Stat_490_Paper

Andrew Maloney

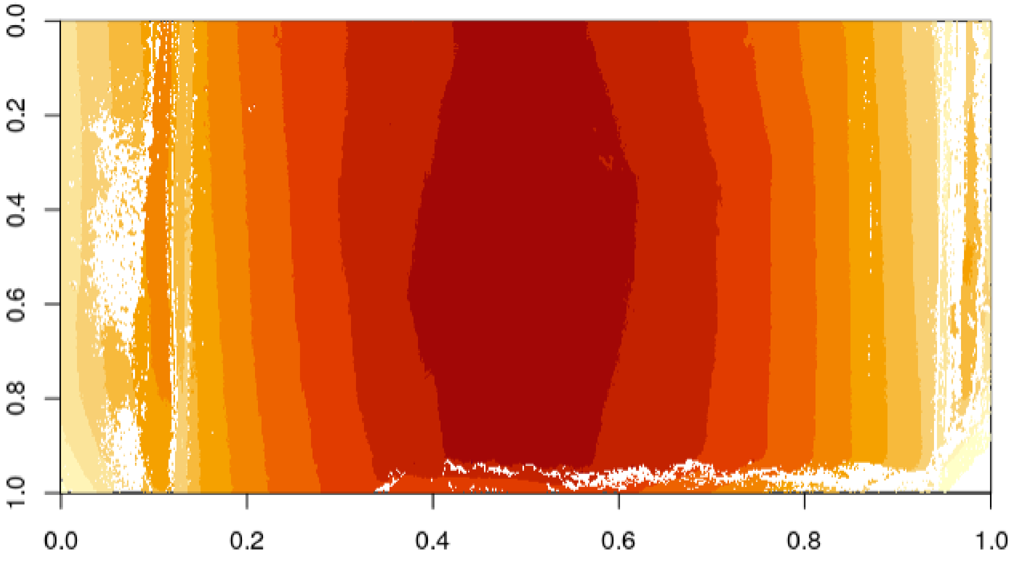
5/5/2020

Consecutive Matching Striae

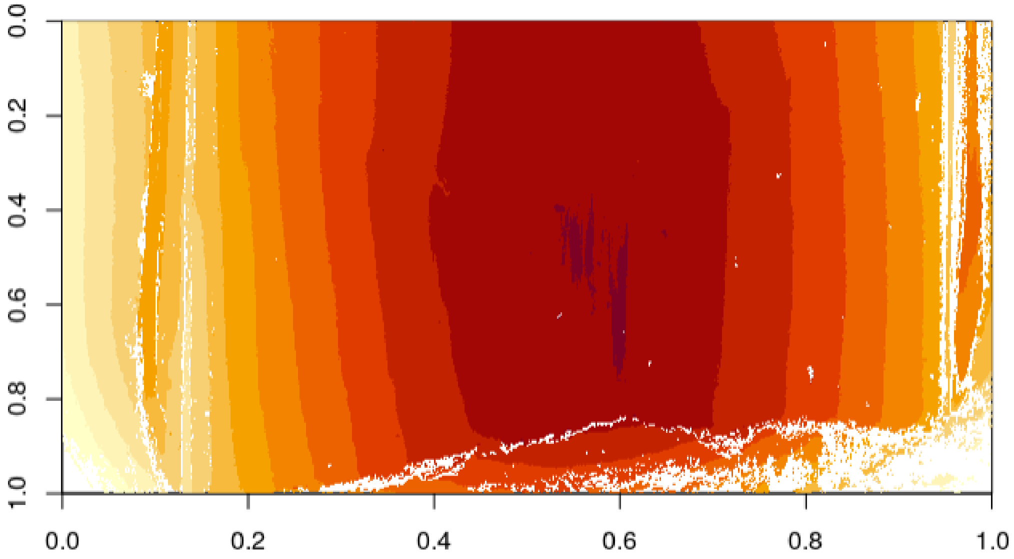
Produced in R Markdown

- When a bullet is fired from a gun, there are distinct markings left on the bullet. These markings are generally caused by the spiral engravings inside of the barrel, the igniting of the powder propellant, or foreign matter inside the barrel. These distinctive markings allow for firearm examination to be conducted. Currently, the comparison of two bullets is a subjective process involving simultaneous visual examination of two bullets by a trained practitioner. Using 3D microscopy, the bullets can be numerically characterized and the matching process can be automated. When the bullet has been numerically characterized we are left with a set of quantitative 'features', each with there own column in a data set. One of these features is known as CMS(Consecutive Matching Striae) and we will examine the underlying structure of how the CMS is calculated, how changing parameters used to calculate CMS affect the final calculation result for CMS. We expand the scope of the matching algorithm by fitting a generalized random forest model using a wide range of different CMS calculation. With this expanded model, we expect that the algorithm's results using different CMS calculations will lead to new insights on how 'important' CMS is as a feature and how different CMS impact the Accuracy of two bullets being a match or not a match. We perform all of our calculations using the R and mulitple R Programming Language. Before we begin to examine the feature known as consecutive matching striae, we will need to perform a number of steps with the bullet scans(edit)
- For this paper we use the Phoenix data set which consist of 198 scanned bullets. The scanned bullets are in a .x3p file format of the surface scan between adjacent groove impressions on the bullet and can be read into R using the R package x3ptools. For this study we will exam two datasets. One Phoenix dataset will have its x3p files containing the scanned bullets down sampled using the x3p_sample() function(See Figure 1). When down sampling is performed the number of pixels in each x3p file are reduced down by a half vertically and horizontally. The other Phoenix dataset will have no down sampling performed to the x3p files containg the scanned bullets(See Figure 2). We examine the scaled measurements of each bullet after being loaded into R. We use the function x3p_get_scale() from the x3ptools package to see what the scale is for each of the bullet scans in both datasets. The original measurement scale of the bullet depends on the microscope and scanner used to scan the bullet before being converted into an .x3p file format. The Phoenix dataset with no sampling has a resolution of 0.654 and the sampled Phoenix dataset has a resolution of 1.29. This makes sense because reducing the pixels by a half will result in a larger micron measurement. *Note: as a final check we make sure that the base of the bullet are at the bottom of the scan. The bottom of the bullet scan is known as the 'heel' of the bullet.*

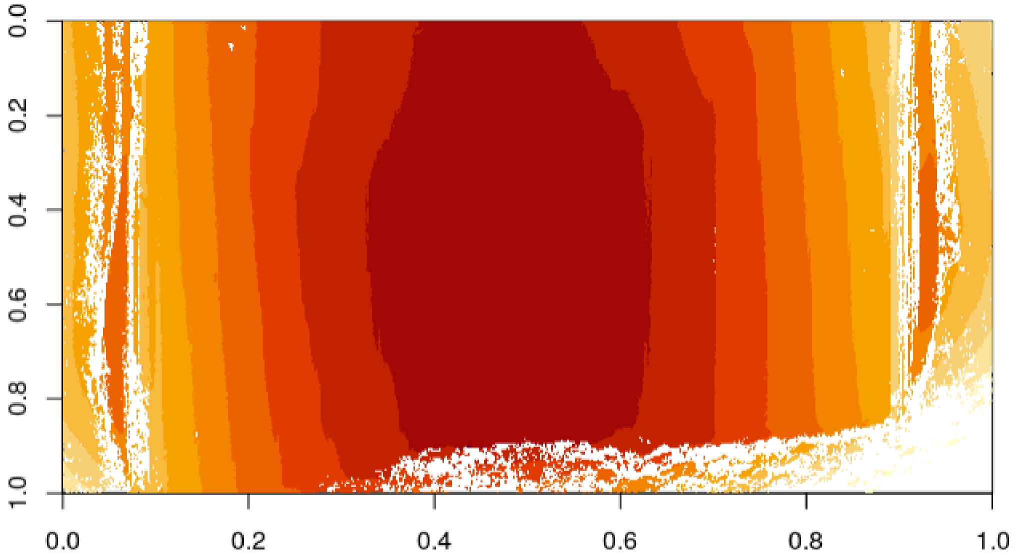
(Figure 1): Image of scanned Bullet with Reduced Pixels



Scan 1 Sampled

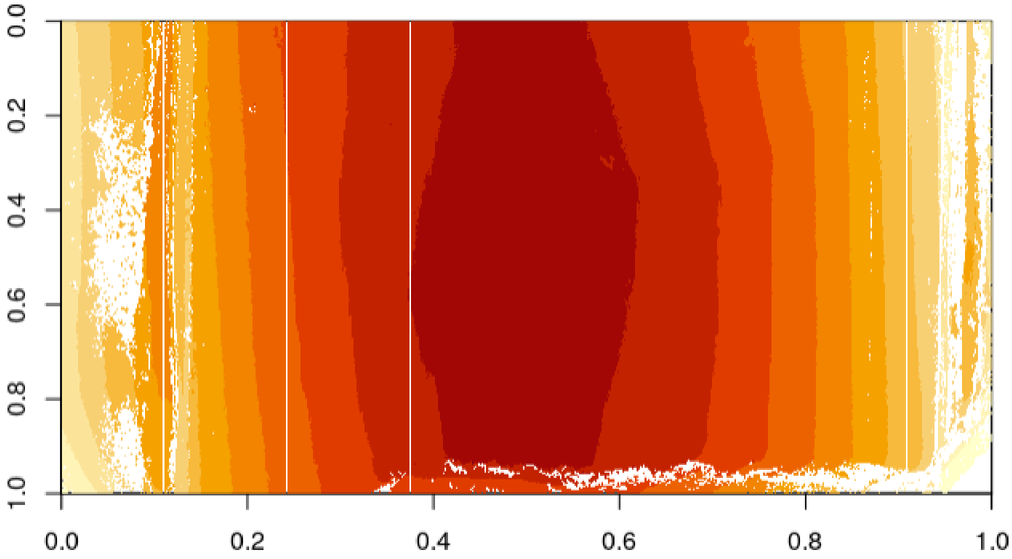


Scan 11 Sampled

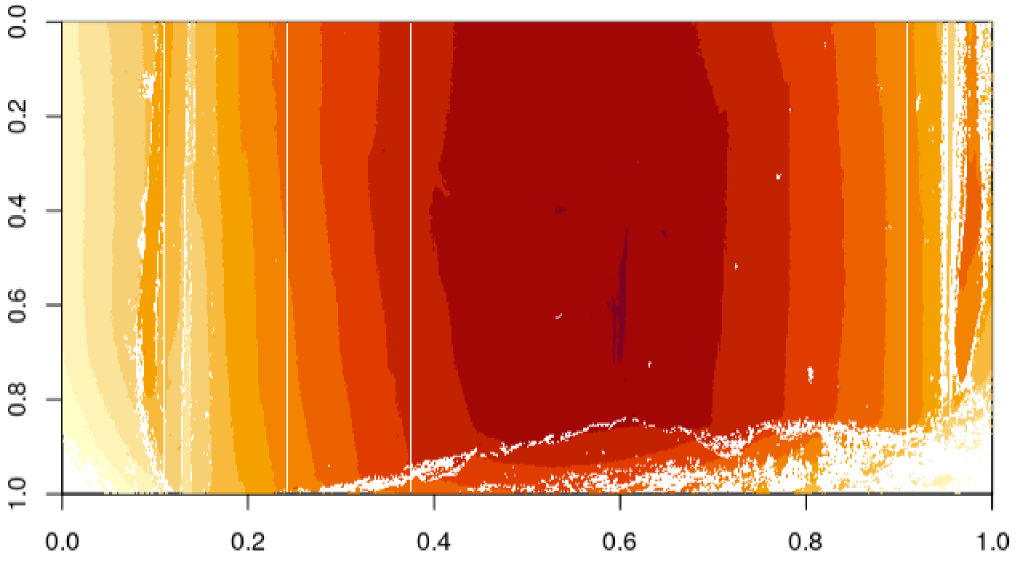


Scan 111 Sampled

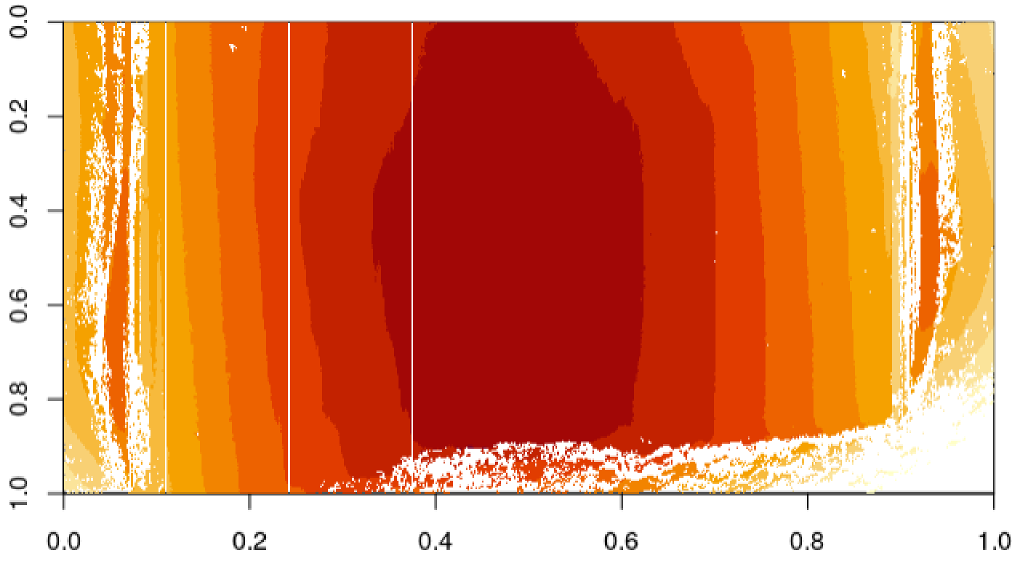
(Figure 2): Regular image of scanned Bullet



Scan 1



Scan 11



Scan 111

- We are now able to analyze the surface scan, by taking a cross section of the surface scan measurements. This can be performed using the x3p_crosscut_optimize() function and then next the x3p_crosscut() function. Both of which are in the bulletxtrctr R package. *Note: We will refer to the surface scan as a land impression from now on.* When plotting the crosscuts we are left with a side profile of the land impressions(See Figure 3). Notice the shape of the side profile, it resembles a perfect semi circle. If we think about the shape of a bullet and look at surface of the bullet outside its casing, we can take a pencil and sketch a 180 degree line along the bullet. Which will give us the same semi circular shape. Also notice the two hills on the left and right side of the cross sections. These hills are known as shoulders and at the valleys of these shoulders is where the land impression starts and ends. The shoulders contain no useful information to date and are removed.

(Figure 3): Side Profile of a bullets land impression

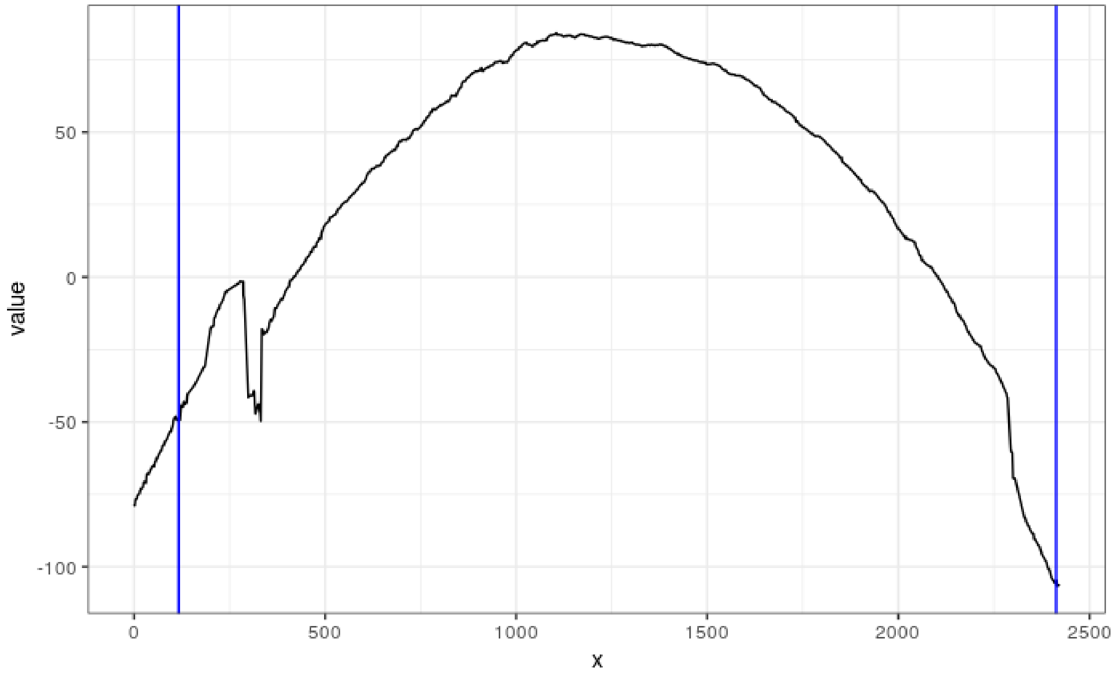
- We begin the process of removing the shoulders. First we identify the shoulder locations of each land impression(See Figure 4). Second we smooth the land impression(See figure 5). We then remove the shoulders of the land impression by specifying the left and right shoulder cutoffs. Currently this process is trying to be automated using the cc_locate_grooves() function in the bulletxtrctr package but for this study we manually removed the shoulders of the smoothed land impression using a shiny app(See Figure 6).

[[[- Briefly compare the the down-sampled data and the non down-sampled data]]]

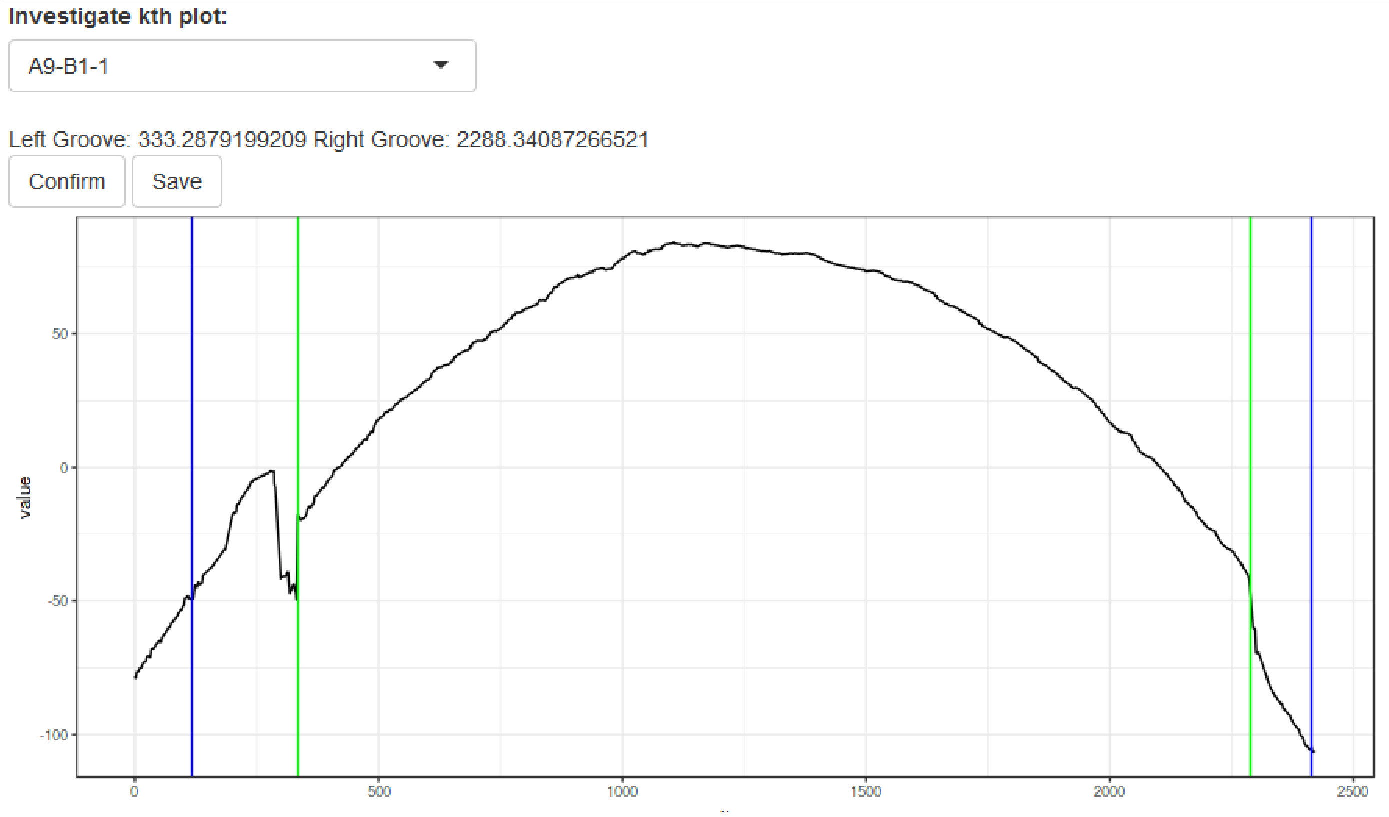
(Figure 4): Shoulder locations x2

(Figure 5): Smoothed Impression x2

(Figure 6): Shiny displaying the new groove cutoffs.



Before Cutoff



Investigate kth plot:

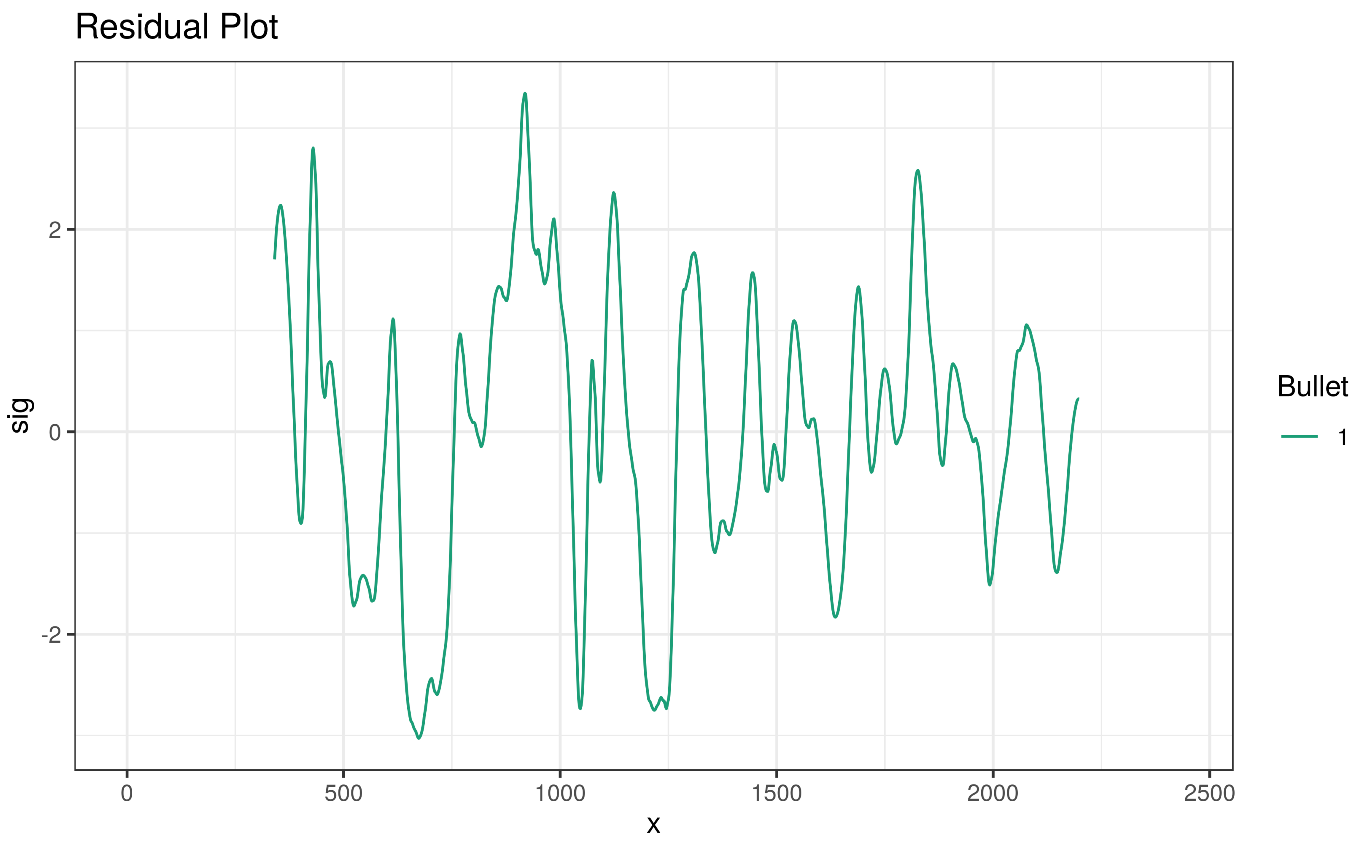
A9-B1-1

Left Groove: 333.2879199209 Right Groove: 2288.34087266521

Cutoff

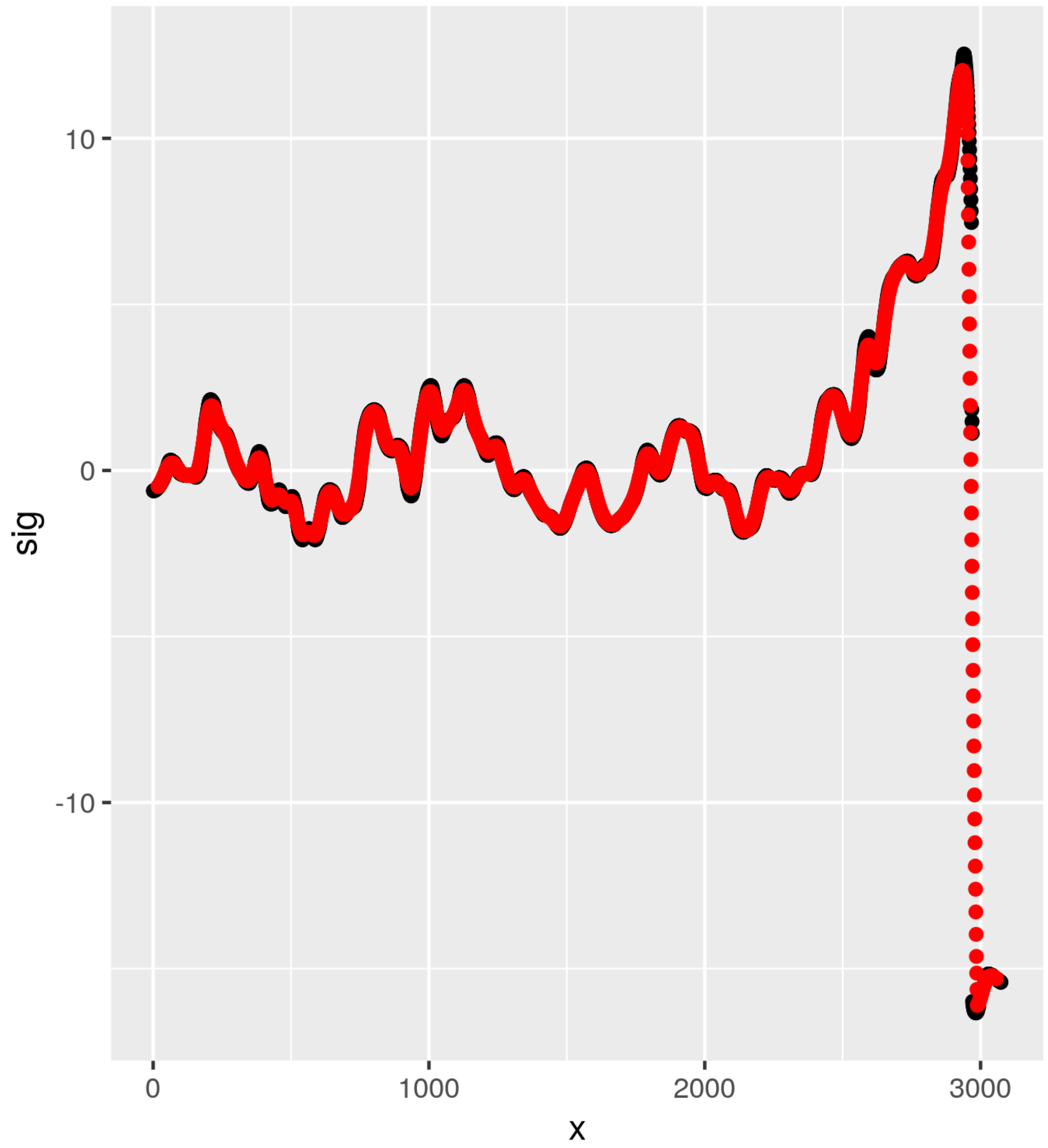
- After we have identified the groove cutoffs we look for the lowest stable region for each land impression and extract the signatures(See Figure 7). The signatures can be obtained using the cc_get_signature() function in the bulletxtrctr package.

(Figure 7): Signature Plot



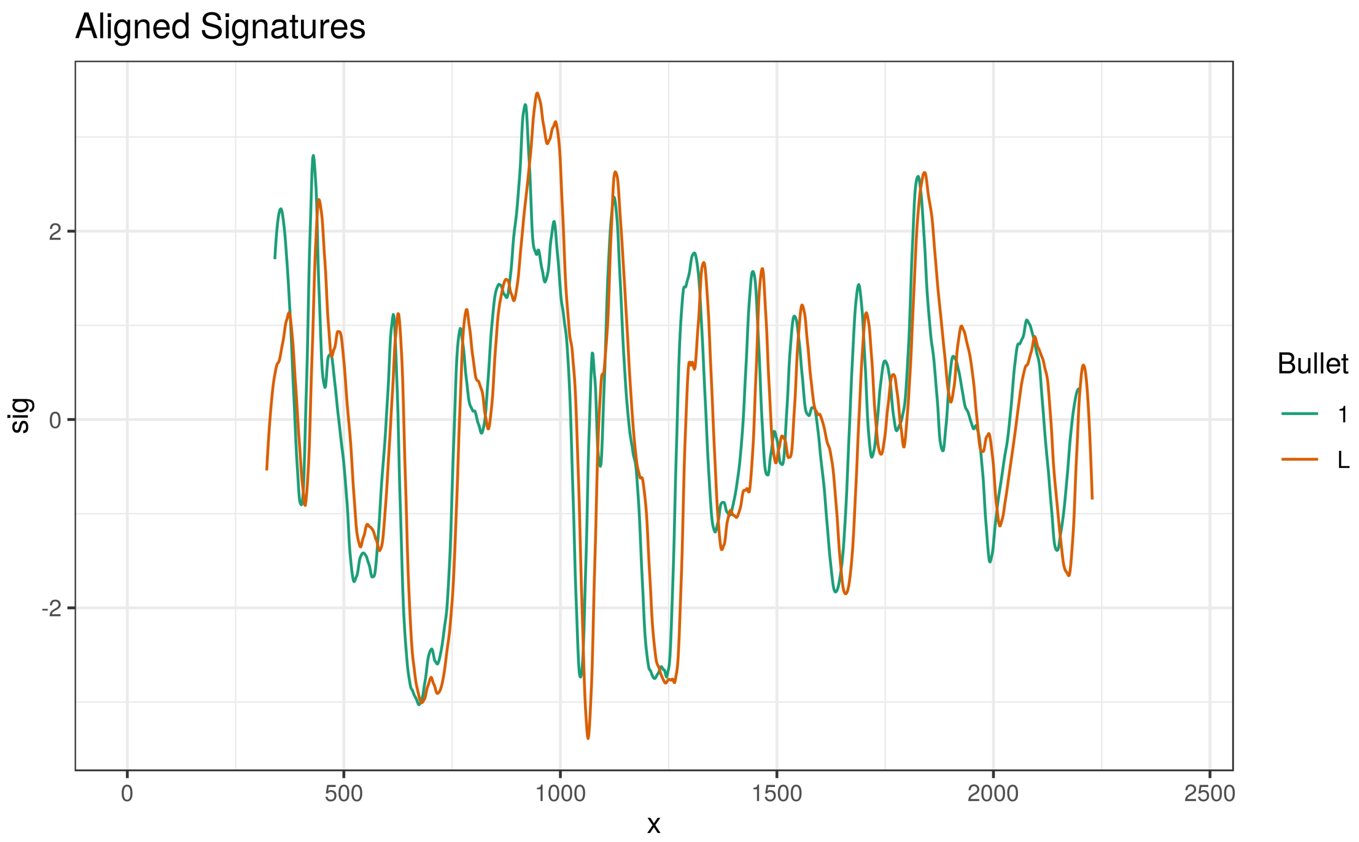
- In the beginning of this paper we talked about bullets being numerically characterized, which we said were features. One of these features is CMS, also known as consecutive matching striae. We are now able extract the CMS and all the other features associated with bullets land impression. We first smooth the signatures(See Figure 8), identifiy peaks and valleys of each signature(See Figure 9), and align every signature with eachother(See Figure 10). When the two signatures are aligned we are able to identify the matching peaks and valleys which coincide to the striation marks left on a bullet's surface after being fired(See figure 11). This is done by using the sig_aligned() function in the bulletxtrctr package. Notice the word "striation," that's right these are our consecutive matching striae. We can calculate the consecutive matching striae using the sig_cms_max() function in the bulletxtrctr package.

(Figure 8): Show Smoothed Vs Non Smoothed(span = default)



(Figure 9): Show identified peaks and valleys

(Figure 10): Show Aligned Signatures



(Figure 11): cite and show actually markings associated with cms)

- We explore the maximum number of CMS using different values for the span parameter in sig_cms_max(). We run 8 different spans and compare the Maximum number of CMS

Summary Statistics

[1] implies same comparisons are kept for statistical summary

[2] implies same comparisons are removed for statistical summary

Span of 15:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 3.000 3.141 4.000 73.000 [1]

0.000 2.000 3.000 2.872 4.000 34.000 [2]

Span of 25:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 3.000 3.062 3.000 62.000 [1]

0.000 2.000 2.000 2.827 3.000 37.000 [2]

Span of 35:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 2.000 2.974 3.000 56.000 [1]

0.000 2.000 2.000 2.776 3.000 32.000 [2]

Span of 45:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 2.000 2.915 3.000 47.000 [1]

0.000 2.000 2.000 2.745 3.000 37.000 [2]

Span of 55:

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 2.000 2.832 3.000 41.000 [1]
0.000 1.000 2.000 2.685 3.000 34.000 [2]
```

Span of 65:

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 2.000 2.698 3.000 37.000 [1]
0.000 1.000 2.000 2.569 3.000 34.000 [2]
```

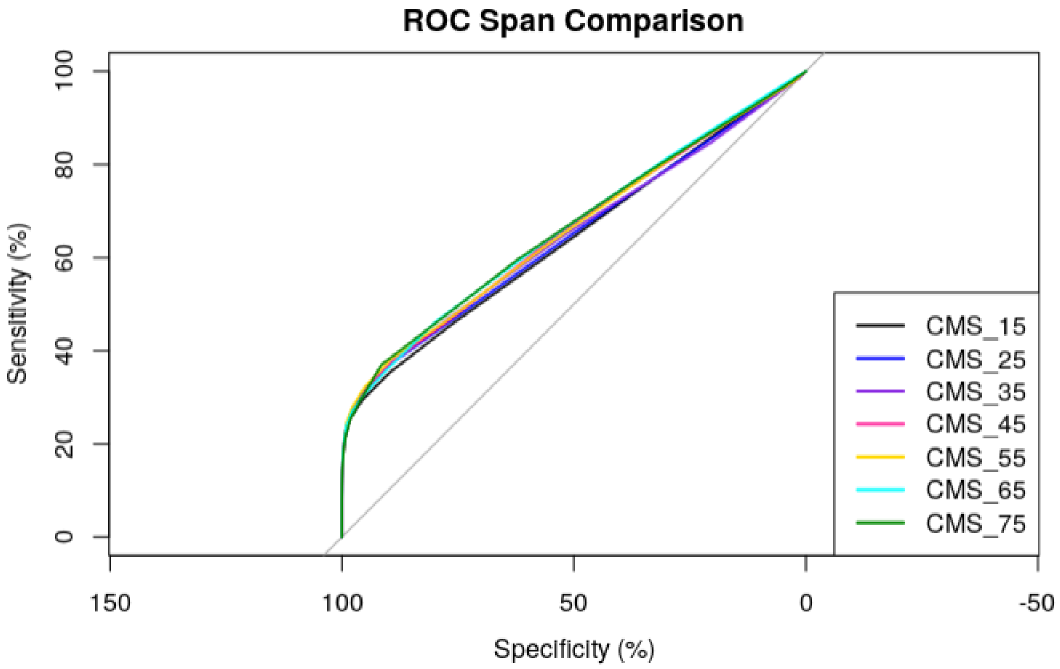
Span of 75:

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 2.000 2.557 3.000 34.000 [1]
0.000 1.000 2.000 2.442 3.000 25.000 [2]
```

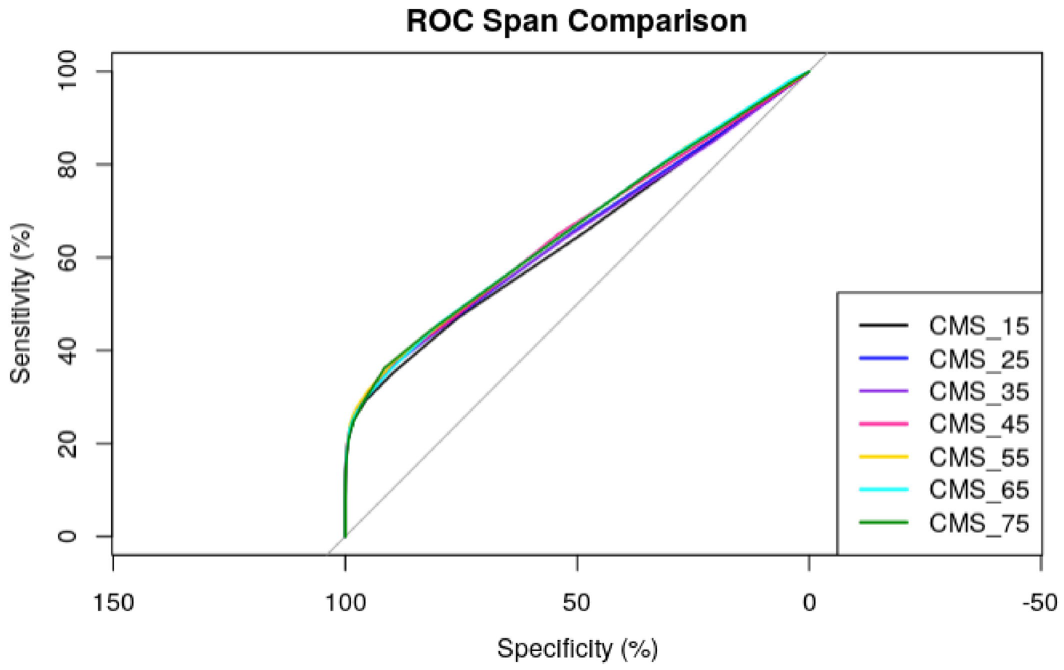
Span of 85:

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 2.000 2.451 3.000 30.000 [1]
0.000 1.000 2.000 2.349 3.000 23.000 [2]
```

AUC Calculations No Samp



Distinct



Non Distinct

Span of 15: Area under the curve: 0.6414

Span of 25: Area under the curve: 0.6498

Span of 35: Area under the curve: 0.6487

Span of 45: Area under the curve: 0.6577

Span of 55: Area under the curve: 0.6611

Span of 65: Area under the curve: 0.6614

Span of 75: Area under the curve: 0.6606

Span of 85: Area under the curve: 0.6602

Code Appendix

```
image(Phoenix$x3p[[7]])
ggsave("No_Sampling.png")

Phoenix <- Phoenix %>% mutate(x3p = map(x3p, sample_x3p, m = 2)) # Down Sampling

image(Phoenix$x3p[[7]])
ggsave("Sampling.png")
```

```
Phoenix <- Phoenix %>%
  mutate(CrossSection = map_dbl(x3p, x3p_crosscut_optimize))
```

```
Phoenix <- Phoenix %>%
  mutate(CrossCut = map2(.x = x3p, .y = CrossSection, .f = x3p_crosscut))

crosscuts_phoenix <- select(Phoenix, -path, -x3p) %>%
      tidyr::unnest(CrossCut)
```

```
library(shiny)
if (file.exists(saved_grooves_location_Phoenix)) {
  Phoenix$Grooves <- readRDS(saved_grooves_location_Phoenix)
}
if (interactive()) { # only run when you're manually running chunks... don't run when the whole document is compiled.
  shinyApp(
    ui = fluidPage(
      selectInput("k", "Investigate kth plot:",
                  selected = 1,
                  choices = (1:length(Phoenix$Grooves)) %>% set_names(Phoenix$id)
      )
      ),
      textOutput("groovelocations"),
      actionButton("confirm", "Confirm"),
      actionButton("save", "Save"),
      plotOutput("groovePlot", click = "plot_click"),
      verbatimTextOutput("info")
    ),

    server = function(input, output, session) {
      output$groovePlot <- renderPlot({
        k <- as.numeric(input$k)
        p <- Phoenix$Grooves[[k]]$plot

        p
      })
      output$groovelocations <- renderText({
        paste(
          "Left Groove: ", Phoenix$Grooves[[as.numeric(input$k)]]$groove[1],
          " Right Groove: ", Phoenix$Grooves[[as.numeric(input$k)]]$groove[2]
        )
      })
      observeEvent(input$confirm, {
        cat(paste(Phoenix$id[as.numeric(input$k)], "\n"))
        updateSelectInput(session, "k", "Investigate kth plot:",
                          selected = as.numeric(input$k) + 1,
                          choices = (1:length(Phoenix$Grooves)) %>% set_names(Phoenix$id)
        )
      })
      observeEvent(input$save, {
        saveRDS(Phoenix$Grooves, file = saved_grooves_location_Phoenix)
        message("groove data saved\n")
      })

      observeEvent(input$plot_click, {
        k <- as.numeric(input$k)
        xloc <- input$plot_click$x

        gr <- Phoenix$Grooves[[k]]$groove
        if (abs(gr[1] - xloc) < abs(gr[2] - xloc)) {
          Phoenix$Grooves[[k]]$groove[1] <<- xloc
        } else {
          Phoenix$Grooves[[k]]$groove[2] <<- xloc
        }
        output$groovePlot <- renderPlot({
          k <- as.numeric(input$k)
          p <- Phoenix$Grooves[[k]]$plot +
            geom_vline(xintercept = Phoenix$Grooves[[k]]$groove[1], colour = "green") +
            geom_vline(xintercept = Phoenix$Grooves[[k]]$groove[2], colour = "green")

          p
        })
      })
      output$info <- renderText({
        paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
      })
    },
    options = list(height = 500)
  )
  saveRDS(Phoenix$Grooves, file = saved_grooves_location_Phoenix)
} else {
  if (!file.exists(saved_grooves_location_Phoenix)) {
    message("run script in interactive mode to fix grooves")
  } else {
    Phoenix$Grooves <- readRDS(saved_grooves_location_Phoenix)
  }
}
```

```
Phoenix <- Phoenix %>%
 mutate(Signatures = future_map2(.x = CrossCut, .y = Grooves, .f = cc_get_signature, span = 0.75, span2 = .0
3))

 Signatures_Phoenix <- Phoenix %>%
  select(id, Barrel, Bullet, Land, Signatures) %>%
   tidyr::unnest()
```

```
Phoenix_Comparisons <- crossing(Bullet1 = Phoenix$id, Bullet2 = Phoenix$id) %>%
  left_join(nest(Phoenix, -id) %>% magrittr::set_names(c("Bullet1", "Bullet1_data"))) %>%
  left_join(nest(Phoenix, -id) %>% magrittr::set_names(c("Bullet2", "Bullet2_data")))
```

```
get_sig <- function(data) {
  map(data$Signatures, "sig")
}
Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(sig1 = map(Bullet1_data, get_sig), sig2 = map(Bullet2_data, get_sig))

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  select(-Bullet1_data, -Bullet2_data)

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(Aligned = map2(sig1, sig2, ~sig_align(unlist(.x), unlist(.y))))
```

```
Phoenix_Comparisons <- Phoenix_Comparisons %>%
 mutate(Striae = map(Aligned, sig_cms_max, span = 15))

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(features = map2(.x = Aligned, .y = Striae, .f = extract_features_all, resolution = resolution_p))

Phoenix_Comparisons <- Phoenix_Comparisons_15 %>%
  tidyr::unnest(features)
```