

# Example 1 using R

## Time to event from parametric hazards

David Garibay, M.P.P.\*      Hawre Jalal, MD, Ph.D.<sup>†</sup>  
Fernando Alarid-Escudero, Ph.D.<sup>‡§</sup>

### Code function

This document presents the code corresponding to the first example presented in the “A Fast Nonparametric Sampling (NPS) Method for Time-to-Event in Individual-Level Simulation Models.” manuscript, all of them using R.

```
# 01 Initial Setup -----  
  
# Clean global environment  
remove(list = ls())  
  
# Free unused R memory  
gc()  
  
# 02 Define general parameters -----  
  
#` Number of samples to draw from the life table  
n_samp_life_tables <- 1e5  
  
#' Numberof samples, by sex, to draw from the life table  
n_samp_by_sex <- 1e5  
  
# Sample size for every sampling iteration
```

---

\*School of Epidemiology and Public Health, Faculty of Medicine, University of Ottawa, Ottawa, ON, CA.

<sup>†</sup>School of Epidemiology and Public Health, Faculty of Medicine, University of Ottawa, Ottawa, ON, CA.

<sup>‡</sup>Department of Health Policy, Stanford University School of Medicine, Stanford, CA, USA.

<sup>§</sup>Center for Health Policy, Freeman Spogli Institute, Stanford University, Stanford, CA, USA.

```

n_samples <- 1e4

# Number of times to repeat the sampling
n_sim_reps <- 1e3

# Number of repetitions in microbenchmark
n_reps_microbench <- 100

# Seed for reproducibility in random number generation
n_seed <- 10242022

# 03 Define required functions -----

# Calculate expected value using NPS method
calc_ev_nps <- function(n_samples, v_probs){

  # Sample times to event
  v_time_to_event_rates_cat <- sample(size = n_samples,
                                     x     = 0:150,
                                     prob  = v_probs,
                                     replace = TRUE)

  return(mean(v_time_to_event_rates_cat))
}

# Calculate expected value using NPS approach and continuous time approximation
calc_ev_nps_corr <- function(n_samples, v_probs){

  # Sample times to event
  v_time_to_event_rates_cat <- sample(size = n_samples,
                                     x     = 0:150,
                                     prob  = v_probs,
                                     replace = TRUE)

  # Generate a random number following a uniform distribution
  v_unif <- runif(n_samples)

  # Add random number
  v_time_to_event_rates_cat_unif <- v_time_to_event_rates_cat + v_unif

  return(mean(v_time_to_event_rates_cat_unif))
}

```

```

calc_stats_nps_corr <- function(n_reps = 1000,
                                n_samples = 10000, v_time_int = 0:150,
                                v_probs, alpha_level = 0.05,
                                true_ev, true_var){
  # Draw 'n_reps' samples from the NPS TTT
  m_out <- replicate(n = n_reps,
                     expr = sample_ttt_nps(n_samples = n_samples,
                                           v_time_int = v_time_int,
                                           v_probs = v_probs))

  ## Calculate summary statistics
  v_ev_est <- colMeans(m_out)
  v_var_est <- matrixStats::colVars(m_out)

  v_se_ev_est <- sqrt(v_ev_est/n_samples)
  v_se_var_est <- sqrt(v_var_est/n_samples)

  # Expected values
  mean_ev_est <- mean(v_ev_est)
  mean_var_est <- mean(v_var_est)

  # Bias
  bias_ev_est <- abs(mean(v_ev_est) - true_ev)
  bias_var_est <- abs(mean(v_var_est) - true_var)

  # Monte Carlo Standard Error (MCSE) of Bias
  mcse_bias_ev_est <- sqrt(
    (sum((v_ev_est - true_ev)^2)/(n_reps - 1))/n_reps)

  mcse_bias_var_est <- sqrt(
    (sum((v_var_est - true_var)^2)/(n_reps - 1))/n_reps)

  # Mean Square Error (MSE)
  mse_ev_est <- sum((v_ev_est - true_ev)^2)/n_reps
  mse_var_est <- sum((v_var_est - true_var)^2)/n_reps

  # Confidence interval of bias
  z_score <- qnorm(p = 1 - alpha_level/2)
  ci_bias_ev_est <- c(LB = bias_ev_est - z_score*mcse_bias_ev_est,
                     UB = bias_ev_est + z_score*mcse_bias_ev_est)
  ci_bias_var_est <- c(LB = bias_var_est - z_score*mcse_bias_var_est,
                     UB = bias_var_est + z_score*mcse_bias_var_est)

```

```

# Confidence intervals pf estimates
chi_score_lb <- qchisq(p = alpha_level/2, df = (n_samples - 1))
chi_score_ub <- qchisq(p = 1 - alpha_level/2, df = (n_samples - 1))

m_ci_ev_est <- cbind(LB = v_ev_est - z_score*v_se_ev_est,
                    UB = v_ev_est + z_score*v_se_ev_est)
# DescTools::MeanCI(v_time_to_event_rates_cat_unif, conf.level = 0.95)
# confint(lm(v_time_to_event_rates_cat_unif ~ 1))
m_ci_var_est <- cbind(LB = ((n_samples - 1)*v_var_est)/chi_score_ub,
                    UB = ((n_samples - 1)*v_var_est)/chi_score_lb)
# DescTools::VarCI(v_time_to_event_rates_cat_unif, conf.level = 0.95)
# Coverage
coverage_ev_est <- mean(true_ev >= m_ci_ev_est[, 1] &
                       true_ev <= m_ci_ev_est[, 2])

coverage_var_est <- mean(true_var >= m_ci_var_est[, 1] &
                       true_var <= m_ci_var_est[, 2])

# Output
return(c(mean_ev_est = mean_ev_est,
        mean_var_est = mean_var_est,
        bias_ev_est = bias_ev_est,
        bias_var_est = bias_var_est,
        coverage_ev_est = coverage_ev_est,
        coverage_var_est = coverage_var_est
    ))
}

sample_ttt_nps <- function(n_samples,
                          v_time_int = 0:150,
                          v_probs){
  v_time_to_event_rates_cat <- sample(size = n_samples, x = v_time_int,
                                     prob = v_probs,
                                     replace = T)

  v_unif <- runif(n_samples)
  v_time_to_event_rates_cat_unif <- v_time_to_event_rates_cat + v_unif
  return(v_time_to_event_rates_cat_unif)
}

```

```

# 04 Calculate expected values from distributions -----

```

```

## Exponential distribution ----

```

```

# Define distribution parameters
par_exp_rate      <- 0.1

# Analytical values
ev_exp            <- 1/par_exp_rate # Analytical expected value
var_exp           <- ev_exp^2       # Analytical variance

# Get instantaneous probability of occurrence
v_prob_exp_rates <- pexp(q = 1:151, rate = par_exp_rate) -
  pexp(q = 0:150, rate = par_exp_rate)

# Simulations using the nps method multiple times
ev_exp_uncorr <- mean(
  replicate(n_sim_reps,
    expr = calc_ev_nps(n_samples = n_samples,
                      v_probs = v_prob_exp_rates)))

# Simulations using the nps method, adding continuous time approximation
ev_exp_corr <- mean(
  replicate(n_sim_reps,
    expr = calc_ev_nps_corr(n_samples = n_samples,
                          v_probs = v_prob_exp_rates)))

# Round values
ev_exp_corr      <- round(ev_exp_corr, 2)
ev_exp_uncorr    <- round(ev_exp_uncorr, 2)

# Measure mean execution time
## Without continuous time correction
l_mbench_exp_uncorr <- microbenchmark::microbenchmark(
  calc_ev_nps(n_samples = n_samples, v_probs = v_prob_exp_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into miliseconds
t_m_exp_u      <- format(round(mean(l_mbench_exp_uncorr$time/1e6), 2),
                        nsmall = 2)

t_CI_exp_u     <- format(round(quantile(x = l_mbench_exp_uncorr$time/1e6,
                                      probs = c(0.025, 0.975)), 2),
                        nsmall = 2)

```

```

## With continuous time correction
l_mbench_exp_corr <- microbenchmark::microbenchmark(
  calc_ev_nps_corr(n_samples = n_samples, v_probs = v_prob_exp_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into milliseconds
t_m_exp_c <- format(round(mean(l_mbench_exp_corr$time/1e6), 2),
  nsmall = 2)

t_CI_exp_c <- format(round(quantile(l_mbench_exp_corr$time/1e6,
  probs = c(0.025, 0.975)), 2),
  nsmall = 2)

# Using general function
v_sim_nps_out_exp <- calc_stats_nps_corr(n_reps      = n_sim_reps,
                                         n_samples   = n_samples,
                                         v_time_int  = 0:150,
                                         v_probs     = v_prob_exp_rates,
                                         alpha_level = 0.05,
                                         true_ev     = ev_exp,
                                         true_var    = var_exp)

## Gamma distribution ----

# Define distribution parameters
par_gamma_shape <- 4
par_gamma_rate <- 0.1

# Analytical values
ev_gamma <- par_gamma_shape/par_gamma_rate      # Analytical expected value
var_gamma <- par_gamma_shape/(par_gamma_rate^2) # Analytical variance

# Get instantaneous probability of occurrence
v_prob_gamma_rates <-
  pgamma(q = 1:151, shape = par_gamma_shape, rate = par_gamma_rate) -
  pgamma(q = 0:150, shape = par_gamma_shape, rate = par_gamma_rate)

# Simulations using the nps method multiple times
ev_gamma_uncorr <- mean(

```

```

replicate(n_sim_reps,
          expr = calc_ev_nps(n_samples = n_samples,
                             v_probs = v_prob_gamma_rates)))

# Simulations using the nps method, adding continuous time approximation
ev_gamma_corr <- mean(
  replicate(n_sim_reps,
            expr = calc_ev_nps_corr(n_samples = n_samples,
                                     v_probs = v_prob_gamma_rates)))

# Round values
ev_gamma_corr <- round(ev_gamma_corr, 2)
ev_gamma_uncorr <- round(ev_gamma_uncorr, 2)

# Measure mean execution time
## Without continuous time correction
l_mbench_gamma_uncorr <- microbenchmark::microbenchmark(
  calc_ev_nps(n_samples = n_samples, v_probs = v_prob_gamma_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into milliseconds
t_m_gamma_u <- format(round(mean(l_mbench_gamma_uncorr$time/1e6), 2),
                      nsmall = 2)

t_CI_gamma_u <- format(round(quantile(l_mbench_gamma_uncorr$time/1e6,
                                     probs = c(0.025, 0.975)), 2),
                      nsmall = 2)

## With continuous time correction
l_mbench_gamma_corr <- microbenchmark::microbenchmark(
  calc_ev_nps_corr(n_samples = n_samples, v_probs = v_prob_gamma_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into milliseconds
t_m_gamma_c <- format(round(mean(l_mbench_gamma_corr$time/1e6), 2),
                      nsmall = 2)

t_CI_gamma_c <- format(round(quantile(l_mbench_gamma_corr$time/1e6,

```

```

                                probs = c(0.025, 0.975)), 2),
                                nsmall = 2)

# Using general function
v_sim_nps_out_gamma <- calc_stats_nps_corr(n_reps = n_sim_reps,
                                           n_samples = n_samples,
                                           v_time_int = 0:150,
                                           v_probs = v_prob_gamma_rates,
                                           alpha_level = 0.05,
                                           true_ev = ev_gamma,
                                           true_var = var_gamma)

## Log-normal distribution ----

# Define distribution parameters
par_lnorm_meanlog <- 3.5
par_lnorm_sdlog   <- 0.15

# Analytical values
## Analytical expected value
ev_lnorm <- exp(par_lnorm_meanlog + ((par_lnorm_sdlog^2)/2))

## Analytical variance
var_lnorm <- exp(2*par_lnorm_meanlog + (par_lnorm_sdlog^2))*
  (exp(par_lnorm_sdlog^2) - 1)

# Round values
ev_lnorm <- round(ev_lnorm, 2)

# Get instantaneous probability of occurrence
v_prob_lnorm_rates <-
  plnorm(q = 1:151, meanlog = par_lnorm_meanlog, sdlog = par_lnorm_sdlog) -
  plnorm(q = 0:150, meanlog = par_lnorm_meanlog, sdlog = par_lnorm_sdlog)

# Simulations using the nps method multiple times
ev_lnorm_uncorr <- mean(
  replicate(n_sim_reps,
    expr = calc_ev_nps(n_samples = n_samples,
                      v_probs = v_prob_lnorm_rates)))

# Simulations using the nps method, adding continuous time approximation

```



```

ev_lnorm_corr <- mean(
  replicate(n_sim_reps,
    expr = calc_ev_nps_corr(n_samples = n_samples,
                           v_probs = v_prob_lnorm_rates)))

# Round values
ev_lnorm_corr <- round(ev_lnorm_corr, 2)
ev_lnorm_uncorr <- round(ev_lnorm_uncorr, 2)

# Measure mean execution time
## Without continuous time correction
l_mbench_lnorm_uncorr <- microbenchmark::microbenchmark(
  calc_ev_nps(n_samples = n_samples, v_probs = v_prob_lnorm_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into milliseconds
t_m_lnorm_u <- format(round(mean(l_mbench_lnorm_uncorr$time/1e6), 2),
  nsmall = 2)

t_CI_lnorm_u <- format(round(quantile(l_mbench_lnorm_uncorr$time/1e6,
  probs = c(0.025, 0.975)), 2),
  nsmall = 2)

## With continuous time correction
l_mbench_lnorm_corr <- microbenchmark::microbenchmark(
  calc_ev_nps_corr(n_samples = n_samples, v_probs = v_prob_lnorm_rates),
  times = n_reps_microbench,
  unit = "ms")

## The default output of microbenchmark is in nanoseconds (1/1e-9).
## The results are converted into milliseconds
t_m_lnorm_c <- format(round(mean(l_mbench_lnorm_corr$time/1e6), 2),
  nsmall = 2)

t_CI_lnorm_c <- format(round(quantile(l_mbench_lnorm_corr$time/1e6,
  probs = c(0.025, 0.975)), 2),
  nsmall = 2)

### Using general function

```

```
v_sim_nps_out_lnorm <- calc_stats_nps_corr(n_reps      = n_sim_reps,  
                                           n_samples   = n_samples,  
                                           v_time_int  = 0:150,  
                                           v_probs     = v_prob_lnorm_rates,  
                                           alpha_level = 0.05,  
                                           true_ev     = ev_lnorm,  
                                           true_var    = var_lnorm)  
  
# Remove seed  
set.seed(NULL)
```