Calculating breadth statistics

Zane Billings

2025-05-15

In this document, we will use an example to show how our summary statistics are calculated. Note that using our method requires adopting a Bayesian approach. **At this time, it is quite difficult to fit a frequentist model that simultaneously accounts for censoring and incorporates random effects.** Because the censoring correction seems to be incredibly important for our results, we *strongly* recommend using the Bayesian approach.

However, if you *must* use a frequentist method for some reason, you could apply frequentist mixed-effects models to our data and use bootstrapping produces to get confidence intervals for the statistics of interest. If you know of an R package which can fit mixed-effects models with censored outcomes, please let us know so we can update these instructions.

# Data description and format

The type of data that our methods apply to are panels of immunological assays to multiple different influenza strains. The following example data are a completely randomized, anonymized subset of the UGAFluVac HAI data from Ted Ross’s lab group. So our results in this document might not make sense, but they are used for a calculation example, not to draw any conclusions. [Table 1](#tbl-ex) shows the example dataset before preprocessing. We do not include any pre-titer or demographic information in these calculations right now.

We store the data in “long form” so that each immunological assay represents one record in the dataset (i.e., one row of data). Before we can fit any models there are several preprocessing steps that should be performed first.

1. Post-vaccination titer should be transformed to the log-scale. We always normalize the HAI titers based on the limit of detection (LoD) of 10 also, so that the values coded as 5 (below LoD) become on the log scale. If is the natural scale HAI titer, like the ones shwon in [Table 1](#tbl-ex), you can get the log-scale values using the transformation
2. The antigenic distance should be normalized. We use min-max normalization[[1]](#footnote-20) to ensure that the range of the datapoints is transformed to . If is a raw distance value like you see in [Table 1](#tbl-ex), and is the normalized antigenic distance, the formula for min-max normalization is

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 1: Example uncleaned dataset for analysis.   | ID | Assay Strain | Post-vac HAI | Ag Distance | | --- | --- | --- | --- | | 01 | HK/68 | 5 | 5.37 | | 01 | PC/73 | 5 | 6.23 | | 01 | TX/77 | 5 | 6.29 | | 01 | MI/85 | 5 | 4.78 | | 01 | Sich/87 | 20 | 4.68 | | 01 | Shan/93 | 40 | 4.55 | |  |  |  |  | | 83 | Uru/07 | 1280 | 0.50 | | 83 | Per/09 | 2560 | 0.35 | | 83 | Vic/11 | 2560 | 0.70 | | 83 | TX/12 | 2560 | 0.87 | | 83 | Switz/13 | 2560 | 0.88 | | 83 | HK/14 | 1280 | 0.00 | |

[Table 2](#tbl-ex-process) shows the same example data after processing.

We also recommend visualizing the data before proceeding to ensure there are no issues. Since the HAI titers are integer-valued, we apply a small amount of random jitter to the data before plotting[[2]](#footnote-22).

As you can see in [Figure 1](#fig-all-ls), the individual landscapes are extremely messy. This is due in part to the inherent measurement error in HAI titers, along with the variance induced by strain-specific effects (i.e. some strains tend to work better with HAI than others, regardless of how many antibodies are in the serum sample). The messy appearance of these individual landscapes **is to be expected.** That is why we need to fit a summary curve.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 2: Example pre-processed dataset for analysis.   | ID | Assay Strain | Post-vac HAI | Ag Distance | | --- | --- | --- | --- | | 01 | HK/68 | 0 | 0.85 | | 01 | PC/73 | 0 | 0.99 | | 01 | TX/77 | 0 | 1.00 | | 01 | MI/85 | 0 | 0.76 | | 01 | Sich/87 | 2 | 0.74 | | 01 | Shan/93 | 3 | 0.72 | |  |  |  |  | | 83 | Uru/07 | 8 | 0.08 | | 83 | Per/09 | 9 | 0.06 | | 83 | Vic/11 | 9 | 0.11 | | 83 | TX/12 | 9 | 0.14 | | 83 | Switz/13 | 9 | 0.14 | | 83 | HK/14 | 8 | 0.00 | |

# Coding censored data

We still have one more essential data cleaning step, however. We will fit all of our Bayesian models using the accessible and performant brms package. However, brms requires us to code our censored outcome in a specific way, which is documented in the [brmsformula information](https://paulbuerkner.com/brms/reference/brmsformula.html).

We know that our HAI data has both left-censored and interval-censored values, and no completely observed (un-censored). [Figure 2](#fig-censoring-mechanism) shows an example of the censoring mechanism for HAI. Other immunological assays will have different mechanisms for censoring, so this particular step only applies to HAI.

|  |
| --- |
| Figure 1: Transformed titer vs. normalized antigenic distance. Each point represents one HAI titer measurement and each line connects all of the measurements for a particular subject. |

|  |
| --- |
| Figure 2: Censoring mechanisms that apply to the latent or underlying continuous titers (left panel) during the measurement process to produce interval censored observations (middle panel), which are binned continuous values, and left censored observations (right panel), which are below the limit of detection. |

Coding censored data with mixed censoring types in brms format is a bit annoying. Instead of having one posttiter variable, we need to have three variables in the data which are called y, c, and y2 in the brms documentation. Since the data below the LoD are coded as 0, but the actual LoD is 10 (or 1 in the transformed HAI data), we have to do some cleanup.

1. The variable c is an indicator for the censoring status. We set it to left if our value is left censored, which occurs when the posttiter is , and we set it to interval otherwise, which is all of our other observations.
2. The variable y should be coded as the lower limit of detection (i.e., on our transformed scale) for left-censored observations, which is easy, because we already did that in the HAI processing transformation. For interval-censored observations, y needs to be the lower bound of the interval. Since HAI titers are always rounded down as part of the data generating mechanism, our outcome variable is already correctly coded, so we don’t need to do anything else here.
3. The variable y2 represents the upper bound for an interval censored observation, so for left-censored observations it should be the same as y, but for interval-censored observations, it should be y + 1.

If you use R and tidyverse, you can use this code for cleaning up the censored data[[3]](#footnote-34). In this code, titer\_data is the dataset shown in [Table 2](#tbl-ex-process), and posttiter is the name of the column containing the HAI data.

titer\_data\_cens <-  
 titer\_data |>  
 dplyr::mutate(  
 c = ifelse(posttiter == 0, "left", "interval"),  
 y = ifelse(c == "left", 1, posttiter),  
 y2 = ifelse(c == "left", 1, posttiter + 1)  
 )

[Table 3](#tbl-hai-cens) shows the dataset with the variables correctly encoded for the censoring correction.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 3: The cleaned dataset with the new columns c, y, and y2 that must be passed to brms to correctly apply the likelihood correction for censoring.   | id | assay\_strain | posttiter | distance | c | y | y2 | | --- | --- | --- | --- | --- | --- | --- | | 01 | HK/68 | 0 | 0.85 | left | 1 | 1 | | 01 | PC/73 | 0 | 0.99 | left | 1 | 1 | | 01 | TX/77 | 0 | 1.00 | left | 1 | 1 | | 01 | MI/85 | 0 | 0.76 | left | 1 | 1 | | 01 | Sich/87 | 2 | 0.74 | interval | 2 | 3 | | 01 | Shan/93 | 3 | 0.72 | interval | 3 | 4 | |  |  |  |  |  |  |  | | 83 | Uru/07 | 8 | 0.08 | interval | 8 | 9 | | 83 | Per/09 | 9 | 0.06 | interval | 9 | 10 | | 83 | Vic/11 | 9 | 0.11 | interval | 9 | 10 | | 83 | TX/12 | 9 | 0.14 | interval | 9 | 10 | | 83 | Switz/13 | 9 | 0.14 | interval | 9 | 10 | | 83 | HK/14 | 8 | 0.00 | interval | 8 | 9 | |

Now that the data are processed and ready to use we can calculate the statistics of interest.

# Math notation

We’ll use the following notation in all of our math formulas.

* indexes study subjects, where represents the first study subject and represents the sample size.
* indexes the different assay strains. Here, is the homologous strain, and are the heterologous strains – the particular order of the strains doesn’t matter.
* is the antigenic distance between strain and the vaccine strain. Thus, for the homologous strain and for every other .

We also define seroprotection and seroconversion as follows:

and

Note that on the log-scale, we have

# Unweighted statistics

First we want to calculate the statistics that don’t take antigenic distance into account. The formulas **without taking censoring into account** are as follows. All are simple to calculate and can be done in a spreadsheet or with basic R code.

1. **Magnitude**: measured by the geometric mean titer to the homologous strain:
2. **Breadth**: measured by the seroconversion rate across all strains. Note that since seroconversion status is binary, taking the mean gives us a proportion:
3. **Strength**: measured by the geometric mean titer across all of the observed strains:

**However, when we take censoring into account we actually need to fit a model.**

## GMT0

Note that when we fit a regression model that *only has an intercept term*, say

the estimate of actually estimates , the mean of the response variable. Since we can easily apply a censoring correction to a regression model, fitting an intercept-only regression model provides an easy way to estimate the mean while correcting for censoring.

To estimate , we need to fit an intercept-only regression model to the post-vaccination titer data for only the homologous strain. We can do this in brms while applying a censoring correction like this (make sure to run library(brms) at the top of your R script!).

titer\_data\_homologous <- dplyr::filter(titer\_data\_cens, distance == 0)  
  
gmt\_0\_model <- brms::brm(  
 formula = y | cens(c, y2) ~ 1,  
 data = titer\_data\_homologous,  
 family = gaussian,  
 prior = c(  
 brms::prior(student\_t(3, 0, 3), class = "Intercept"),  
 brms::prior(student\_t(3, 0, 3), class = "sigma")  
 ),  
 cores = 4,  
 chains = 4,  
 warmup = 250,  
 iter = 1250,  
 backend = "cmdstanr",  
 seed = 213948  
)

A full tutorial on the brms package is beyond the scope of our instructions, but excellent tutorials can be found [on the brms package website](https://paulbuerkner.com/brms/). We will provide a brief explanation of the arguments we chose.

* formula: follows the brms syntax for censoring, which is y | cens(c, y2) as explained earlier. The ~ separates the response terms from the predictor terms, and having 1 as the only predictor fits a regression model with only an intercept.
* data: the data we want to fit the model to goes here. All of the variables mentioned in formula have to match a column in data.
* family: “gaussian” indicates that we want to fit a standard regression model where the error term has a normal (or Gaussian) distribution.
* prior: since we are using a Bayesian model, we have to set priors. Setting priors can be contentious, and in general you can use whichever priors you prefer. We advocate for using generic student\_t(3, 0, 3) priors for these models, because they allow for large parameter estimates if they are supported by the data, but they prefer to be skeptical and assume that parameters are small, similar to the how we assume a null hypothesis is true and try to falsify it in frequentist hypothesis testing.
* cores: if your computer has multiple cores (most modern computers have at least 4), you can run multiple model chains in parallel to speed up the model fitting.
* chains: each “chain” of the model starts from a different initial condition and then tries to sample from the posterior distribution. Running multiple chains and making sure they agree provides evidence that we are sampling from the whole posterior distribution. Most bayesian statisticians recommend running at least 4 chains per model.
* warmup: the number of samples the model should draw in the warmup step, when it is calibrating. These will not be used for inference, but if you don’t run enough it will degrade the quality of your model fit.
* iter: the number of total samples the model should draw (per chain), so the number of post-warmup samples you will use for inference is iter - warmup.
* backend: we strongly recommend using the [cmdstanr](https://mc-stan.org/cmdstanr/articles/cmdstanr.html) package here, because it is more modern and has more tools available for use than the default rstan package. However if you have difficulties installing cmdstanr, you can use rstan instead which is easier to set up.
* seed: you have to specify this to make sure your results are exactly reproducible. It can be any integer number, but different values will result in different random number draws and thus slightly different results.

We should also check the model diagnostics by looking at the summary. Again, we won’t do an in-depth explanation of Bayesian model diagnostics here, but typically you want all of the Rhat estimates to be 1.00 (sometimes 1.01 can be ok if you are doing a quick check), and you want both bulk ESS and tail ESS to be above 500 (or above 1000 if you need precise estimates of quantiles or variances).

summary(gmt\_0\_model)

Family: gaussian   
 Links: mu = identity; sigma = identity   
Formula: y | cens(c, y2) ~ 1   
 Data: titer\_data\_homologous (Number of observations: 83)   
 Draws: 4 chains, each with iter = 1250; warmup = 250; thin = 1;  
 total post-warmup draws = 4000  
  
Regression Coefficients:  
 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS Tail\_ESS  
Intercept 5.19 0.16 4.87 5.51 1.00 2291 2578  
  
Further Distributional Parameters:  
 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS Tail\_ESS  
sigma 1.49 0.13 1.27 1.77 1.00 2572 2569  
  
Draws were sampled using sample(hmc). For each parameter, Bulk\_ESS  
and Tail\_ESS are effective sample size measures, and Rhat is the potential  
scale reduction factor on split chains (at convergence, Rhat = 1).

We see that our model passes those quick diagnostic guidelines. We’ll skip these checks for the rest of these instructions because I already checked the models, but whenever you fit your bayesian models, you need to check these diagnostics. If any of the checks don’t pass, you should try increasing warmup (above 500 typically is not useful, though) and iter (increase this one as much as needed until you reach the diagnostic targets). More troubleshooting advice is easy to find on the [Stan discourse forum](https://discourse.mc-stan.org/).

Now with the arguments explain we can use this code to extract the posterior samples of the intercept, which correspond to the posterior samples of the GMT. Then we can summarize the posterior samples into a point estimate and credible interval for our metric[[4]](#footnote-41).

gmt\_0\_samples <-  
 gmt\_0\_model |>  
 tidybayes::tidy\_draws() |>  
 dplyr::select(b\_Intercept)  
  
tidybayes::mean\_hdci(gmt\_0\_samples) |>  
 # This step back-transforms the log-scale estimates to the natural scale  
 dplyr::mutate(  
 dplyr::across(c("b\_Intercept", ".lower", ".upper"), \(x) 5 \* 2 ^ x)  
 )

# A tibble: 1 × 6  
 b\_Intercept .lower .upper .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 183. 146. 227. 0.95 mean hdci

So our censoring-corrected estimate of the GMT0 metric is with a 95% CI of .

## SCR

Now that we’ve estimated the , the other parameters will be simpler because almost everything stays the same. Because all seroconversion measurements are binary (they are either for not seroconverted or for seroconverted), several things are different about this model.

First we need to calculate the seroconversion status, which is easy. **However, you must have the pre-vaccination titers**. We have them for our example data, but if you don’t have pre-vaccination data, you can only look at seroprotection, not seroconversion. Note that we calculate the fold change (ratio of post-vaccination titer divided by pre-vaccination titer) using a difference instead of a division because the log of a fraction is equal to the difference of the logs[[5]](#footnote-43).

titer\_data\_sc <-  
 titer\_data\_cens |>  
 tibble::add\_column(pretiter = log2(example\_data\_pretiters / 5)) |>  
 dplyr::mutate(  
 fold\_change = posttiter - pretiter,  
 seroprotection = posttiter >= 3,  
 seroconversion = seroprotection \* (fold\_change >= 2)  
 )

We don’t have to correct for censoring, and we need to fit a logistic regression model instead of a linear regression model. Even though we don’t apply censoring corrections here, **we choose to fit this with a bayesian model anyways to ensure our CI’s are comparable!** In the code for this model, notice that we only have one prior – because logistic regression models don’t have a residual variance parameter, we don’t need a prior for it.

scr\_model <- brms::brm(  
 formula = seroconversion ~ 1,  
 data = titer\_data\_sc,  
 family = bernoulli("logit"),  
 prior = c(  
 brms::prior(student\_t(3, 0, 3), class = "Intercept")  
 ),  
 cores = 4,  
 chains = 4,  
 warmup = 250,  
 iter = 1250,  
 backend = "cmdstanr",  
 seed = 213948  
)

We still get the posterior samples of the intercept though.

scr\_samples <-  
 scr\_model |>  
 tidybayes::tidy\_draws() |>  
 dplyr::select(b\_Intercept)  
  
tidybayes::mean\_hdci(scr\_samples)

# A tibble: 1 × 6  
 b\_Intercept .lower .upper .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 -1.30 -1.43 -1.18 0.95 mean hdci

So our estimate of the seroconversion rate is with a 95% CI of .

## GMT

Estimating the GMT is exactly the same as estimating the GMT0, except we use the entire dataset instead of only the homologous strain measurements.

gmt\_model <- brms::brm(  
 formula = y | cens(c, y2) ~ 1,  
 data = titer\_data\_cens,  
 family = gaussian,  
 prior = c(  
 brms::prior(student\_t(3, 0, 3), class = "Intercept"),  
 brms::prior(student\_t(3, 0, 3), class = "sigma")  
 ),  
 cores = 4,  
 chains = 4,  
 warmup = 250,  
 iter = 1250,  
 backend = "cmdstanr",  
 seed = 213948  
)

We extract and summarize the posterior samples of the intercept, exactly the same as the GMT0.

gmt\_samples <-  
 gmt\_model |>  
 tidybayes::tidy\_draws() |>  
 dplyr::select(b\_Intercept)  
  
tidybayes::mean\_hdci(gmt\_samples) |>  
 # This step back-transforms the log-scale estimates to the natural scale  
 dplyr::mutate(  
 dplyr::across(c("b\_Intercept", ".lower", ".upper"), \(x) 5 \* 2 ^ x)  
 )

# A tibble: 1 × 6  
 b\_Intercept .lower .upper .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 65.9 59.8 73.0 0.95 mean hdci

So our censoring-corrected estimate of the GMT metric is with a 95% CI of .

Now, calculating the models with antigenic distance included is a bit more involved. The first step is to fit our antibody landscape model.

# Summary antibody landscape

The first step in calculating the antigenic-distance based metrics is to compute the **summary antibody landscape**. This is a mathematical model that takes all of the individual antibody landscapes from [Figure 1](#fig-all-ls) as the input, and produces a study-level summary curve that represents the average antibody landscape in the study sample.

We use a multilevel linear regression (AKA mixed-effects regression) model to construct the summary antibody landscape. Letting the log postvaccination titers by , the model is as follows.

We have a population level intercept and slope for distance, and each individual also gets their own intercept and slope, which are correlated and drawn from a distribution that we can estimate. Because we are using a Bayesian model, we also have to specify priors. Here we use priors that are sensible for most similar problems, but discussing the parametrization is quite mathematically dense so not too useful here[[6]](#footnote-47), so just note that the prior we use for class “corr” is different because of this parametrization.

summary\_landscape\_model <-  
 brms::brm(  
 formula = y | cens(c, y2) ~ 1 + distance + (1 + distance | id),  
 data = titer\_data\_cens,  
 family = gaussian,  
 prior = c(  
 brms::prior(student\_t(3, 0, 3), class = "Intercept"),  
 brms::prior(student\_t(3, 0, 3), class = "b"),  
 brms::prior(student\_t(3, 0, 3), class = "sd"),  
 brms::prior(student\_t(3, 0, 3), class = "sigma"),  
 brms::prior(lkj(2), class = "cor")  
 ),  
 cores = 4,  
 chains = 4,  
 warmup = 250,  
 iter = 1250,  
 backend = "cmdstanr",  
 seed = 213948  
 )

Note that in the above code, the formula part 1 + distance adds the study-level (fixed effects) parameters for the intercept and the antigenic distance slope, and the part (1 + distance | id) adds the individual-level (random effects) parameters for each individual id.

Now that we fit the model (and I made sure to check that the diagnostics are sufficient as mentioned earlier) we want to extract the summary antibody landscape fit. Because we used a multilevel model, there are many ways to get predictions and uncertainty about the best antibody landscape that summarizes the population level. However, we chose to use the global grand mean prediction[[7]](#footnote-48) and associated HDCI.

**Make sure to keep the entire set of posterior predictions, because the AUC and other landscape metrics are calculated from the posterior samples.** (That’s summary\_landscape\_preds in our code below.) Also note the variable h that we assign below – this is called the step size for our interpolated predictions. Making h smaller will result in more fine-grained predictions and thus more accurate metric calculations, but the storage space and memory requirements for the posterior samples increase rapidly as h decreases. We have found that h = 0.01 offers a good balance between memory usage and prediction accuracy.

h <- 0.01  
  
summary\_landscapes\_preds <-  
 summary\_landscape\_model |>  
 tidybayes::epred\_draws(  
 newdata = data.frame(distance = seq(0, 1, h)),  
 re\_formula = NA  
 ) |>  
 dplyr::ungroup()  
  
summary\_landscape\_intervals <-  
 summary\_landscapes\_preds |>  
 dplyr::summarize(  
 tidybayes::mean\_hdci(.epred),  
 .by = distance  
 )

Now we’ll take our previous plot of the individual antibody landscapes and add the summary antibody landscape on top ([Figure 3](#fig-sum-landscape)).

We’re currently experimenting with nonlinear methods for fitting the summary antibody landscape, but right now this linear regression version appears to be doing ok. Now that we have the summary antibody landscape posterior samples, we can calculate the metrics based on the posterior samples.

# Landscape-based statistics

All three of the landscape-based statistics can be expressed as functions of the posterior predictions of the summary antibody landscape, so we’ll use that to calculate them.

|  |
| --- |
| Figure 3: Individual antibody landscapes (lines) and observed HAI data (points), with the study-level posterior mean summary landscape and 95% HDCI (blue line and ribbon). |

## Magnitude

We measure the magnitude of the summary landscape as the intercept of the line – recall that we only saved the posterior samples of the conditional predictions though, so we should express this as the average prediction when . That is,

where indexes the posterior samples from the model. Estimating this from the posterior samples is easy because all we have to do is average the predictions where .

summary\_landscapes\_preds |>  
 dplyr::filter(distance == 0) |>  
 dplyr::summarize(tidybayes::mean\_hdci(.epred))

# A tibble: 1 × 6  
 y ymin ymax .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 5.92 5.57 6.28 0.95 mean hdci

So we see that our estimate for the magnitude is with a credible interval of .

We can also backtransform this one to HAI units to compare it to the GMT0 estimate.

5 \* 2 ^ c(5.92, 5.57, 6.28)

[1] 302.7384 237.5238 388.5424

As you can see, this is actually larger than the estimate – accounting for the antigenic distance allows us to get a better estimate of the average homologous response.

## Breadth

Our estimate of the breadth is the intersection between the horizontal line (or on the natural scale of the HAI titers) and the summary antibody landscape. This estimates the antigenic distance where we expect post-vaccination HAI titers to drop below the clinical protection threshold. The first thing we need to do is for each sample find the predicted values that are closest to 3. Then we get the distance value associated with that prediction, and that is our measurement of the breadth.

summary\_landscapes\_preds |>  
 # minimizing |prediction - 3| over the draws will tell us which x-value is  
 # closest to a y-value of 3.  
 dplyr::mutate(  
 diff\_from\_threshold = abs(.epred - 3)  
 ) |>  
 dplyr::summarize(  
 # For each posterior sample (aka posterior draw) of the summary landscape,  
 # get the predicted y-value that is closest to 3.  
 closest = min(diff\_from\_threshold),  
 # Now get the x-value that's associated with that y-value -- this is the  
 # estimate for that posterior draw.  
 breadth = distance[which.min(diff\_from\_threshold)],  
 .by = .draw  
 ) |>  
 # Summarize the posterior samples into a point estimate and CI  
 dplyr::summarize(  
 tidybayes::mean\_hdci(breadth)  
 )

# A tibble: 1 × 6  
 y ymin ymax .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 0.613 0.56 0.66 0.95 mean hdci

So our estimate of the breadth for our example data is with a 95% CI of .

## Strength

The last metric we need to calculate is the AUC. Because we’re using a linear regression model, we could actually write out a formula to easily calculate the AUC from the intercept and slope samples. But because we want to generalize to potentially more complicated models, we need to do some numeric calculations (this is true for the breadth estimate as well).

Since we already have the interpolated posterior samples, we can calculate the numeric AUC using [any of the existing methods for numerical integration](https://en.wikipedia.org/wiki/Numerical_integration). The computationally fastest and simplest to implement is the trapezoidal rule, which has a computationally efficient implementation in the function pracma::trapz(). All we have to do is put in the interpolated distance values and their corresponding posterior predictions and this function estimates the area under the curve for us – we repeat this for each posterior sample of the summary landscape to get a credible interval.

summary\_landscapes\_preds |>  
 # Calculate the AUC for each posterior sample of the line  
 dplyr::summarize(  
 AUC = pracma::trapz(x = distance, y = .epred),  
 .by = .draw  
 ) |>  
 # And summarize it into a point estimate and interval  
 dplyr::summarise(  
 tidybayes::mean\_hdci(AUC)  
 )

# A tibble: 1 × 6  
 y ymin ymax .width .point .interval  
 <dbl> <dbl> <dbl> <dbl> <chr> <chr>   
1 3.54 3.31 3.78 0.95 mean hdci

So our estimate for the AUC for the example data is with a 95% CI of . Normally, the units for the AUC estimate are a product of the units for the x-variable and the units for the y-variable. But since we min-max scaled the x-variable, it is actually unitless – that means we can interpret the AUC as an average log-titer measurement over the antigenic distances. So, we can backtransform the AUC to the nominal HAI scale to get a better understanding.

[1] 58.15890 49.58831 68.68523

These numbers are on the same order of magnitude as the GMT estimates, but a bit smaller – it’s still kind of hard to directly interpret the “goodness” of this number, and it is best used for comparison across multiple vaccinces, panels of historical viruses, or subsamples.

# Contact info

Please contact Zane Billings (email: wesley.billings@uga.edu) or Andreas Handel (email: ahandel@uga.edu) for questions about these methods.

1. Min-max normalization is a linear and monotonic transformation, so it preserves all of the statistical properties we care about. Using min-max normalization on the antigenic distance makes sure that AUC values using different antigenic distance methods and different assay panels can be fairly compared against each other (as long as the outcome is HAI). Other normalization methods like -score standardization might be useful but we haven’t compared them. [↑](#footnote-ref-20)
2. If you use ggplot2 like we do, we recommend adding jitter manually rather than using ggplot2’s built-in jitter functionality. As of the time of writing, there is no way to have ggplot2’s built-in jitter apply the same jitter to both ends of a line and to a point. [↑](#footnote-ref-22)
3. If you prefer not to use tidyverse, the ifelse() statements in the code will still work, you just need to make sure to use a function like transform() or with(), or manually specify the dataset for each column every time, i.e. titer\_data$posttiter if you aren’t using a function that adds the column names to the scope. [↑](#footnote-ref-34)
4. We typically use the mean as the point estimate along with a 95% highest density continuous interval (HDCI), implemented through tidybayes::mean\_hdci(). You could also use the median or the mode (also called the MAP or Maximum *a posteriori* estimate) for the point estimate, and you could also use an equal-tailed credible interval. The choice depends on personal preference and interpretation. [↑](#footnote-ref-41)
5. That is, . [↑](#footnote-ref-43)
6. The Student priors are the same as those we used in the simpler models. However, the multivariate normal distribution can be problematic to parametrize, especially in terms of a prior for , the covariance matrix of the random effects. The best solution, and the one used by brms, is to use a Cholesky decomposition of the covariance matrix and apply a Lewandowski-Kurowicka-Joe prior to the Cholesky factor of . We recommend as a sensible default for the covariance Cholesky factor prior. [↑](#footnote-ref-47)
7. there is unfortunately no standard language used to describe these different methods, other than getting lost in describing different parts of the model as marginal or conditional. We adopt the terminology of Andrew Heiss, who provides an excellent layman-friendly explanation of this issue in [a blog post](https://www.andrewheiss.com/blog/2021/11/10/ame-bayes-re-guide/). [↑](#footnote-ref-48)