Metric Test

Zane

In this document, I’ll simulate titer data for our synthetic lab experiment. The goal here is to show that, if we could completely observe correct titer data, our metric would work better.

When we test our metric vs. the old metrics on the CIVIC data, the old metrics appear to have less variability. Right now, we think that this is because Ted’s cohort has a large amount of non-responders, and thus in addition to biasing our estimates lower, the uncertainty in the estimates is falsely reduced by imputing all of the values below the limit of detection with a constant. So in addition to biasing the actual estimate, this biases any variability estimates to be lower than they actually should.

# Data setup and notation

Suppose we have several different lab groups who each run separate cohort studies to test the immunogenicity of a vaccine candidate. Say there are of these and we label them . Each of these lab groups recruits people who we index by . For simplicity we can assume and this likely will not have a drastic effect on our results. We define for convenience.

Now we also need to suppose a finite universe of strains that we could get HAI assays for. And we suppose that these strains can be indexed only by their antigenic distance (*this is maybe a questionable assumption but many people implicitly make it*). Calling the antigenic distance , we would say the vaccine strain has distance , because it is identical to itself. For the purpose of this analysis, we will only consider a universe of strains where each strain has a unique distance. We can then consider every strain with a distance in (divided into some number of bins which are not necessarily of equal size), where a strain with distance would be as different as possible to the vaccine strain.

For example, if we take the antigenic distance to be the dominant -Epitope distance, then any strain where all epitopes are equivalent in amino acid sequence to the vaccine strain would have distance , and a strain where any epitope is different from the vaccine strain at all residues in that epitope would have distance . The possible antigenic distance values would be the union of all possible fractions formed by dividing a given number of differences at an epitope by the number of residues in that epitope. If all the epitopes are of equal sequence length, then this reduces to the set of all fractions where is the difference in epitope sequences and is the length of the sequence.

For convenience, we will consider different antigenic distance values (in addition to the homologous strain with ) which are equally spaced across the interval . So for example, if , the universe of potential strains would have distances .

Now, we can consider the scenario where each of the labs performing a cohort study does not sample all possible strains in the defined universe of strains. Real-world studies typically use a subset of relevant strains based on what they have available or what is convenient for them to obtain, so all labs will have some panel of strains that they test, which we call . The panel for lab will consist of strains plus the homologous strain. For simplicity, we assume that that is, all of the panels have the same number of strains. Of course this is untrue across real world studies. We then call the panel of strains used by lab and index it by . The homologous strain is also included in each panel and will be index as . Using the indexing notation, we can then say that the distance for strain is , the th entry in the panel for lab because the set indexes the panel . For convenience we denote this as , the th element of .

NOTE NEED TO WORK ON DISTANCES NOTATION. THE INDEXING IS A BIT CONFUSING. J SHOULD PROBABLY BE INDEPENDENT OF K, AND D CAN BE INDEXED BY SMALL D.

For each person in lab , assume they get the vaccine. We then want to get their HAI titer (or some assay) for the homologous strain and for all of the strains in . So denote the response to strain by that individual as . This response value can be written as a function of a corresponding distance value with random variation, i.e.,

where is some systematic model and is a random deviation from the predicted model. Here, is a vector of model parameters.

[PUT IN A TABLE OF NOTATION HERE]

## Data generating model

Now it remains to choose the form of the model. Rather than continuing to write the model in the form, we can adopt a parametric distribution for the model and write it in a more understandable way. We assume that each titer observation is drawn from some distribution and is therefore subject to some random variability. Typically, a normal distribution is a reasonable assumption here, so we can say

This is equivalent to specifying that

as in many references. Note that we have made some simplifying assumptions in the expression of this model.

1. The variance, is constant for all observations. This is unlikely to be true but is an extremely common simplifying assumption. We can address this assumption by using a hierarchical model, which we will deal with in a future step. This also implies that all measurements are uncorrelated.
2. The mean is dependent **only on the distance of a given strain.** That means that all strains of a given distance have the same expected titer. This combines multiple simplifying assumptions.
3. There are no systematic variations between labs – all between-lab variation comes from the random variation of each measurement. Measurements within the same lab are not correlated.
4. Multiple measurements for each individual are not correlated. There is no systematic variation between individuals.

Any further assumptions depend on what functional form we select for . The simplest choice would be

for some number . If we chose ([Equation 2](#eq-f-constant)), that is equivalent to saying that we expect an individual to have the same response (on average) to any strain, regardless of the antigenic distance to the vaccine, or any other factors.

We can make arbitrarily complicated, but for the purposes of our simulation we will use a common assumption for the form of : a linear model. To specify a linear model, we would use

instead. Under this model, a 0.1 unit increase in the distance (for example) would result in a unit increase in the expected titer. If is negative, that is a decrease instead. The parameter would then represent the expected titer for the homologous strain.

Note that we could further address several of the simplifying assumptions we described by using a hierarchical model that allows these parameters to vary between groups and be correlated. For example, our data generating model assumes that titer values are continuous, which is untrue. Real titers may have an underlying normal model (on the log scale), but we only observe discrete values and must account for both this and for limits of detection by physical assays. We will deal with these problems in future experiments.

We will leave those extensions to a future endeavor for now.

Finally, we note that in this format, the model is easy to implement in either a frequentist framework using a standard linear regression model, or through a Bayesian framework by the specification of priors for the model parameters (, , and ). The Bayesian framework permits easier generalization of the likelihood function to realistic cases, and the specification of priors can provide a number of benefits for inference, which we will discuss in a future experiment. However, the frequentist framework is much less computationally intensive so we will use the standard maximum likelihood estimation approach for our proof-of-concept study.

With the model specified, we next need to discuss how we will quantify the breadth of a vaccine based on the simulated lab studies.

## Breadth calculation

In each lab’s immunogenicity study, they recruit people and test each of these people against virus strains. This gives us a matrix of responses for each lab (and extends to a partially missing matrix in the general case where and vary). If we want to quantify how good a vaccine candidate is, we need to turn these data matrices into summary statistics. We propose that a vaccine candidate should be evaluate not only on the **magnitude** of the elicited response, but also by the **breadth** of the response. Furthermore, we can attempt to combine these to get a single-statistic summary of the **total strength** of the vaccine.

### Current method

In the current literature, there is no real analogue for measuring the total strength of the vaccine. Instead, the magnitude and breadth can be measured by one of two different statistics, depending on the outcome of interest. However, the general idea for both methods is to analyze the homologous strain only to measure strength, and to pool all of the strains together with no notion of antigenic distance to measure the breadth.

The two statistics that are commonly used are the **geometric mean titer** (GMT) and the **seroconversation rate**. The GMT is simply the geometric mean of the titers of all the individuals in the study. For evaluating strength, this is straightforward. For evaluating breadth, we pool all the strains together to evaluate the GMT, which is less intuitive. To calculate the seroconversion rate (which is not really a rate, because there is no element of time involved; epidemiologically this is a risk), we first construct an indicator variable for **seroprotection**. An individual is seroprotected if their final titer is at some threshold or above (for influenza HAI titers, this threshold is 1:40). Seroconversion is also determined by titer increase, which requires us to introduce an element of time that we previously have ignored.

Typically, studies of the type we are interested in measure an immunological assay both pre- and post-vaccination, so that baseline titer can be adjusted out and we can better isolate the effect of the vaccine. In this scenario, we often calculate the **titer increase**, the ratio of the post-vaccination titer to the pre-vaccination titer. An individual is said to have seroconverted if they are seroprotected post vaccination and also had at least a four-fold titer increase. Thus, seroconversion rate is a proxy for the vaccine’s ability to induce protection in unprotected individuals and can be thought of as the incidence counterpart to the prevalence measurement represented by seroprotection. We estimated the seroconversion rate for a given study by multiplying the indicator variables for seroprotection and four-fold increase, and then finding the mean of the resulting variate. For the purposes of our first experiment, we will assume that all individuals are naive at the pre-vaccination time point. This is a very silly assumption that is literally never true in any human population, but we should start with as simple a model as possible. We will incorporate time distances and the direct effect of the vaccine in a future simulation study.

### Our proposed method

We propose that a more robust and informative summary of vaccine breadth and strength can be obtained by fitting the linear model described previously and calculating summary statistics based on the linear model. Fitting a statistical model enforces a smoothing effect to reduce noise in the raw data and accounting for antigenic distance can help us to better understand the breadth of the response. After fitting the linear model, we will evaluate the three components of the response that we’ve outlined as follows.

The magnitude is naturally estimated as the intercept of the linear model. The intercept represents the conditional expectation of the titer when the antigenic distance is zero, which is exactly what we want for an estimate of the magnitude.

The breadth is more difficult to define, and we initially attempted to use the slope to estimate the breadth of the response. However, this does not include any information about the magnitude and can lead to unintuitive situations where a vaccine that induces no response to any strain is in one sense “more broad” than a vaccine that only induces a response to the homologous strain (or a few closely related strains). So we chose to incorporate an element of seroprotection into this definition, and calculate breadth as the proportion of the estimated regression line that is above the threshold (again, 1:40 for influenza HAI). Numerically, this can be calculated by creating a sufficiently dense grid of evenly spaced x-values ranging from 1:40, getting the predicted values from the regression line, and taking the proportion of points that are above 1:40.

Finally, we estimate the total strength of the vaccine by calculating the area under the curve (AUC) of the linear regression line. The AUC is conceptually a way to weight the average used in the GMT method, that also adjusts for noise in the data based on the statistical model of interest, and covers a continuous range of interpolated values rather than just the discrete set of values for the observed points. The AUC can be estimated using a simple numerical integration technique. For our purposes, the trapezoid method is sufficient. Since the curve is also a linear regression line, we can analytically derive the unweighted AUC as the area of a triangle. Applying different weighting schemes to the AUC may help to make better judgments about vaccine candidates in specific scenarios, although there is no objectively correct weighting scheme and expert opinion is required. For the purposes of this simulation study we consider only the unweighted AUC.

### Our hypothesis

Based on our previous test to validate our method on real data, we believe that censoring inherent to real-world titer data makes the current method look falsely better than our new proposed method. So we want to validate our method on theoretical data, and then we can determine if accounting for the limit of detection in real world data solves the problem.

# Simulation study

So the first thing we need to do is get a simulation study working. The basic steps to running our simulation will be as follows.

1. Determine the simulation parameters. These are:
   * The linear model parameters , , and ; and
   * The simulation counts: , the number of labs; , the number of individuals per lab; , the number of potential antigenic distance values to consider; and , the number of strains each lab should choose in addition to the homologous strain.
2. For lab , generate all titers for individual . Repeat for all individuals, then repeat for all labs.
3. For lab , calculate the current metrics. Repeat for all labs.
4. For lab , fit the linear model. Get the predictions for each observation, and calculate our proposed metrics.
5. Compare the spread of each metric across all of the labs.

Note that since the observations are exchangeable under the model assumptions we made for the simple model, we could compute the simulations in a more computationally efficient way (generate all random titers for each mean, and then label them as if we generated them sequentially for each lab/individual) but for now we will pretend that we cannot do that.

In due time we will need to vary the parameters of the simulation in order to gain a better understand of our metrics. But for now we will do only use one set of parameters as a proof of concept.

PUT A TABLE OF PARAMETERS HERE.

## Naive continuous data

For the first simulation, our titer observations will be normally distributed random variates, assuming only that titers are lognormally distributed and we can somehow measure them exactly. We will make this assumption more realistic in a few iterations of the simulation study. Since we can easily modified the generated titers with simple post-processing, starting with the continuous variates gives us “ideal data” that we can manipulate later.

# Step one set simulation parameters  
  
# This function will take the simulation count parameters and set up the  
# universe of possible strains, sample K different lab panels, and then  
# implement a generative model specified by the function argument sim\_fun.  
one\_sim <- function(K, J, M, n, sim\_fun, ..., seed = 370) {  
 ## seed: integer value used to set the random seed  
 ## K: the number of different "lab groups", i.e. the number of panels that  
 ## will be sampled.  
 ## J: the number of strains each lab should test. Right now this only accepts  
 ## a single number  
   
 set.seed(seed)  
   
 # Get the universe of strains  
 D <- seq(0 + 1 / M, 1, length.out = M)  
   
 # Step two run the simulation  
 sim\_data <-  
 tibble::tibble(  
 lab = 1:K,  
 panel = list(D)  
 ) |>  
 # Get the panel subset for each lab  
 dplyr::mutate(  
 panel = purrr::map(  
 panel,   
 \ (p) c(0, sample(p, size = J, replace = FALSE)) |> sort()  
 )  
 ) |>  
 # Add individual IDs for each lab  
 tidyr::expand\_grid(id = 1:n) |>  
 # Unnest the panel  
 tidyr::unnest(panel) |>  
 dplyr::rename(d = panel)  
   
 # Invoke sim\_fun to actually implement a generative model -- can swap  
 # out this function to get different behavior. Its arguments are passed  
 # in via ... argument.  
 # TODO should probably change to a list of arguments that gets  
 # tidy evaluated into it  
 gen\_model <-  
 sim\_data |>  
 sim\_fun(...)  
   
 return(gen\_model)  
}  
  
generate\_lm\_data <- function(sim\_data, alpha, beta, sigma, yfun = identity) {  
 out <-  
 sim\_data |>  
 # Simulate the titer values  
 dplyr::mutate(  
 mu = alpha + beta \* d,  
 y\_sim = rnorm(dplyr::n(), mean = mu, sd = sigma),  
 # Apply a post-processing function to y, e.g. flooring and censoring.  
 # Default is identity for no transformation.  
 y\_obs = yfun(y\_sim),  
 # Apply the inverse transformation to get y-values on the natural scale.  
 y\_nat = 5 \* (2 ^ y\_obs)  
 )  
   
 return(out)  
}  
  
ex\_sim <- one\_sim(  
 ## Counting parameters  
 K = 10,  
 J = 9,  
 M = 20,  
 n = 100,  
 # Generating model function  
 sim\_fun = generate\_lm\_data,  
 # Linear model parameters  
 alpha = 4,  
 beta = -3,  
 sigma = 0.5  
)

I decided to organize the code into a set of modular functions that can be used as arguments in a nested way. So there is the top-level function, one\_sim(), that controls the actual simulation parameters like the number of labs and the number of strains per lab. Eventually this will be extended to be a bit more flexible than it is now.

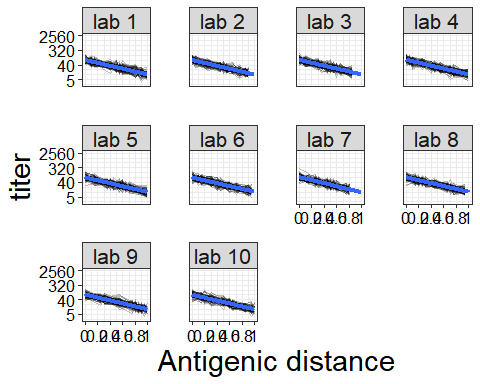
The function one\_sim() generates the universe of possible antigenic distance values, and gets a sample for each lab. It also sets up the infrastructure for simulating a titer for each value. Once this infrastructure is set up, this function invokes a functional argument, sim\_fun. For now, I’ve only implemented one function intended to be used here. This function represents the “data generating” part of the model. So here is where the part that actually draws titers from a linear model would go. In the future there should be more functions to use here to generate data from, e.g., a hierarchical model or an intercept-only model, or some kind of nonlinear model.

The data-generating function itself also takes a functional argument, called yfun. The actual argument here will be invoked taking the generated y values as its argument, and is intended to perform some kind of post-transformation that occurs when we observe the titers during the data-generating process. The default here is identity() which returns the same values for y. However, we could, for example, pass floor() here to round down the observed titers, or we could pass a custom function that implements the lower limit of detection.

Next we may want to visualize the simulation results. The function below will plot the trajectories (w.r.t distance) for each simulated individual, separated by the lab group (so all individuals with the same virus panel are plotted on the same plot). The yvar argument is tidy evaluated and so accepts a bare name of a variable from sim\_df, intended to be y\_sim, y\_obs, or y\_nat. The scale argument should be identity for y\_sim or y\_obs, and should be log for y\_nat.

Making this plot is an easy way to check that the simulation produced sensible values.

plot\_sim\_over\_time <- function(sim\_df, yvar, scale = "identity") {  
 plt <- sim\_df |>  
 ggplot() +  
 aes(x = d, y = {{yvar}}, group = id)  
   
 plt <- plt +  
 geom\_line(alpha = 0.25) +  
 geom\_smooth(  
 method = "lm", formula = "y~x",  
 lwd = 1.5,  
 aes(group = 1), fullrange = TRUE  
 ) +  
 facet\_wrap(~lab, labeller = \(x) label\_both(x, sep = " ")) +  
 labs(  
 x = "Antigenic distance",  
 y = "log2(titer / 5)"  
 ) +  
 scale\_x\_continuous(  
 breaks = seq(0, 1, 0.2),  
 minor\_breaks = seq(0, 1, 0.1),  
 labels = seq(0, 1, 0.2),  
 limits = c(0, 1)  
 )  
   
 if (scale == "log") {  
 plt <- plt +  
 ggplot2::scale\_y\_continuous(  
 breaks = 5 \* (2 ^ c(0, 3, 6, 9)),  
 minor\_breaks = 5 \* (2 ^ seq(-1, 9, 1)),  
 labels = 5 \* (2 ^ c(0, 3, 6, 9)),  
 limits = c(5 \* 2^(-1), 5 \* 2^9),  
 trans = "log2"  
 ) +  
 ylab("titer")  
 } else if (scale != "identity") {  
 stop('The only accepted arguments for "scale" are "identity" and "log".')  
 }  
   
 return(plt)  
}  
  
plot\_sim\_over\_time(ex\_sim, y\_nat, "log")



OK, now we want to calculate the metrics of interest. That means we need to fit the linear model ourselves, so that we don’t have to rely on whatever ggplot is doing (fitting a linear model, but this gives us more control and allows us to extract results). In the future, we might want to generalize this fitting process, but for now I’ll make this specific to a linear model.

This will return a version of the simulation data that is nested, so there is a data frame of individual observations for each lab. The fitted linear model is included in the output, along with the predictions on the observed data, and the predictions on an interpolated data set. In order to save memory, the model predictions and the original model data **are consolidated** in the lm\_preds argument (in the language of broom, this argument contains the model-augmented data frame).

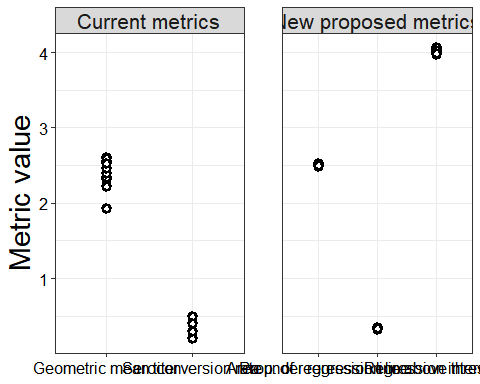
get\_lm\_fit <- function(sim\_data, ...) {  
 lm\_fits <-  
 sim\_data |>  
 tidyr::nest(model\_data = -lab) |>  
 dplyr::mutate(  
 lm\_fit = purrr::map(model\_data, \(x) lm(formula = y\_obs ~ d, data = x)),  
 lm\_stats = purrr::map(lm\_fit, \(x) broom::tidy(x)),  
 # Get the predictions on the observed data points  
 lm\_preds = purrr::map2(lm\_fit, model\_data, \(x, y) broom::augment(x, y)),  
 # Get the predictions on the interpolated x grid  
 lm\_interp = purrr::map(  
 lm\_fit,  
 \(x, y) broom::augment(x, newdata = tibble::tibble(d = seq(0, 1, 0.01)))  
 )  
 ) |>  
 dplyr::select(-model\_data)  
 return(lm\_fits)  
}  
  
lm\_fits <- get\_lm\_fit(ex\_sim)

Next we need to calculate the metric set. Other than writing a quick function to get the geometric mean, this is fairly straightforward.

geo\_mean <- function(x, na.rm = FALSE, ...) {  
 if (!is.numeric(x)) {  
 stop("All inputs must be numeric.")  
 }  
   
 if (isTRUE(na.rm)) {  
 x <- na.omit(x)  
 }  
   
 out <- exp(mean(log(x)))  
 return(out)  
}  
  
get\_metrics <- function(lm\_fit\_df) {  
 out <-  
 lm\_fit\_df |>  
 dplyr::mutate(  
 #AUC Calculation -- no weighting  
 AUC = purrr::map\_dbl(  
 lm\_interp,  
 # Calculate the pointwise AUC using the trapezoidal method  
 \(d) pracma::trapz(d$d, d$.fitted)  
 ),  
 # Intercept of line  
 intercept = purrr::map\_dbl(  
 lm\_interp,  
 \(d) d$.fitted[d$d == 0]  
 ),  
 # percent of values above threshold (3)  
 p40 = purrr::map\_dbl(  
 lm\_interp,  
 \(d) mean(d$.fitted >= 3)  
 ),  
 # seroconversion rate  
 scr = purrr::map\_dbl(  
 lm\_preds,  
 \(d) mean(d$.fitted >= 3)  
 ),  
 # geometric mean titer  
 gmt = purrr::map\_dbl(  
 lm\_preds,  
 \(d) geo\_mean(d$.fitted)  
 )  
 )  
   
 return(out)  
}  
  
sim\_metrics <- get\_metrics(lm\_fits)

So now we have a data frame that has all of the metrics for each lab. Now we want to analyze whether our new proposed metrics have less variability than the older metrics. Of course, a quick check is to plot these. The current format of the data is not suitable for plotting so we’ll write both a quick plotting function and a data reshaping function to make the plot look nice.

reshape\_metrics\_data <- function(metrics\_df, ...) {  
 out <-  
 metrics\_df |>  
 dplyr::select(  
 lab,  
 AUC,  
 intercept,  
 p40,  
 scr,  
 gmt  
 ) |>  
 tidyr::pivot\_longer(-lab) |>  
 # Cleanup for plotting  
 dplyr::mutate(  
 which\_set = ifelse(  
 name %in% c("scr", "gmt"),  
 "Current metrics",  
 "New proposed metrics"  
 ),  
 name = dplyr::case\_match(  
 name,  
 "AUC" ~ "Area under regression line",  
 "intercept" ~ "Regression intercept",  
 "p40" ~ "Prop. of regression line above threshold",  
 "scr" ~ "Seroconversion rate",  
 "gmt" ~ "Geometric mean titer"  
 )  
 )  
   
 return(out)  
}  
  
plot\_lab\_metrics <- function(metrics\_df) {  
 plt <-  
 metrics\_df |>  
 reshape\_metrics\_data() |>  
 ggplot()  
   
 plt <- plt +  
 aes(x = name, y = value) +  
 geom\_point(  
 alpha = 1,  
 stroke = 2,  
 size = 2,  
 shape = 21,  
 col = "black",  
 fill = "transparent"  
 ) +  
 facet\_wrap(~which\_set, nrow = 1, scales = "free\_x") +  
 xlab(NULL) + ylab("Metric value")  
   
 return(plt)  
}  
  
plot\_lab\_metrics(sim\_metrics)



So for this toy example, we can see that it appears that our proposed metrics really do have less variability than the metrics that are currently used. We likely need to do some statistical calculations to confirm this, since all of the metrics are on different scales. And we also need to test this over a range of parameters.

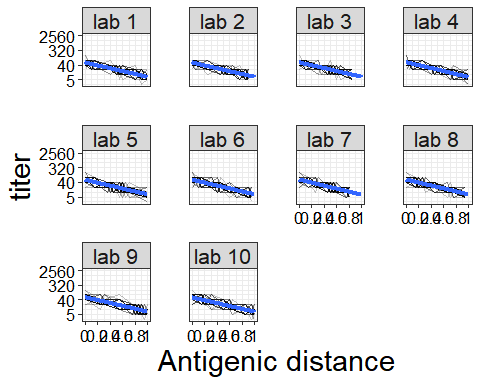
Before we test a lot of parameters, we also want to investigate what happens when we do not observe the entirely correct continuous titer value. We suspect that censored values made our data look bad on the real-world data, so we want to verify if a similar simulation starts to look bad if we interval censor and apply the LoD to the continuous titer data.

## Discrete data

First we’ll start with the interval-censoring. According to Jon Zelner, if there is no censoring then there should be an easy fix that helps to un-bias the results here, but hopefully we still see the same increase in precision. If our metrics stop being better at this stage, we need to try and account for interval censoring uncertainty in a statistical way.

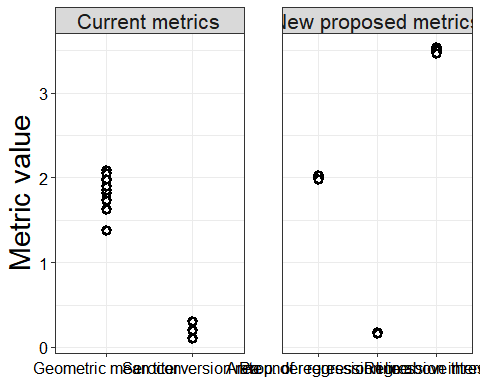
Anyways this will be a good idea to show what the code flow looks like without all the function definitions in the way.

ex\_discrete <- one\_sim(  
 ## Counting parameters  
 K = 10,  
 J = 9,  
 M = 20,  
 n = 100,  
 # Generating model function  
 sim\_fun = generate\_lm\_data,  
 # Y-data postprocessor  
 yfun = floor,  
 # Linear model parameters  
 alpha = 4,  
 beta = -3,  
 sigma = 0.5  
)  
  
plot\_sim\_over\_time(ex\_discrete, y\_nat, "log")



The effect of flooring the titers is immediately obvious, as there are now only a discrete number of overlapping values where the trajectories can fall. Note to self, maybe the alpha parameter should be lowered or adjustable since it should probably be tuned a bit for this plot. Let’s see what happens with the metrics.

lm\_discrete <- get\_lm\_fit(ex\_discrete)  
metrics\_discrete <- get\_metrics(lm\_discrete)  
plot\_lab\_metrics(metrics\_discrete)



OK, so based on the shift in the y-scale of this plot, we can see that there is definitely some change in the estimates even though this simulation is the same other than adding the flooring. This makes sense, if we round all of our values down then we are biasing our estimates down as well. However, regardless of that bias, we still see the same pattern in the variability.

TODO how to quantify variability? Just the variance/SD/MAD or something like that of the distribution of each metric?

## Data with limit of detection

TODO add a censoring example. we will need to do two examples here, one with the same parameters, and one with parameters where more censoring happens. For the second example we probably need to repeat all three simulations.

# Conclusions and future directions

* Future direction: parameter variation simulations. need to do probably quite a lot of these. This is probably the main thing we need to do next.
* Future direction: probably of the most importance would be the percentage of values that get censored (controlled by alpha and beta), and the variance. So I guess that’s all three linear model parameters. But it is feasible to me that by holding the means constant, changing the variance could lead to regimes where one set of metrics is better. Need toe xperiment.
* Future direction: implement and test more data-generating models.
* TODO adjust post-processing so that it can be applied afterwards instead of needing to regenerate the simulation (easy, just need a wrapper function).