Presenting Results: Visualizing Simulation Outcomes

Ashley I Naimi

Spring 2024

Contents

1 Visualizing Simulation Outcomes 2
2 Nested Loop Plot 2
3 Zipper Plots 7

1 Visualizing Simulation Outcomes

Once a simulation study is complete, the outcome of the study often consists of several datasets that contain information on the performance of the methods being studied. These datasets must then be processed in order to present the findings from the study.

Several different forms of presentation can be used, including tables and figures. Besides the common repertoire of figures that can be used to present results (histograms, density plots, scatter plots, heat maps and other), there are a few types of figures that can be tailored to a simulation study, and that can potentially reveal useful information about the performance of a set of methods under a particular set of circumstances.

Here, we will briefly introduce nested loop plots, and zipper (or “zip”)[1] plots, and demonstrate how they can be used to convey results from a simulation study.

[1] Sometimes zipper plots are referred to as zip plots (e.g., Morris et al., 2019). However, searching the term “zip plot” online will yield many plot structures tailored to plotting geographical regions using zip codes.

2 Nested Loop Plot

Nested loop plots can be used to represent the results of a simulation study over all possible cross-combination of parameters used to define the simulation data.[2]

[2] this section was based heavily on a very useful site by Michael Kammer: https://bit.ly/4bm77eP \ Thanks to an Emory Epi PhD student, Qi Zhang, for bringing this to my attention.

Suppose we conduct a simulation study evaluating the performance of three different methods under a range of difference scenarios. Suppose further that from our simulation code, we obtain a dataset with the following information:

```r
pacman::p_load(
  tidyverse,
  dplyr,
  purr,
  magrittr
  )

thm <- theme_classic() +
  theme(
    legend.position = "top",
    legend.background = element_rect(fill = "transparent", colour = NA),
    legend.key = element_rect(fill = "transparent", colour = NA)
  )
theme_set(thm)

set.seed(123)
params = list(
    samplesize = c(100, 200, 500),
    param1 = c(1, 2),
    param2 = c(1, 2, 3),
    param3 = c(1, 2, 3, 4)
)

design = expand.grid(params)

# add some "results"
design %<>%
    mutate(method1 = rnorm(n = n(),
                           mean = param1 * (param2 * param3 + 1000 / samplesize),
                           sd = 2),
           method2 = rnorm(n = n(),
                           mean = param1 * (param2 + param3 + 2000 / samplesize),
                           sd = 2),
           method3 = rnorm(n = n(),
                           mean = param1 * (param2 + param3 + 3000 / samplesize),
                           sd = 2))

knitr::kable(head(design, n = 10))
```

samplesize	param1	param2	param3	method1	method2	method3
100	1	1	1	9.879049	24.011477	28.796928
200	1	1	1	5.539645	10.581599	15.938187
500	1	1	1	6.117417	4.623983	5.076489
100	2	1	1	22.141017	46.051143	65.375833
200	2	1	1	12.258575	23.430454	38.200218
500	2	1	1	9.430130	9.558565	13.425939
100	1	2	1	12.921832	23.362607	34.575478
200	1	2	1	4.469877	12.722217	19.538085
500	1	2	1	2.626294	7.011528	9.664405
100	2	2	1	23.108676	46.770561	63.983247

We can use a nested loop plot to present these results in a single image. To deploy a nested loop plot, we first need to install the relevant package. This package is a development package hosted on GitHub, so we'll need to use the `remotes` or `devtools` packages to install it:

```r
remotes::install_github("matherealize/looplot")
```

We can then proceed to use the `nested_loop_plot` function, which relies on `ggplot2` functionality. With the dataset above, we can construct a plot using the following code:

```r
pacman::p_load(looplot)

p = nested_loop_plot(resdf = design,
                     x = "samplesize", steps = c("param2", "param3"),
                     grid_rows = "param1",
                     steps_y_base = -10, steps_y_height = 3, steps_y_shift = 10,
                     x_name = "Sample Size", y_name = "Error",
                     spu_x_shift = 200,
                     steps_values_annotate = TRUE, steps_annotation_size = 3,
                     hline_intercept = 0,
                     y_expand_add = c(10, NULL),
                     post_processing = list(
                        add_custom_theme = list(
                                    axis.text.x = element_text(angle = -90,
                                                               vjust = 0.5,
                                                               size = 8)
                               ))
                            )


ggsave(here("_images", "nested_loop_plot.pdf"), width = 8, height = 6)
```

Here is what the figure we generated looks like:

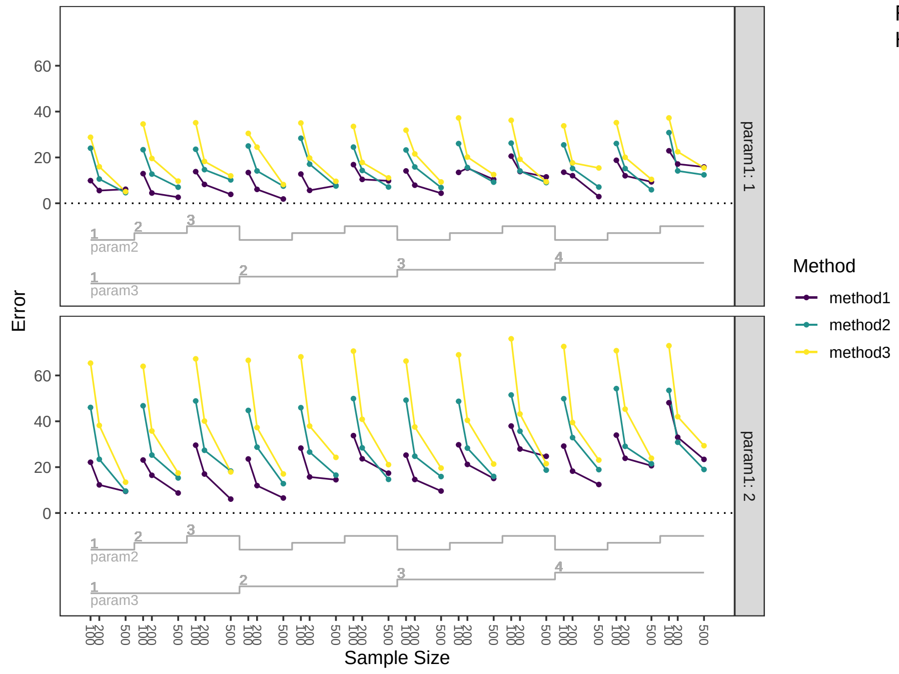


Figure 1: Example Nested Loop Plot of Hypothetical Simulation Results.

This Figure shows the magnitude of the simulated error for each sample size split by the two distinct `param1` values, across all combinations of `param2` and `param3`.

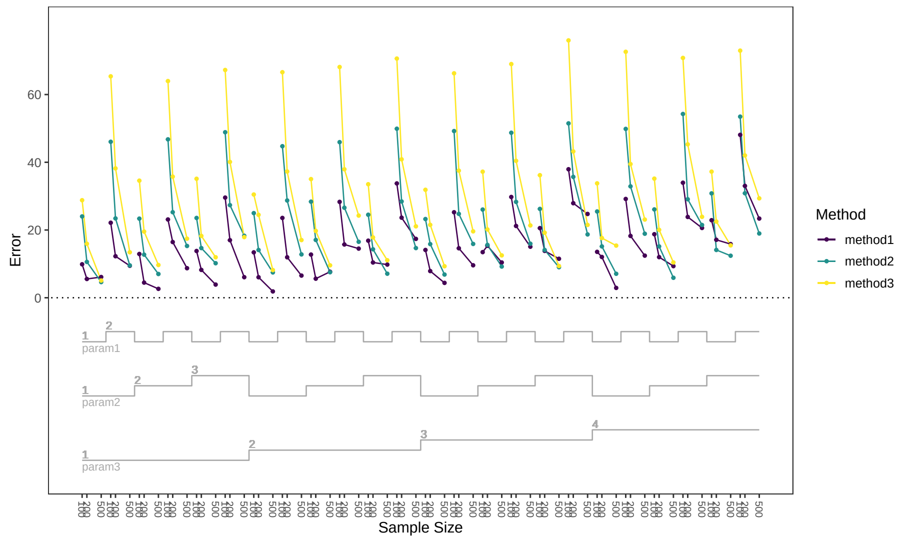
There are many different ways to formulate a plot like this. For example, we can remove the separate across `param1`

```
pacman::p_load(looplot)
```

```
p = nested_loop_plot(resdf = design,
                     x = "samplesize", steps = c("param1", "param2", "param3"),
                     #grid_rows = "param1",
                     steps_y_base = -10, steps_y_height = 3, steps_y_shift = 10,
                     x_name = "Sample Size", y_name = "Error",
                     spu_x_shift = 200,
                     steps_values_annotate = TRUE, steps_annotation_size = 3,
                     hline_intercept = 0,
                     y_expand_add = c(10, NULL),
                     post_processing = list(
                        add_custom_theme = list(
                            axis.text.x = element_text(angle = -90,
                                                       vjust = 0.5,
                                                       size = 8)
                        ))
                     )


ggsave(here("_images", "nested_loop_plot2.pdf"), width = 10, height = 6)
```

Which gives us a single panel figure:



We can add separate grids for each parameter as well. For a range of different options of the nested loop plot, refer to the package demo: https://bit.ly/4bm77eP.

3 Zipper Plots

Zipper plots are most often used to present bounds, such as confidence intervals. Consider data from the `rsimsum` package on the performance of different methods to estimate a hazard ratio when the baseline hazard is misspecified:

```
##   dataset  n    baseline       theta        se model
## 1       1 50 Exponential -0.88006151 0.3330172   Cox
## 2       2 50 Exponential -0.81460242 0.3253010   Cox
## 3       3 50 Exponential -0.14262887 0.3050516   Cox
## 4       4 50 Exponential -0.33251820 0.3144033   Cox
## 5       5 50 Exponential -0.48269940 0.3064726   Cox
## 6       6 50 Exponential -0.03160756 0.3097203   Cox
```

```
## [1] 1200    6
```

```
##
##  50 250
## 600 600
```

```
##
## Exponential     Weibull
##         600         600
```

```
##
##   Cox   Exp RP(2)
##   400   400   400
```

The survival outcomes in each dataset were simulated from a binary treatment variable with a log-hazard ratio of -0.50, under sample sizes of 50 and 250 individuals, and under two different baseline hazard functions (exponential and Weibull). We can also see that for each combination of simulation parameters (sample size, baseline hazard, and model), the Monte Carlo sample size was 100:

```r
relhaz %>%
  group_by(n, baseline, model) %>%
  count()
```

```
## # A tibble: 12 x 4
## # Groups:   n, baseline, model [12]
##         n baseline    model    nn
##     <dbl> <chr>       <chr> <int>
##  1     50 Exponential Cox     100
##  2     50 Exponential Exp     100
##  3     50 Exponential RP(2)   100
##  4     50 Weibull     Cox     100
##  5     50 Weibull     Exp     100
##  6     50 Weibull     RP(2)   100
##  7    250 Exponential Cox     100
##  8    250 Exponential Exp     100
##  9    250 Exponential RP(2)   100
## 10    250 Weibull     Cox     100
## 11    250 Weibull     Exp     100
## 12    250 Weibull     RP(2)   100
```

Each of the 100 simulated datasets was then analyzed using a Cox proportional hazards regression model, a parametric exponential model, and a flexible parametric model developed by Patric Royston and Mahesh Parmar, where the baseline hazard is fit with natural cubic splines with two degrees of freedom (Royston and Parmar, 2002).[3]

[3] Additional details on this dataset are available here: `https://bit.ly/3K6zp0r`

Suppose we're interested in exploring the performance of the normal-interval (or Wald) confidence interval estimator in these data. We can construct upper and lower confidence interval bounds in the data above using the standard equation:

$$(LCL, UCL) = \hat{\theta} \pm 1.96 \times SE(\hat{\theta})$$

In R, we could implement this as follows:

```r
relhaz <- relhaz %>%
  mutate(lcl = theta - 1.96*se,
         ucl = theta + 1.96*se)

head(relhaz)
```

```
##   dataset  n    baseline       theta        se model        lcl        ucl
## 1       1 50 Exponential -0.88006151 0.3330172   Cox -1.5327752 -0.2273478
## 2       2 50 Exponential -0.81460242 0.3253010   Cox -1.4521923 -0.1770125
## 3       3 50 Exponential -0.14262887 0.3050516   Cox -0.7405299  0.4552722
## 4       4 50 Exponential -0.33251820 0.3144033   Cox -0.9487487  0.2837123
## 5       5 50 Exponential -0.48269940 0.3064726   Cox -1.0833857  0.1179869
## 6       6 50 Exponential -0.03160756 0.3097203   Cox -0.6386593  0.5754442
```

We can also add an indicator of whether the bounds just created include the true value of -0.5 or not:

```r
relhaz <- relhaz %>%
  mutate(include_flag = if_else(lcl<-.5 & ucl>-.5, "Include", "Exclude"))
```

With these upper and lower bounds, we can now create a zipper plot for a subset of these results:

```r
p <- relhaz %>%
  filter(n == 50, baseline == "Exponential") %>%
  ggplot(.) +
  geom_hline(yintercept = -.5, lty = 2) +
  geom_pointrange(aes(x = dataset,
                      y = theta,
                      ymin = lcl,
                      ymax = ucl, color = include_flag),
                  size = .2,
                  alpha = .75) +
  scale_color_manual(values=c("red","grey")) +
  ylab("log Hazard Ratio") +
  xlab("Sample Number") +
  coord_flip() +
  theme(legend.position = "none", text=element_text(size=12)) +
  facet_wrap(~model)

ggsave(here("_images", "zip_plot_version1.pdf"), p)
```

This plot reveals the distribution of confidence intervals across all 100 iterations:

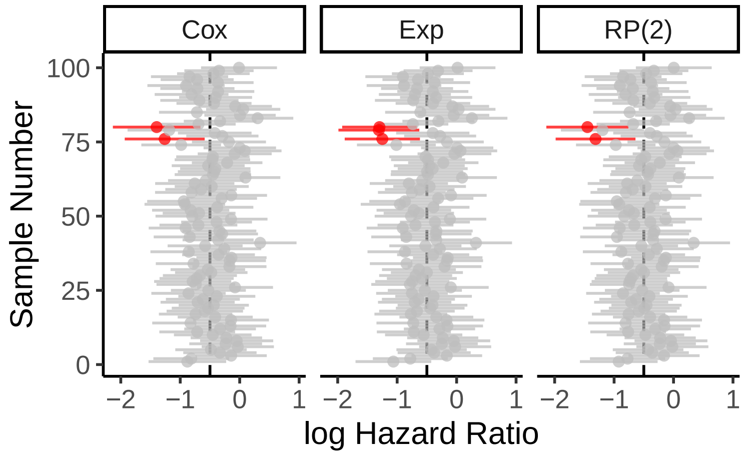


Figure 2: Zipper plot displaying the distribution of normal-interval (Wald) confidence intervals in the relhaz data.

Some authors like to present these zipper plots with the bounds ranked according to some criterion. For confidence intervals, we can rank our results according to the magnitude of the Wald test statistic for, say, a null test hypothesis:

```
relhaz <- relhaz %>%
  mutate(test_statistic = abs(theta/se))
```

We can then incorporate an `arrange` argument into our plot code:

```
p <- relhaz %>%
  filter(n == 50, baseline == "Exponential") %>%
  ggplot(.) +
  geom_hline(yintercept = -.5, lty = 2) +
  geom_pointrange(aes(x = test_statistic,
                      y = theta,
                      ymin = lcl,
                      ymax = ucl, color = include_flag),
                  size = .2,
                  alpha = .75) +
  scale_color_manual(values=c("red","grey")) +
  ylab("log Hazard Ratio") +
  xlab("Wald Null Test Statistic") +
  coord_flip() +
  theme(legend.position = "none", text=element_text(size=12)) +
  facet_wrap(~model)

ggsave(here("_images", "zip_plot_version2.pdf"), p)
```

Which gives us the following modified figure:

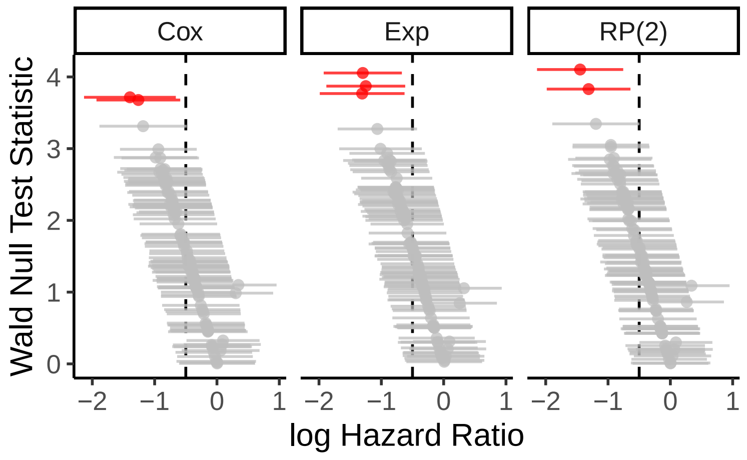


Figure 3: Zipper plot displaying the distribution of normal-interval (Wald) confidence intervals in the relhaz data. Bounds are ranked according to the magnitude of the Wald test statistic for each point estimate.

Alternative rankings for the y-axis can be considered, such as the magnitude of the standard error, or the centile rank of the test statistic.

References

Tim P. Morris, Ian R. White, and Michael J. Crowther. Using simulation studies to evaluate statistical methods. *Statistics in Medicine*, 38(11):2074–2102, 2019.

Patrick Royston and Mahesh K. B. Parmar. Flexible parametric proportional-hazards and proportional-odds models for censored survival data, with application to prognostic modelling and estimation of treatment effects. *Statistics in Medicine*, 21(15):2175–2197, 2002.