Calculating Serostatus Thresholds

2025-05-22

Introduction

The purpose of this document is to compute a threshold of serostatus when there is not an external threshold like there was with measles. This document describes how to use control samples with known prior exposure status to a pathogen (positive or never exposed is usually the case) to establish a threshold then describes how to calculate important performance metrics of this threshold (e.g. sensitivity and specificity). Upon completing this document you will be able to:

- Identify a threshold of seropositivity using well characterized control samples using negative controls, or using positive and negative controls together.
- Characterize the sensitivity and specificity of this threshold.

General housekeeping

Before we start, let's navigate to the appropriate working directory. You can accomplish this by navigating to the "Session" tab of Rstudio, and choosing "Set Working Directory" –> "Choose Directory" and using your file browser to navigate to the Data folder within the seroanalytics_workshop folder. Alternatively, you can modify the code below as appropriate for your files to get to the Data folder in the seroanalytics_workshop folder.

```r
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message=FALSE)

# set my_path to be the working directory location of where the *seroanalytics_workshop* folder is stor
my_path <- "/Users/soniahegde/Library/CloudStorage/OneDrive-JohnsHopkins/seroanalytics_workshop"

source(paste(my_path, "Source/utils.R", sep="/"))
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
## Warning: package 'tidyr' was built under R version 4.1.2

## Warning: package 'MASS' was built under R version 4.1.2

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## Warning: package 'mclust' was built under R version 4.1.2

## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

Reading in data

Note this is the same procedure as in lab 3 and lab 4.

```
control_data <- read.csv(paste(my_path, "Data/simulated_control_long_training_data.csv", sep="/"))
sample_data <- read.csv(paste(my_path, "Data/simulated_sample_wide_training_data.csv", sep="/"))

#this converts sample_data from a wide to a long dataframe. edit the column names to your data.
sample_long <- reshape(
  sample_data,
  varying = setdiff(names(sample_data), c("id", "age", "sex")),
  v.names = "mfi",
  timevar = "antigen",
  times = setdiff(names(sample_data), c("id", "age", "sex")),
  idvar = "id",
  direction = "long"
)
rownames(sample_long) <- NULL
```

Choosing a cutoff based on negative controls.

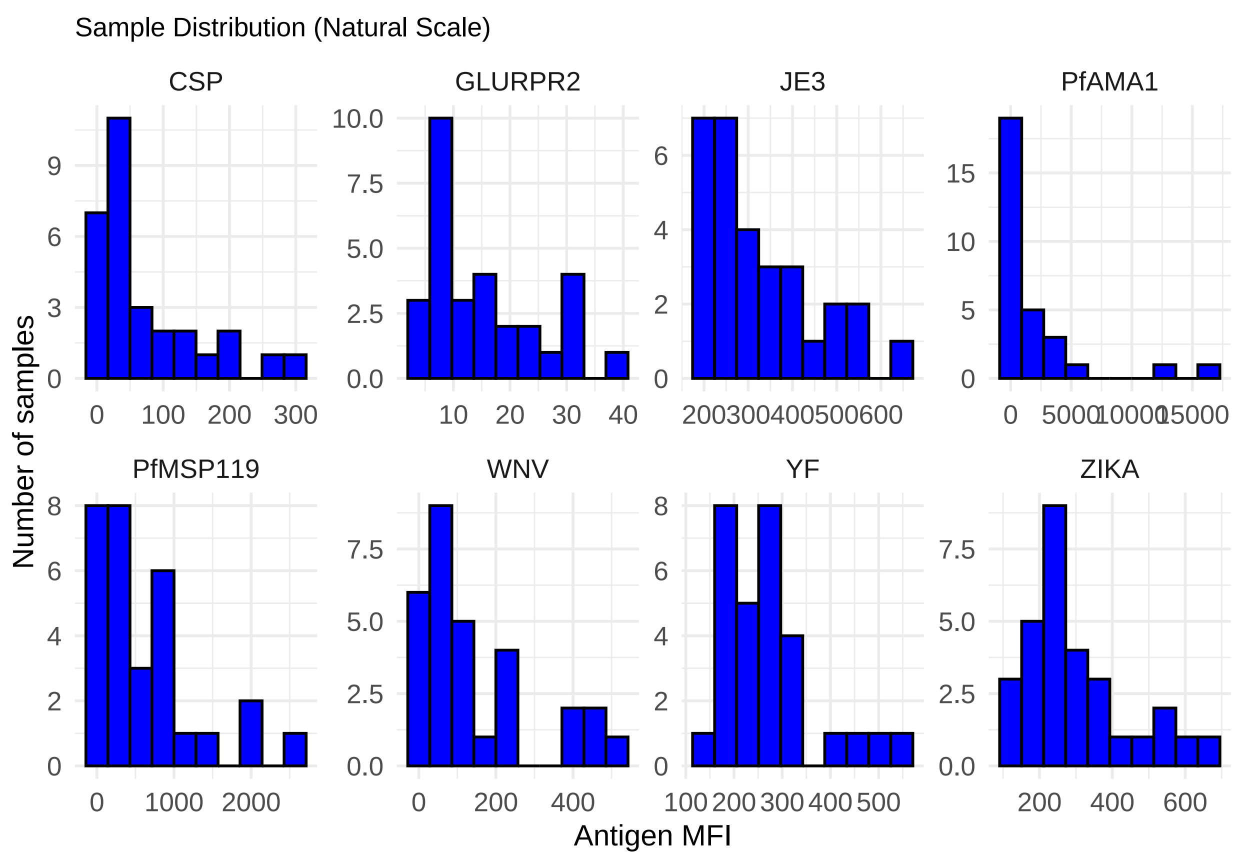
We begin by focusing on PfAMA1 responses in this section, feel free to explore other antigens after you have completed these steps.

We will begin by exploring the distribution of the negative controls. One way to determine a cutoff is to consider samples above distribution of negative controls to be positive. This can be particularly useful for long lasting markers of prior exposure that do not show signs of waning, as is the case for PfAMA1 (long lasting marker of prior malaria exposure). There are several ways one can consider the distribution of negative controls. The simplest way is to assign every sample that has a higher MFI value than the highest negative control to be positive. We can begin with this method.

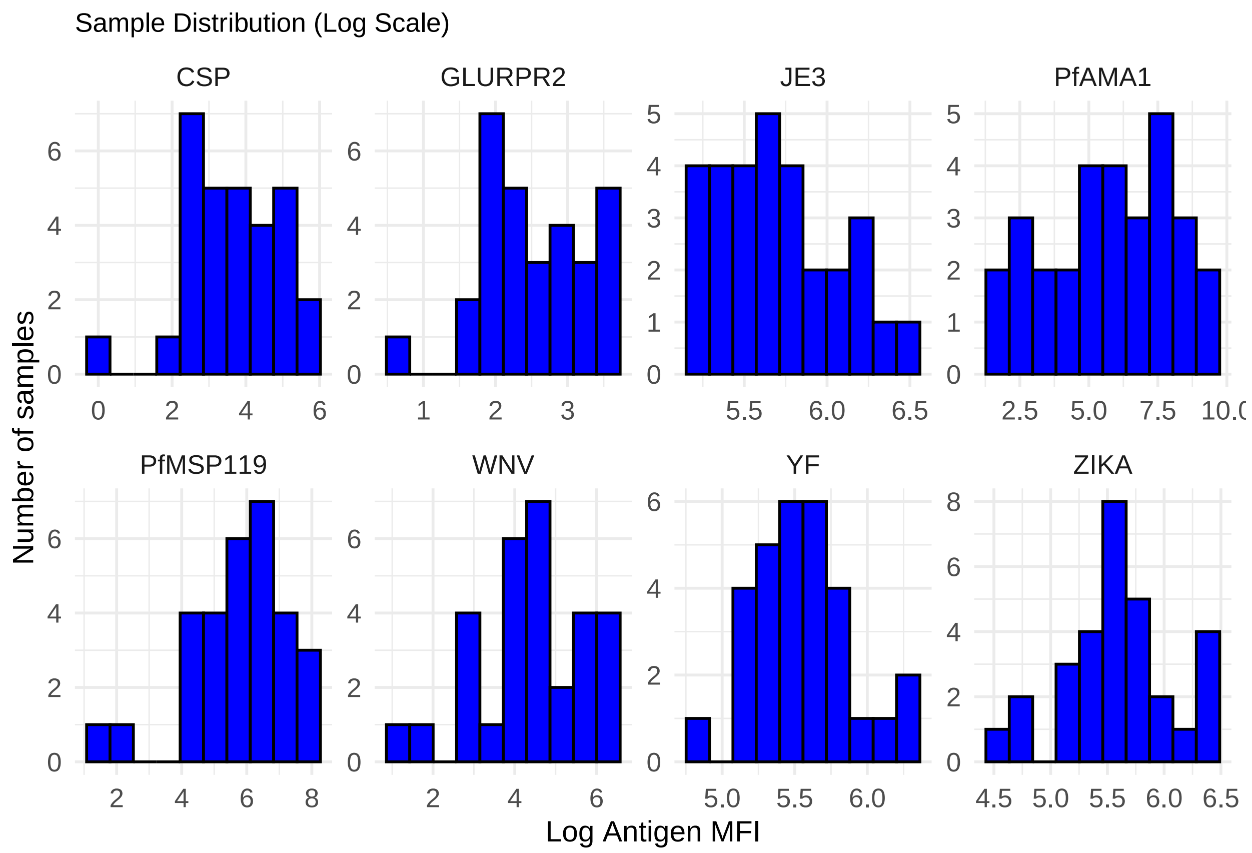
First, we can visuzlise the distribution of negative controls.

```
#restrict to negative controls
control_negs <- control_data[control_data$pos_neg=="negative",] #edit this line if needed

#check distributions of data - untransformed
neg_controls_natural_scale <- ggplot(control_negs, aes(x = mfi)) +
  geom_histogram(bins = 10, color = "black", fill = "blue") +
  facet_wrap(~ antigen, scales = "free", ncol = 4) +  # <- Set 5 columns per row
  labs(
    title = "Sample Distribution (Natural Scale)",
    x = "Antigen MFI",
    y = "Number of samples"
  ) +
  theme_minimal() +
  theme(
    strip.text = element_text(size = 10),  # Smaller font for facet labels
    axis.text = element_text(size = 10),  # Smaller font for axis text
    plot.title = element_text(size = 10)  # Smaller font for the plot title
  )
neg_controls_natural_scale
```



```
neg_controls_log_scale <- ggplot(control_negs, aes(x = log(mfi))) +
  geom_histogram(bins = 10, color = "black", fill = "blue") +
  facet_wrap(~ antigen, scales = "free", ncol = 4) +  # <- Set 5 columns per row
  labs(
    title = "Sample Distribution (Log Scale)",
    x = "Log Antigen MFI",
    y = "Number of samples"
  ) +
  theme_minimal() +
  theme(
    strip.text = element_text(size = 10),  # Smaller font for facet labels
    axis.text = element_text(size = 10),  # Smaller font for axis text
    plot.title = element_text(size = 10)  # Smaller font for the plot title
  )
neg_controls_log_scale
```



Now we can identify the cutoff as being the highest negative control.

```
highestNeg_cutoff<- max(control_negs$mfi[which(control_negs$antigen=="PfAMA1")])
```

Now, we might want to understand how good our cutoff is, one way to assess this is to compute the sensitivity and specificity under this cutoff. Recall, sensitivity is the proportion of true positives that are accurately classified as positive, and specificity is the proportion of true negatives that are accurately classified as negative. Note, we are able to calculate this because we have both positive and negative control samples, this may not always be the case.

```
#compute serostatus using the highest negative value
control_data_ama1<- control_data[which(control_data$antigen=="PfAMA1"),]
serostatus_highNeg<- ifelse(control_data_ama1$mfi>highestNeg_cutoff, 1, 0)

#add this serostatus variable to our original control dataset for pfAMA1
control_data_ama1WithSerostatus<- cbind(control_data_ama1,serostatus_highNeg)

#compute sensitivity
sens_ama1_highestNeg<- length(which(control_data_ama1WithSerostatus$pos_neg=="positive"&control_data_ama

sens_ama1_highestNeg
```

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

```
#compute specificity

spec_ama1_highestNeg<- length(which(control_data_ama1WithSerostatus$pos_neg=="negative"&control_data_ama

spec_ama1_highestNeg
```

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

Another common method that only relies on negative controls is to use a fixed number of standard deviations above the mean of the negative controls to establish a cutoff. In this example we choose 3 standard deviations above the mean of the negative controls. Note here, we will work on the log scale. We will check the same test metrics for this cutoff.

```
#calculate the cutoff
mean3sd_cutoff<- mean(log(control_data_ama1$mfi[which(control_data_ama1$pos_neg=="negative")]))+3*sd(log

#calculate serostatus
serostatus_mean3SD<- ifelse(log(control_data_ama1$mfi)>mean3sd_cutoff, 1, 0)




#add this serostatus variable to our original control dataset for pfAMA1
control_data_ama1WithSerostatus<- cbind(control_data_ama1WithSerostatus,serostatus_mean3SD)

#compute sensitivity
sens_ama1_mean3sd<- length(which(control_data_ama1WithSerostatus$pos_neg=="positive"&control_data_ama1Wi

sens_ama1_mean3sd
```

```
## [1] 0
```

```
#compute specificity

spec_ama1_mean3sd<- length(which(control_data_ama1WithSerostatus$pos_neg=="negative"&control_data_ama1Wi

spec_ama1_mean3sd
```

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

Exercise 1: What do you notice about the sensitivity and specificity when comparing these two approaches? Which approach would you choose?

Exercise 2:

Using code below, and the sample data (not control!) for pfAMA1, compute the seroprevalence and 95% confidence intervals using the highest negative cutoff, and then adapt the code to use the mean plus 3SD cutoff. Do you notice any differences?

```
#Applying cutoff:
  #ifelse function, if first statement is true, then outcome is set to 1, and if first statemnet is fal
  #since the first statement is a vector, then outcome will be a vector of 1's and 0's.
seropositivity <- ifelse(sample_data$PfAMA1 > highestNeg_cutoff, 1, 0) #1 indicates seropositive, and 0
```

```
#number of people seropositve and seronegative
cat("Table of number seronegative and seronegative", "\n")
```

```
## Table of number seronegative and seronegative
```

```
table(seropositivity,useNA="always")
```

```
## seropositivity
##    0    1 <NA>
##  469  531    0
```

```
#percent of people seropositive and negative
cat("Table of percent seronegative and seronegative", "\n")
```

```
## Table of percent seronegative and seronegative
```

```
round(prop.table(table(seropositivity,useNA="always")),3)*100
```

```
## seropositivity
##    0    1 <NA>
## 46.9 53.1  0.0
```

```
# Set your parameters
  x <- 531 #number seropositive. Get the number seropositive in your sample from the seropositive table
  n <- nrow(sample_data) #total number of samples in your data, note in these data we saw above there w
  conf <- 0.95 #confidence interval. 95% is a standard CI but you can adjust this if you want

  # Exact interval
  ci <- binom.exact(x, n, conf.level = conf) #epitools function
  #
  cat("CI lower", round(ci$lower,4)*100, "%, CI upper", round(ci$upper,4)*100, "%")
```

```
## CI lower 49.95 %, CI upper 56.23 %
```

Now we will use the positive and negative controls jointly to come up with a cutoff that seeks to maximize both sensitivity and specificity.First we will plot an ROC curve, this curve describes the relationship between sensitivity and specificity.

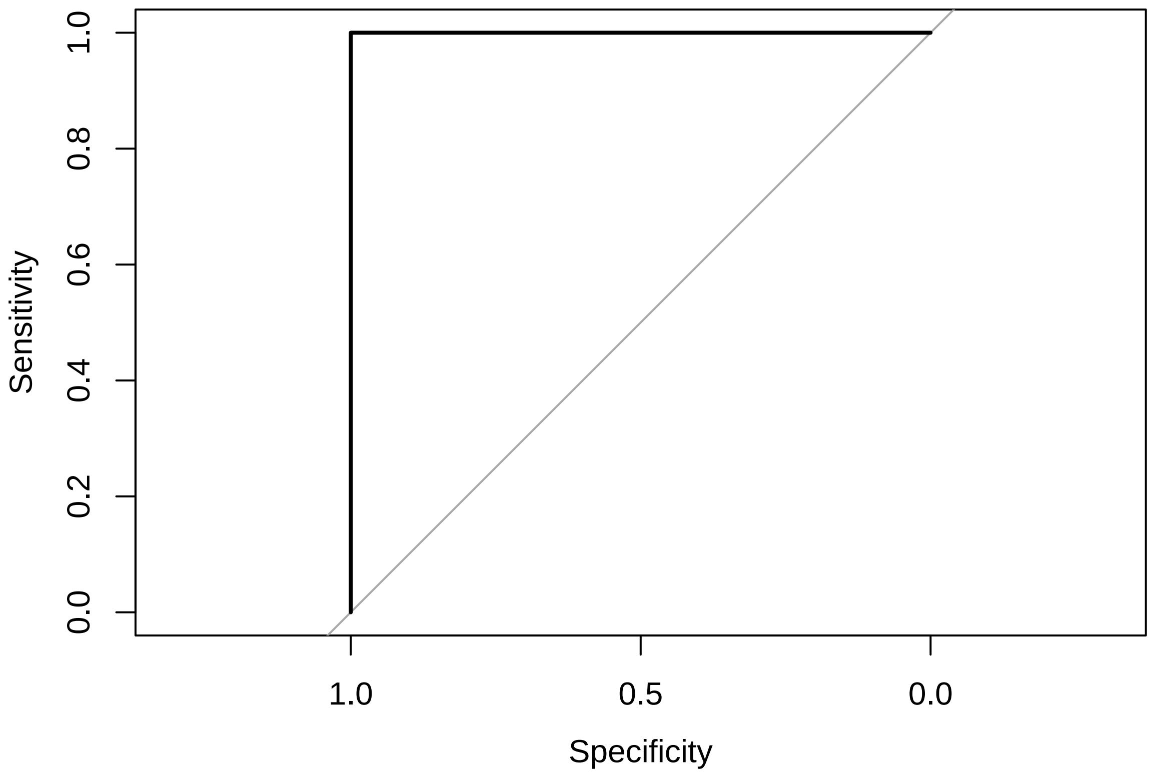
In general the ROC fixes a threshold for seropositivity and then computes sensitivity and specificity under that threshold. Then the method repeats this process for a different threshold, and explores a wide range of thresholds (in our case from -11.5 to 58767 MFI). In general for very low MFI thresholds, specificity is high, but sensitivity is low, but as the threshold increases specificity increases. However, at a certain level of threshold specificity starts to decrease.

```
#this fits the ROC curve
rocFit<- roc(response = control_data_ama1$pos_neg,
               predictor = control_data_ama1$mfi)
#look at the output

rocFit
```

```
##
## Call:
## roc.default(response = control_data_ama1$pos_neg, predictor = control_data_ama1$mfi)
##
## Data: control_data_ama1$mfi in 30 controls (control_data_ama1$pos_neg negative) < 30 cases (control_d
## Area under the curve: 1
```

```
plot(rocFit)
```



Commonly, we want to choose a threshold that maximizes both sensitivity and specificity. This is a quantity known as youden's J. Formally we want to choose the threshold for which the equation sensitivity+specificity-1 has the largest possible value. We will identify the optimal threshold under these conditions and compute sensitivity and specificity.

```
#first we compute all possible values of this under the different thresholds of the ROC
indices<- rocFit$sensitivities+rocFit$specificities-1

ROCThreshold= rocFit$thresholds[which(indices==max(indices))][1]

serostatus_ROC<- ifelse(control_data_ama1$mfi>ROCThreshold, 1, 0)



#add this serostatus variable to our original control dataset for pfAMA1
control_data_ama1WithSerostatus<- cbind(control_data_ama1WithSerostatus,serostatus_ROC)
```

```
#compute sensitivity
sens_ama1_ROC<- length(which(control_data_ama1WithSerostatus$pos_neg=="positive"&control_data_ama1WithSe

sens_ama1_ROC
```

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

```
#compute specificity

spec_ama1_ROC<- length(which(control_data_ama1WithSerostatus$pos_neg=="negative"&control_data_ama1WithSe

spec_ama1_ROC
```

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

How do these outputs compare to those from the previous cutoffs?

Exercise 3:

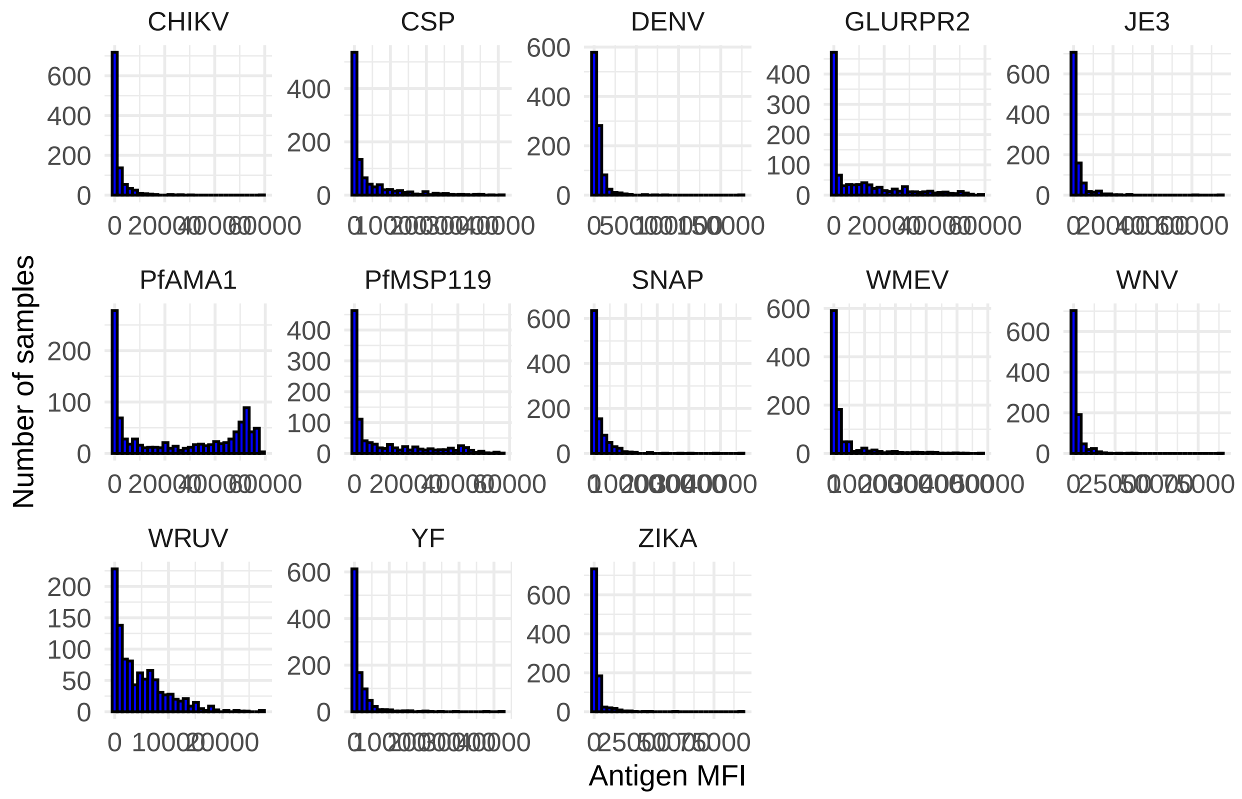
Using code from earlier in the lab, and the sample data (not control!) for pfAMA1, compute the seroprevalence and 95% confidence intervals using ROC cutoff. Do you notice any differences?

Optional

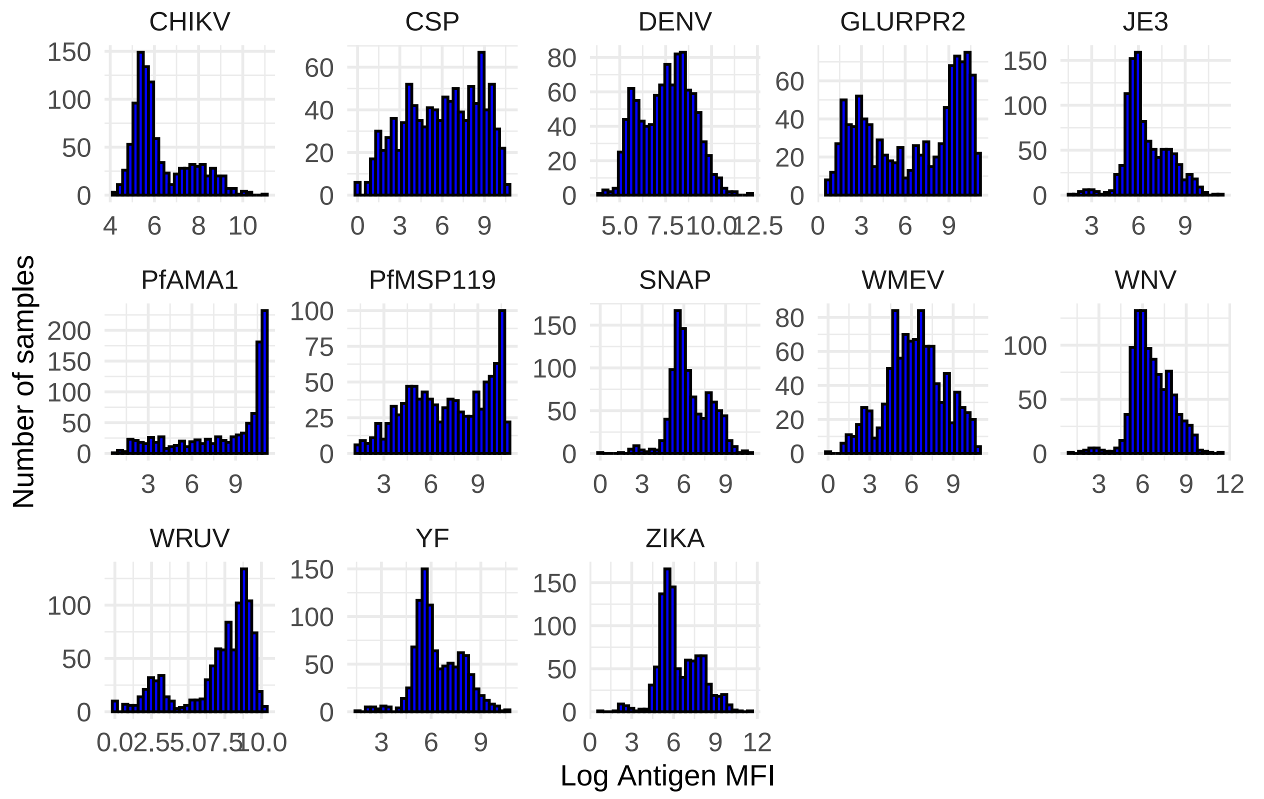
If you want to apply a mixture model, answer the following questions

a. Plot your sample values and consider if they have a good distribution to fit a mixture model.

```
  faceted_natural_scale <- ggplot(sample_long, aes(x = mfi)) +
  geom_histogram(bins = 30, color = "black", fill = "blue") +
  facet_wrap(~ antigen, scales = "free", ncol = 5) +
  labs(
    title = "",
    x = "Antigen MFI",
    y = "Number of samples"
  ) +
  theme_minimal() +
  theme(
    strip.text = element_text(size = 10),  # Smaller font for facet labels
    axis.text = element_text(size = 10),  # Smaller font for axis text
    plot.title = element_text(size = 10)  # Smaller font for the plot title
  )
faceted_natural_scale
```



```r
#log scale
faceted_log_scale <- ggplot(sample_long, aes(x = log(mfi))) +
  geom_histogram(bins = 30, color = "black", fill = "blue") +
  facet_wrap(~ antigen, scales = "free", ncol = 5) +  # <- Set 5 columns per row
  labs(
    title = "",
    x = "Log Antigen MFI",
    y = "Number of samples"
  ) +
  theme_minimal() +
  theme(
    strip.text = element_text(size = 10),  # Smaller font for facet labels
    axis.text = element_text(size = 10),  # Smaller font for axis text
    plot.title = element_text(size = 10)  # Smaller font for the plot title
  )
faceted_log_scale
```



b. Decide which antigen you want to test. Make a vector of your antigen values. If you want to do a log transformation, do so before putting data into the FMM. Use the following code to run the model with 2 components

```
mfi_values <- sample_data$JE3 #fill in your own data here
log_antigen_vector <- log(mfi_values)
k <- 2 #number of components
fmm_model <- Mclust(log_antigen_vector, G = k, modelNames = "V")
```

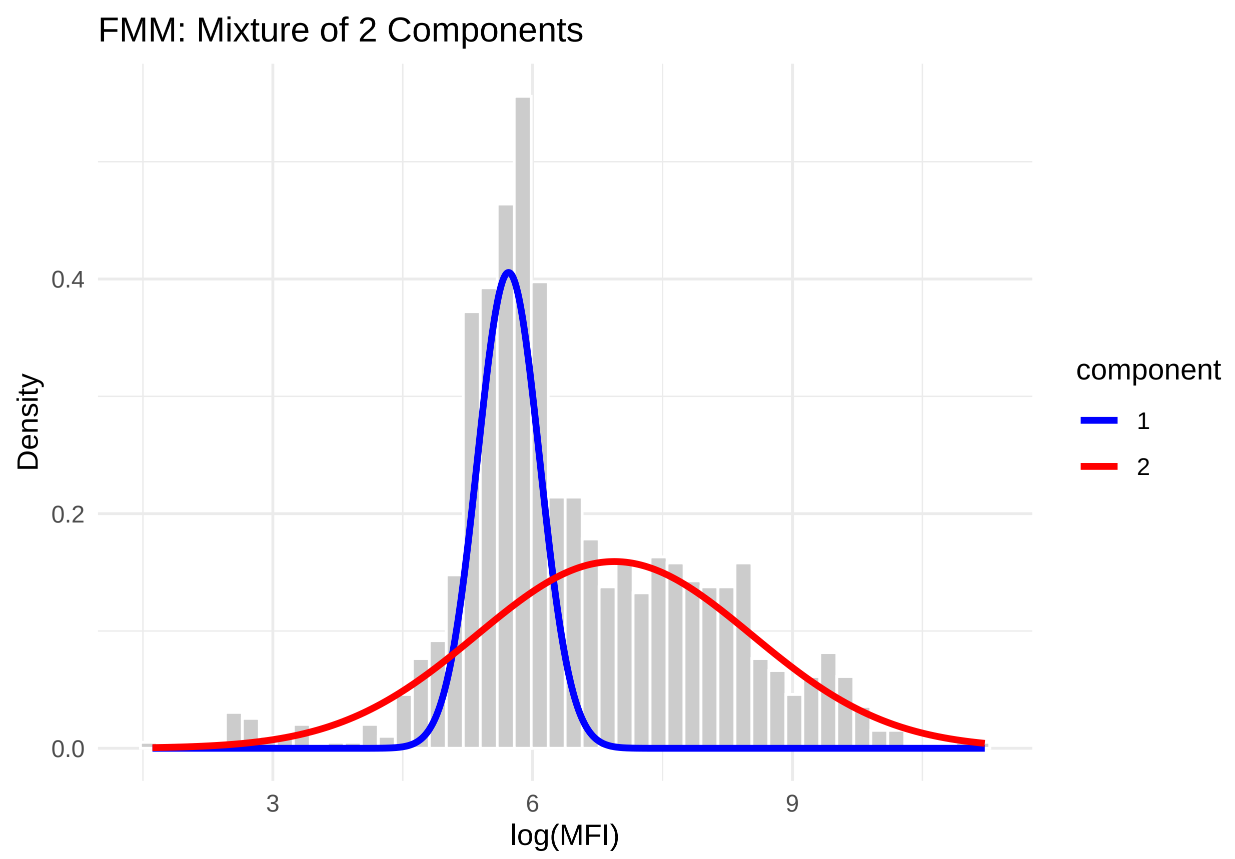
c. Make a plot of your data with the 2 predicted components mapped over them.

```
# Extract parameters
means <- fmm_model$parameters$mean
sds <- sqrt(fmm_model$parameters$variance$sigmasq)
props <- fmm_model$parameters$pro

# Create density curves for each component
x_vals <- seq(min(log_antigen_vector), max(log_antigen_vector), length.out = 1000)

dens_df <- data.frame(
x = rep(x_vals, 2),
density = c(
  dnorm(x_vals, mean = means[1], sd = sds[1]) * props[1],
  dnorm(x_vals, mean = means[2], sd = sds[2]) * props[2]
),
  component = factor(rep(1:2, each = length(x_vals)))
)

# Plot histogram + density curves
ggplot() +
  geom_histogram(aes(x = log_antigen_vector, y = after_stat(density)),
                 bins = 50, fill = "gray80", color = "white") +
  geom_line(data = dens_df, aes(x = x, y = density, color = component), size = 1.2) +
  labs(title = "FMM: Mixture of 2 Components",
       x = "log(MFI)", y = "Density") +
  scale_color_manual(values = c("blue", "red")) +
  theme_minimal()
```

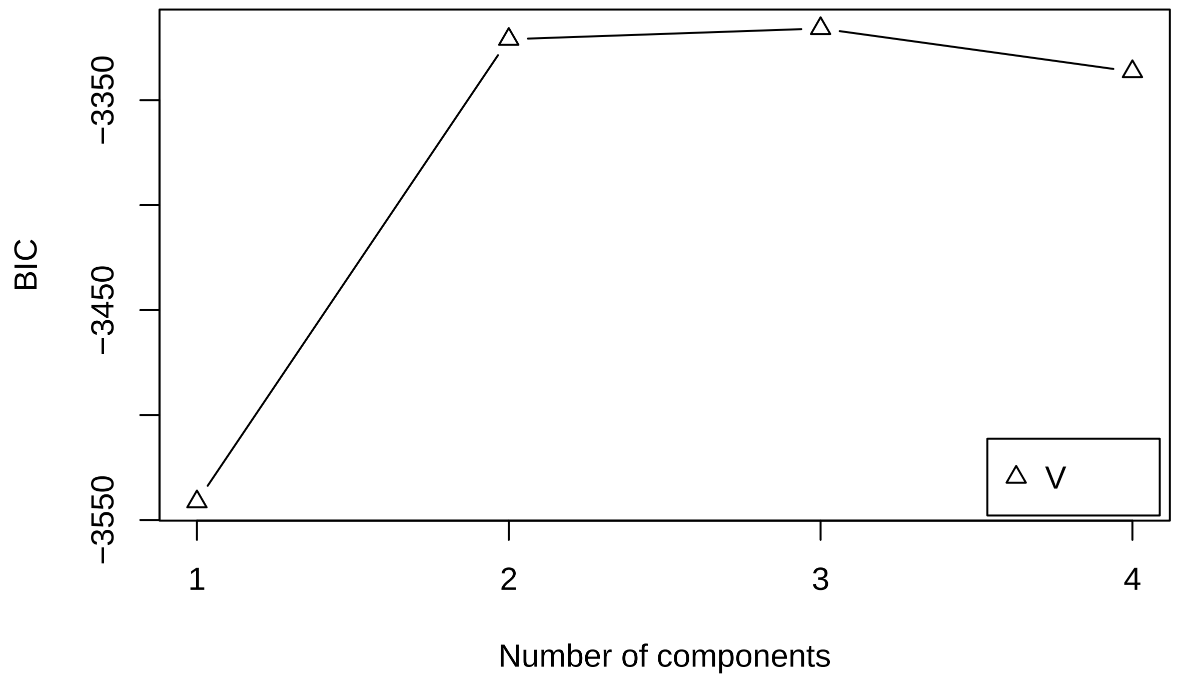


d. Test to see what number of components is the best fit for your data. Remember, FMM might not be an appropriate approach if 2 components is not the best fit for your data.

```
# Step 1: Estimate BIC values for G = 1 to 4 components
bic_values <- mclustBIC(log_antigen_vector, modelNames = "V", G = 1:4)

# Step 2: Extract BIC values into a data frame
bic_df <- as.data.frame(bic_values[,1])
bic_df$G <- as.numeric(rownames(bic_df)) # G is the column for number of components
names(bic_df) <- c("BIC", "G") # Label column names

# Step 3: Find best model based on max BIC value as derived from the mclust package
# Note that this package calculates a transformed BIC value; typically a model of best fit would have t
plot(bic_values)
```



```
best_index <- which.max(bic_df$BIC)

cat("Best number of components based on BIC:", best_index, "\n")
```

```
## Best number of components based on BIC: 3
```

e. If in part d, the model indicates that 2 components is not the best fit for your model, remake the plot showing your sample data fitted with the correct number of components. Copy the code from parts b and c below and edit to run with the best number of components.

f. Based on 2 components, calculate seroprevalence using the code below. Remember, if 2 components is not the best fit for your data, then this estimate may not be valid

```
#this uses the fmm_model results you generated in b.

# Step 1: Out of the 2 components, we identify which distribution has a higher mean, and assume that is
component_means <- fmm_model$parameters$mean
seropositive_component <- which.max(component_means)

# Step 4: Get the posterior probabilities for each sample
posteriors <- fmm_model$z  #For each sample, this is the probability of the component being in the firs
```

```r
seropositive_probs <- posteriors[, seropositive_component] #this pulls out just the probability of each

# Step 5: Estimate seroprevalence
estimated_seroprevalence <- mean(seropositive_probs) #the mean of all the probabilities will be equal t

# Step 6: Report as percentage
cat("Estimated seroprevalence:", round(estimated_seroprevalence * 100, 1), "%\n")
```

```
## Estimated seroprevalence: 63.4 %
```

g. Compare the seroprevalence you calculated in part f. to the seroprevalence calculated from the other cutoff methods earlier in the lab.