generating-missing-data

Tony Ni

9/18/2020

Playing around with how to make vectors and datasets with missing values...

Some websites/sources:

https://cran.r-project.org/web/packages/missMethods/vignettes/Generating-missing-values.html https://rmisstastic.netlify.app/how-to/generate/misssimul

Look through this website: https://www.itrcweb.org/gsmc-1/Content/GW%20Stats/5%20Methods%20in%20indiv%20Topics/5%207%20Nondetects.htm#:~:text=Robust%20ROS%20is%20semi%2Dparametric,are%20made%20for%20t

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(missMethods) #package for generating missing data
library(NADA) #package for mle for left censored data
```

```
## Loading required package: survival
##
## Attaching package: 'NADA'
## The following object is masked from 'package:stats':
##
##     cor
```

Generating missing data

Assuming log-normal distribution, which is the most common distribution for censored water data, (refer to technotes pdf file) for contaminant values in groundwater data...

Ok let's just try to pull random numbers from a logormal(1, 1) distribution for a "artificial" dataset.

```
set.seed(7271999)

num <- 1000

m <- 1
s <- 1
```

```
location <- log(m^2 / sqrt(s^2 + m^2))
shape <- sqrt(log(1 + (s^2 / m^2)))
print(paste("location:", location))
```

```
## [1] "location: -0.346573590279973"
```

```
print(paste("shape:", shape))
```

```
## [1] "shape: 0.832554611157698"
```

```
id <- seq(1, num, by = 1)
value <- rlnorm(num, location, shape)

my_df <- as.data.frame(cbind(id, value))
```

Now, to play around with the methods in the `missMethods` package. These methods let us generate missing values in a dataset, in our case – the artificial dataset we generated above.

```
my_df2 <- delete_MAR_censoring(ds = my_df, #dataframe
                     p = 0.3, #probability that a value is missing
                     cols_mis = "id",
                     cols_ctrl = "value")

glimpse(my_df2)
```

```
## Rows: 1,000
## Columns: 2
## $ id    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, NA, 12, NA, 14, 15, 16, 17, 18...
## $ value <dbl> 1.5557094, 0.9067474, 0.8399910, 1.0081827, 0.7199531, 1.9305...
```

Uh... I don't think the methods in this package are what we're looking for... This `delete_MAR_censoring` "generate MAR values using a censoring mechanism. This leads to a missing value in `id`, if the value is below the 30% quantile of `value`" We really only have 1 numeric variable of interest here...

Maybe we can just do it by hand...

Let's just say something like, for any value below `some number`, we make a new column and mark it as being censored/below limit of detection. I just chose 0.5 as a completely arbitrary value...

```
threshold <- 0.3

my_df3 <- my_df %>%
  mutate(below_detection = case_when(value <= threshold ~ "T",
                                     value > threshold ~ "F"))
tibble(my_df3)
```

```
## # A tibble: 1,000 x 3
##       id value below_detection
##    <dbl> <dbl> <chr>
##  1     1 1.56  F
##  2     2 0.907 F
##  3     3 0.840 F
##  4     4 1.01  F
##  5     5 0.720 F
##  6     6 1.93  F
##  7     7 1.14  F
##  8     8 1.20  F
##  9     9 0.842 F
```

```
## 10    10 0.847 F
## # ... with 990 more rows
```

Great! This is basically the general format of how missing values are encoded in the groundwater data – there is a variable called called `<` which is `<` is the value is below the LOD and left blank if not. This is exactly what we want! This is a bit weird because it doesn't make sense to have negative concentrations, but it's not worth it to try to fix it since we just want some artificial dataset to work with at the moment (may be in the future though)...

Playing around with missing data

Now, we want to try playing around with this dataset, let's try seeing what our sample mean and sd are – as a comparison point.

```
mean(my_df3$value)
```

```
## [1] 0.9599192
```

```
sd(my_df3$value)
```

```
## [1] 0.9049119
```

Now, let's make the values for which the observations in which `below_detection` is `T` – NA.

```
my_df4 <- my_df3 %>%
  mutate(value = if_else(
    below_detection == "F", value, NA_real_))

glimpse(my_df4)
```

```
## Rows: 1,000
## Columns: 3
## $ id              <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
## $ value           <dbl> 1.5557094, 0.9067474, 0.8399910, 1.0081827, 0.71995...
## $ below_detection <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "...
```

Substitution Approach

Ok, let's true our substitution approach where we impute in the missing values with value/2 and value/sqrt(2). Um... for this artificial dataset, we don't really have a LOD but let's make it an arbitrary value 9.

First, replacing all our missing values with LOD/2

```
lod_div_2 <- my_df4 %>%
  mutate(value = if_else(
  below_detection == "T", 9/2, value))

mean(lod_div_2$value)
```

```
## [1] 1.562124
```

```
sd(lod_div_2$value)
```

```
## [1] 1.459535
```

Next, replacing all our missing value with LOD/sqrt(2)

```
lod_div_sqrt2 <- my_df4 %>%
  mutate(value = if_else(
  below_detection == "T", 9/sqrt(2), value))
```

```r
mean(lod_div_sqrt2$value)
```

```
## [1] 1.823078
```

```r
sd(lod_div_sqrt2$value)
```

```
## [1] 2.020845
```

We know the true population mean and sd of the distribution we pulled samples from is lognormal(1, 1)

The sample mean and sd of our artificial dataset is 0.95 and 0.905 respectively.

The mean and sd we get from LOD/2 substitution method yields 1.56 amd 1.46 respectively.

The mean and sd we get from LOD/sqrt(2 substitution method yields 1.82 and 2.02 respectively.

We can see that this method is OK... It doesn't really capture the true mean and sd very well...

MLE Approach

Now using the MLE approach, we can use the `cen_mle` function to compute statistics when the data contains left-censored values.

```r
my_df5 <- my_df4 %>%
  mutate(below_detection = as.logical(below_detection))

mle_res = cenmle(my_df5$value, my_df5$below_detection, conf.int=0.95, dist = "gaussian")

mean(mle_res)
```

```
##      mean        se   0.95LCL   0.95UCL
## 1.0838646 0.0312669 1.0225826 1.1451466
```

```r
sd(mle_res)
```

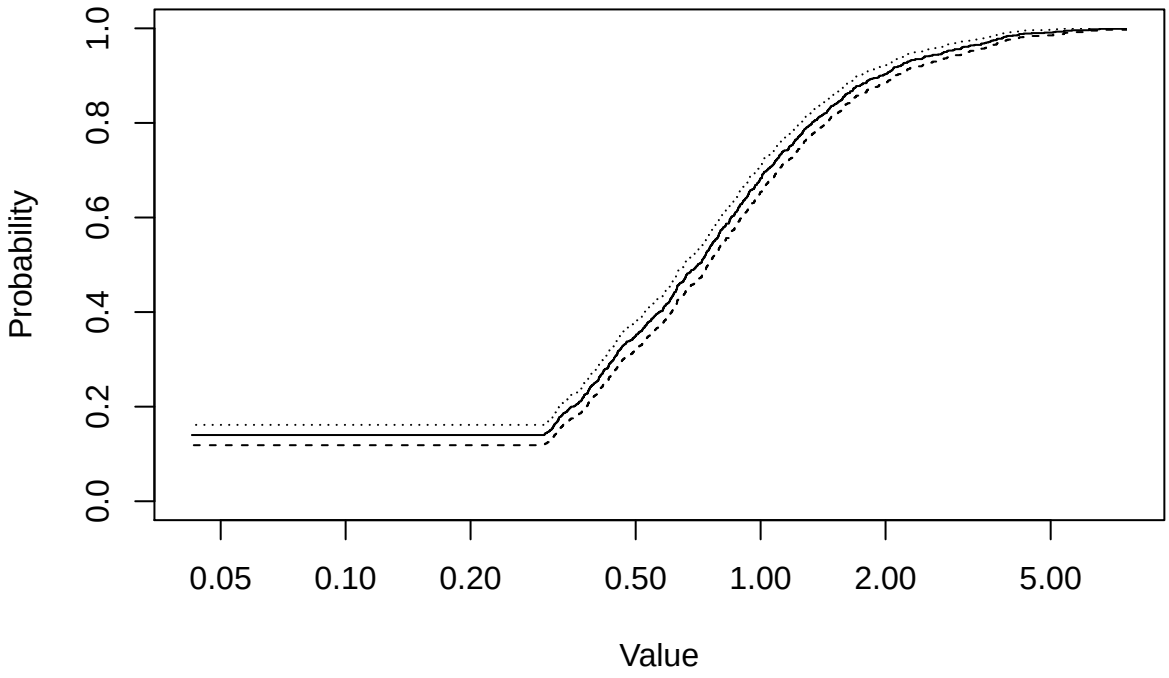
```
## [1] 0.9169254
```

We obtain a mean of 1.08 and sd of 0.92. This method performs much better than the substitution method!

Kaplan Meier Approach

This method doesn't have any distributional assumptions, which is an advantage it has over the MLE approach:

```r
km_res = cenfit(my_df$value, my_df5$below_detection, conf.int=0.95)
plot(km_res) #plots cdf of value
```




```
mean(km_res)
```

```
##       mean         se    0.95LCL    0.95UCL
## 0.97425809 0.02824486 0.91889918 1.02961701
```

```
sd(km_res)
```

```
## [1] 0.893181
```

We obtain a mean of 0.97 and sd of 0.89. This method is comparable to the MLE method!

Imputation Method with MLE

Imputation Method with KM