Table1-ImputeTrue-WeibullX (The Commentary Edition)

This notebook comments the living daylights out of the script used to produce Table 1 in the "It's Integral" paper, which really sticks it to the nemesis and does not deserve the hatred of the dreaded Reviewer #2. It censors a Weibull covariate X and imputes with the true survival function S(X|Z).

The block below loads the appropriate packages and defines our experiment settings and ensures we remain in our reproducible era, which is better than the nemesis ever did! We specified a sample size of 100.

```
#The Setup Block
## RUN ONCE: install.packages("devtools")
## RUN ONCE: devtools::install_github("sarahlotspeich/imputeCensRd", ref = "main")
#library(imputeCensRd)
set.seed(114)
censoring = "light" # censoring rate, other options included "moderate" or "heavy"
n = 100 # sample size, other options included 500, 1000, or 2000
```

This block generates the data for the experiment. It creates an uncensored covariate Z~Binomial(100,0.5), a covariate X~Weibull(0.75,0.25) that will eventually be censored, a random error vector e simulated from a standard normal distribution, and specifies a true linear relationship and a response variable. Q sets up a censoring condition to be used as a scale parameter in the censoring mechanism. I am confused about the construction of C. My understanding so far is that if C > X, we consider the observation censored. This just means we generated more data than we can use. Why does changing the scale of the generating distribution censor the data? W compares X and C and outputs their minimums, and D is an indicator that returns 1 if $X \leq C$. Since D is an indicator, its mean is the percentage that isn't censored.

```
#The Data Generation and ProofReading Block
z = rbinom(n = n, size = 1, prob = 0.5) # Uncensored covariate
x = rweibull(n = n, shape = 0.75, scale = 0.25) # To-be-censored covariate
e = rnorm(n = n, mean = 0, sd = 1) # Random errors
y = 1 + 0.5 * x + 0.25 * z + e # Continuous outcome
q = ifelse(censoring == "light", 2,
           ifelse(censoring == "moderate", 0.35, 0.05)) # Rate parameter for censoring
c = rweibull(n = n, shape = 1, scale = q) # Random censoring mechanism
w <- pmin(x, c) # Observed covariate value
d <- as.numeric(x <= c) # "Event" indicator
dat = data.frame(z, w, y, d) # Construct dataset

# Check % censored
1 - mean(dat$d) # 12%
```

```
## [1] 0.12
```

This block writes a function to specify the true survival function (modeled by a Weibull distribution). Why does it take in z as a parameter? I'm not seeing it used.

```r
# Write a function for the true survival function used to generate Weibull X
trueSURV = function(q, z) {
  pweibull(q = q, shape = 0.75, scale = 0.25, lower.tail = FALSE)
}
```

This block is copied from the imputeCensRd package on Github, because I couldn't figure out how the cmi_custom function worked without doing a little detective work. Someone call Sherlock, I'm coming for him! I've learned that Delta=0 indicates a censored observation, and we use similar variable naming conventions. The commented section on top is helpful to understand the function structure. If a Z isn't specified, does that assume the passed-in survival function has a default null Z? Is that why it wasn't used in the survival function above? Is uncens just a copy of Delta? Is t-diff the difference in function heights?

```r
#' Custom conditional mean imputation (CMI) for a censored predictor with user-specified survival funct
#'
#' Custom conditional mean imputation (CMI) for a censored predictor using (externally calculated) user
#'
#' @param W Column name of observed predictor values (including censored opens).
#' @param Delta Column name of censoring indicators. Note that \code{Delta = 0} is interpreted as a cen
#' @param Z Column name of additional fully observed covariates.
#' @param data Dataframe or named matrix containing columns \code{W}, \code{Delta}, and \code{Z}.
#' @param useSURV Assumed survival function for \code{W} given \code{Z}. The only arguments to \code{us
#' @param trapezoidal_rule A logical input for whether the trapezoidal rule should be used to approxima
#'
#' @return
#' \item{imputed_data}{A copy of \code{data} with added column \code{imp} containing the imputed values
#' \item{code}{Indicator of algorithm status (\code{TRUE} or \code{FALSE}).}
#'
#' @export

cmi_custom <- function(W, Delta, Z, data, useSURV, trapezoidal_rule = FALSE) {
  # Calculate survival with original model coefficients using custom function
  if(is.null(Z)) {
    data <- data.frame(data, surv = useSURV(data[, W]))
  } else {
    data <- data.frame(data, surv = useSURV(data[, W], data[, Z]))
  }

  # Order data by W
  data <- data[order(data[, W]), ]

  # Create an indicator variable for being uncensored
  uncens <- data[, Delta] == 1

  # Calculate imputed values
  data$imp <- data[, W]
  if (trapezoidal_rule) {
    # Distinct rows (in case of non-unique obs values)
    data_dist <- unique(data[, c(W, Delta, Z, "surv")])

    # [T_{(i+1)} - T_{(i)}]
    t_diff <- data_dist[- 1, W] - data_dist[- nrow(data_dist), W]

    # Censored subject values (to impute)
```

```
    t_cens <- data[data[, Delta] == 0, W]

    # Follow formula assuming Accelerated Failure Time model for S(X|Z)
    for (x in which(!uncens)) {
      Cj <- data[x, W]
      Sj <- data_dist[-1, "surv"] + data_dist[- nrow(data_dist), "surv"]
      num <- sum((data_dist[-nrow(data_dist), W] >= Cj) * Sj * t_diff)
      denom <- data[x, "surv"]
      data$imp[x] <- (1 / 2) * (num / denom) + Cj
    }
  } else {
    ## Use integrate() to approximate integral from W to \infty of S(t|Z) (this is adaptive quadrature)
    if (is.null(Z)) {
      int_surv <- sapply(
        X = which(!uncens),
        FUN = function(i) {
          tryCatch(expr = integrate(f = function(t) useSURV(t), lower = data[i, W], upper = Inf)$value,
                   error = function(e) return(NA))
        }
      )
    } else {
      int_surv <- sapply(
        X = which(!uncens),
        FUN = function(i) {
          tryCatch(expr = integrate(f = function(t) useSURV(t, data[i, Z]), lower = data[i, W], upper =
                   error = function(e) return(NA))
        }
      )
    }
    ## Calculate E(X|X>W,Z) = int_surv / surv(W|Z) + W
    data$imp[which(!uncens)] <- data[which(!uncens), W] + int_surv / data[which(!uncens), "surv"]
  }

  ## Check for infinite/NA imputed values
  if (any(is.na(data$imp))) {
    data$imp[which(is.na(data$imp))] <- data[which(is.na(data$imp)), W]
  }
  if (any(data$imp  == Inf)) {
    data$imp[which(data$imp == Inf)] <- data[which(data$imp == Inf), W]
  }

  # Return input dataset with appended column imp containing imputed values
  if (any(is.na(data$imp))) {
    return(list(imputed_data = data, code = FALSE))
  } else {
    return(list(imputed_data = data, code = TRUE))
  }
}
```

This block executes the imputation. It calculates conditional means with trapezoidal rule and adaptive quadrature, and this is done just by changing the rule parameter in the cmi_custom function from the imputeCensRd package.

```r
# Imputation approach 1: Calculate conditional means with trapezoidal rule
trap_rule_imp = cmi_custom(W = "w", Delta = "d", Z = "z", data = dat, useSURV = trueSURV, trapezoidal_ru
trap_rule_imp$code # If TRUE, imputation was successful
```

```
## [1] TRUE
```

```r
trap_rule_fit <- lm(y ~ imp + z, data = trap_rule_imp$imputed_data) # Fit analysis model to imputed dat

# Imputation approach 2: Calculate conditional means with adaptive quadrature
adapt_quad_imp = cmi_custom(W = "w", Delta = "d", Z = "z", data = dat, useSURV = trueSURV, trapezoidal_r
adapt_quad_imp$code # If TRUE, imputation was successful
```

```
## [1] TRUE
```

```r
adapt_quad_fit <- lm(y ~ imp + z, data = adapt_quad_imp$imputed_data) # Fit analysis model to imputed d

# Compare models
trap_rule_fit
```

```
##
## Call:
## lm(formula = y ~ imp + z, data = trap_rule_imp$imputed_data)
##
## Coefficients:
## (Intercept)          imp            z
##      1.3558       0.4605      -0.1127
```

```r
adapt_quad_fit
```

```
##
## Call:
## lm(formula = y ~ imp + z, data = adapt_quad_imp$imputed_data)
##
## Coefficients:
## (Intercept)          imp            z
##      1.3562       0.4351      -0.1079
```