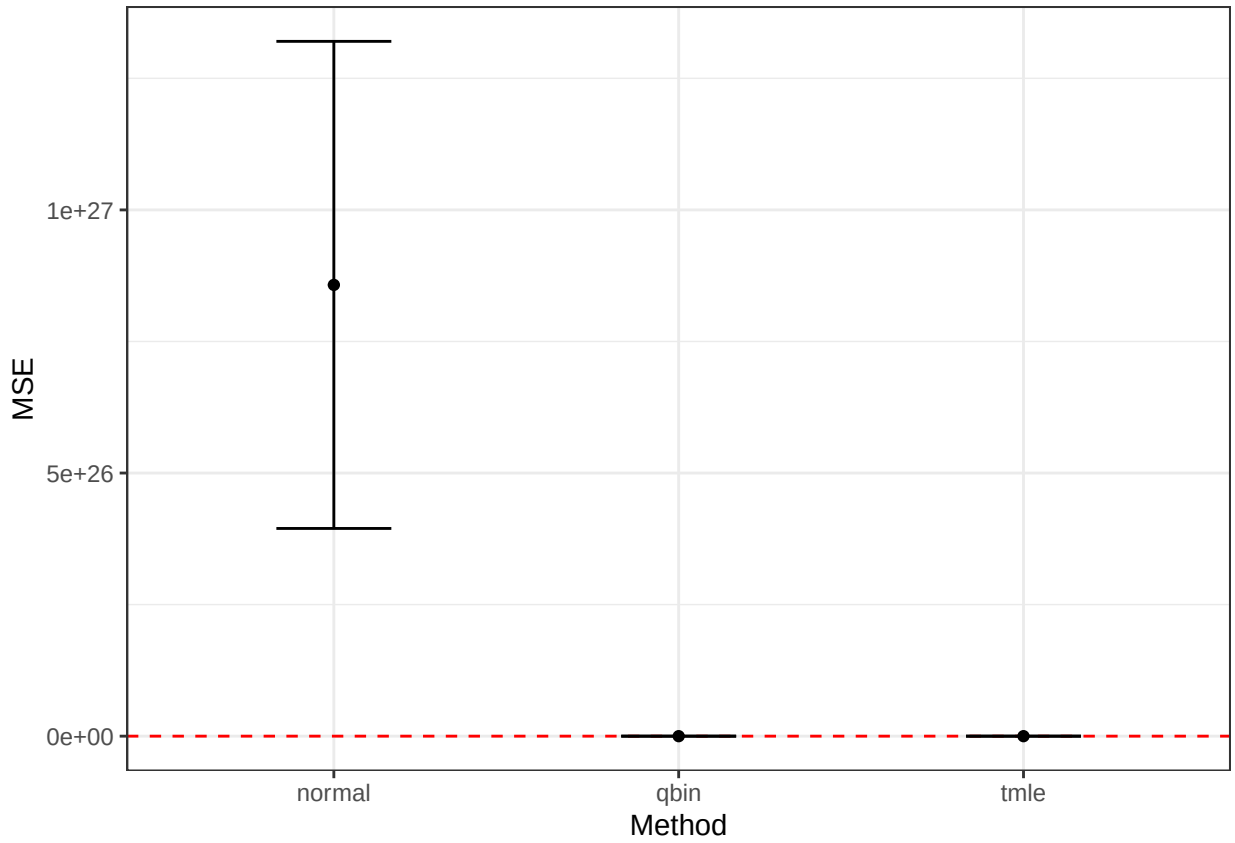
2.4 Visualizing Simulation Performance

Plots help compare estimator performance.

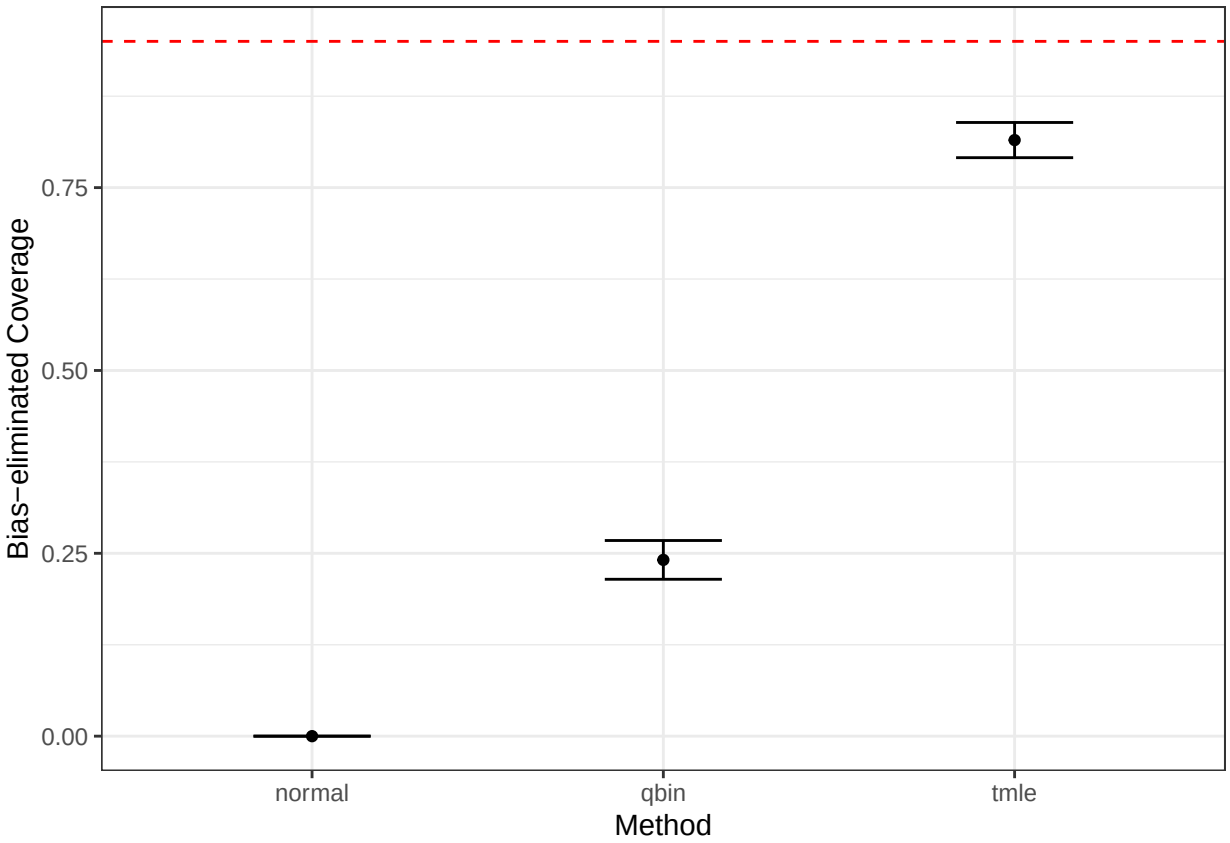
```
# Plot bias, MSE, coverage, MCSE
ggplot(tidy(ss1, stats = "bias"), aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_hline(yintercept = 0, color = "red", lty = "dashed") +
  geom_point() +
  geom_errorbar(width = 1 / 3) +
  theme_bw() +
  labs(x = "Method", y = "Bias")
```



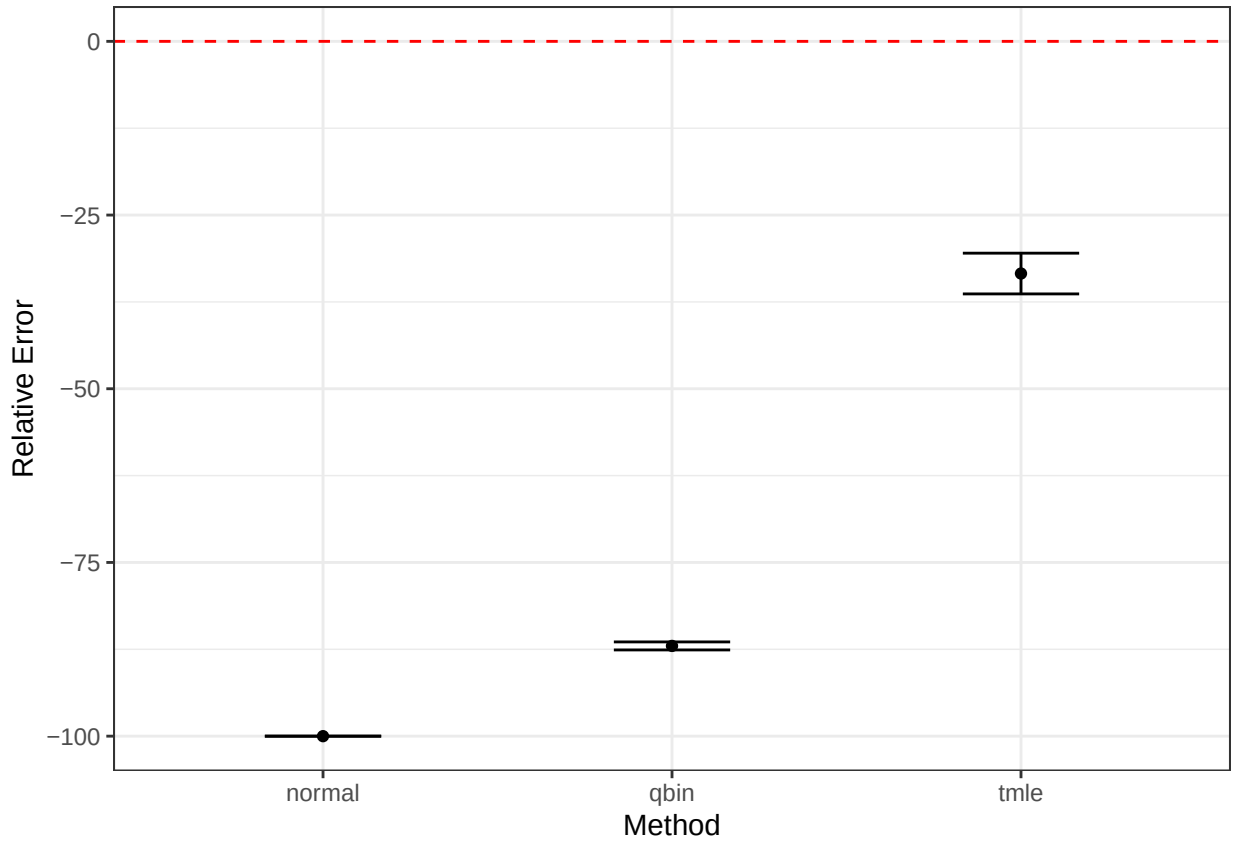

```
ggplot(tidy(ss1, stats = "mse"), aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_hline(yintercept = 0, color = "red", lty = "dashed") +
  geom_point() +
  geom_errorbar(width = 1 / 3) +
  theme_bw() +
  labs(x = "Method", y = "MSE")
```




```
ggplot(tidy(ss1, stats = "becover"), aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_hline(yintercept = 0.95, color = "red", lty = "dashed") +
  geom_point() +
  geom_errorbar(width = 1 / 3) +
  theme_bw() +
  labs(x = "Method", y = "Bias-eliminated Coverage")
```




```
ggplot(tidy(ss1, stats = "relerror"), aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_hline(yintercept = 0, color = "red", lty = "dashed") +
  geom_point() +
  geom_errorbar(width = 1 / 3) +
  theme_bw() +
  labs(x = "Method", y = "Relative Error")
```



2.5 Save results

```r
library(ggplot2)
library(patchwork)  # for side-by-side layout
library(rsimsum)

library(forcats)
bias_df <- tidy(ss1, stats = "bias") %>%
  mutate(method = fct_recode(method,
                             "IPW (Normal)" = "normal",
                             "IPW (Quantile Binning)" = "qbin",
                             "TMLE (Shift)" = "tmle"))

# Then use bias_df in the plot
p1 <- ggplot(bias_df, aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_point() +
  geom_errorbar(width = 0.2) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(y = "Bias", x = "") +
  theme_bw(base_size = 12) +
  theme(panel.grid = element_blank(),
        axis.text.x = element_text(angle = 20, hjust = 1))

mse_df <- tidy(ss1, stats = "mse") %>%
```

```
  mutate(method = fct_recode(method,
                             "IPW (Normal)" = "normal",
                             "IPW (Quantile Binning)" = "qbin",
                             "TMLE (Shift)" = "tmle"))

p2 <- ggplot(mse_df, aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_point() +
  geom_errorbar(width = 0.2) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(y = "Mean squared error", x = "") +
  theme_bw(base_size = 12) +
  theme(panel.grid = element_blank(),
        axis.text.x = element_text(angle = 20, hjust = 1))

becover_df <- tidy(ss1, stats = "becover") %>%
  mutate(method = fct_recode(method,
                             "IPW (Normal)" = "normal",
                             "IPW (Quantile Binning)" = "qbin",
                             "TMLE (Shift)" = "tmle"))

p3 <- ggplot(becover_df, aes(x = method, y = est, ymin = lower, ymax = upper)) +
  geom_point() +
  geom_errorbar(width = 0.2) +
  geom_hline(yintercept = 0.95, linetype = "dashed") +
  labs(y = "Bias-eliminated Coverage (95%)", x = "") +
  theme_bw(base_size = 12) +
  theme(panel.grid = element_blank(),
        axis.text.x = element_text(angle = 20, hjust = 1))

# Combine plots side by side
combined_plot <- p1 + p2 + p3 + plot_layout(ncol = 3)

# Save to PNG
ggsave("sim_summary_bw.png", combined_plot, width = 10, height = 4, dpi = 600)
```

2.5.1 Trace plot

This plot shows the cumulative mean of estimates over simulations, indicating convergence.

```
# Plotting the estimates
true_logOR <- 0.4  # True conditional log-odds ratio

# Cumulative mean
results$cum_mean_normal <- cumsum(results$normal) / seq_len(nrow(results))
results$cum_mean_qbin <- cumsum(results$qbin) / seq_len(nrow(results))
results_tmle$cum_mean_tmle <- cumsum(results_tmle$logOR_tmle) / seq_len(nrow(results_tmle))

# Add iteration index
results$iteration <- seq_len(nrow(results))
results_tmle$iteration <- seq_len(nrow(results_tmle))

# Combine for plotting
cum_long <- rbind(
```

```r
  data.frame(iteration = results$iteration, cum_mean = results$cum_mean_normal, method = "Normal Model")
  data.frame(iteration = results$iteration, cum_mean = results$cum_mean_qbin, method = "Quantile Binning
  data.frame(iteration = results_tmle$iteration, cum_mean = results_tmle$cum_mean_tmle, method = "TMLE")
)

# Plot
library(ggplot2)
ggplot(cum_long, aes(x = iteration, y = cum_mean, color = method)) +
  geom_line(size = 1) +
  geom_hline(yintercept = true_logOR, linetype = "dashed", color = "black") +
  labs(
    title = "Cumulative Mean of log(OR) Over Simulations",
    x = "Simulation Iteration",
    y = "Cumulative Mean of log(OR)",
    color = "Estimator"
  ) +
  theme_minimal()
```