ECS 145 Term Project

Gavin Grey, Jeremy Kwan, Shayan Mandegarian, Michael Nguyen

1 Introduction and Goals

Many people like to use a histogram or kernel density plot to visualize their data and the shape of the distributions. Histograms are controlled by the argument 'breaks' which changes the number of bins that are displayed in the plot. Where kernel density plots have the argument 'bw' which control the wiggliness of the curve. These arguments are called tuning parameters and are the purpose of this project. This assignment explores applying different tuning parameters to a dataset to generate plots of histograms or density plots. Our goal is to produce an R package that helps users explore plots based on their given parameters. We wanted to develop a solution that was not only correct, but also easy for anyone to use. For our term project, we had to write a function of call form exploreShape(x, estMethod, tuning, twoAtATime) where x is a numeric vector, estMethod is whether we want to graph a histogram or a density graph, tuning is the number of breaks for a histogram, or the bw value of the density graph, and twoAtATime is whether we want to superimpose the previous graph onto a new graph.

2 Graphing

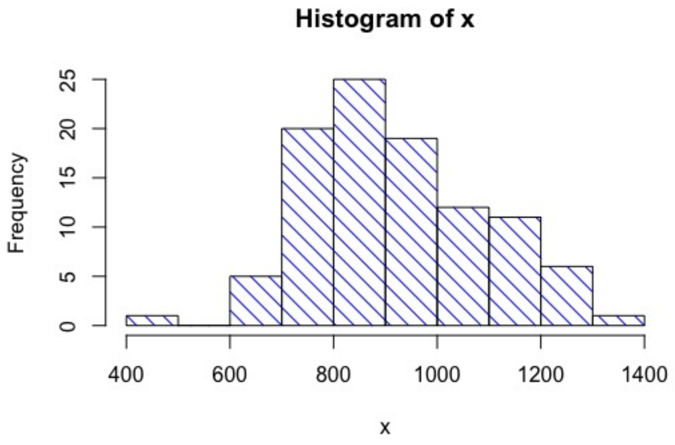

Figure 1: Sample output explore(Nile, 'hist', 7, T)

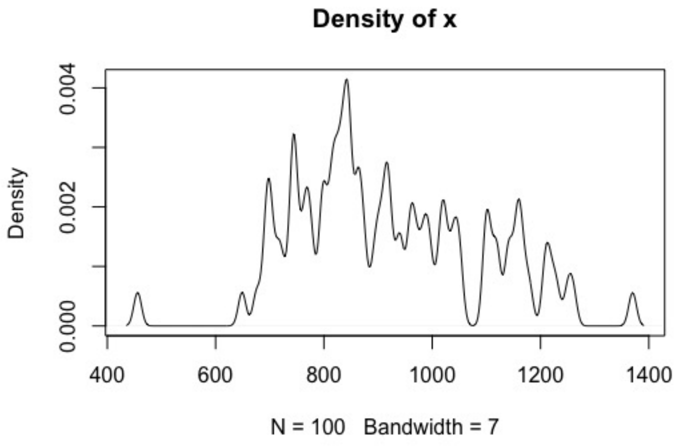

Figure 2: Sample output explore(Nile, 'density', 7, T)

2.1 Changing Tuning Parameter

Whenever we change the tuning parameter of the graph, if twoAtATime is initialized as true, then the previous graph is kept and superimposed onto the other graph. The previous graph is colored red to help differentiate between the graphs. To do this, we use a global variable('gmemory') that keeps track of the previous tuning parameter, so that when we graph with some new parameter, the old graph can be superimposed on top of it, and the new parameter gets stored into the global variable.

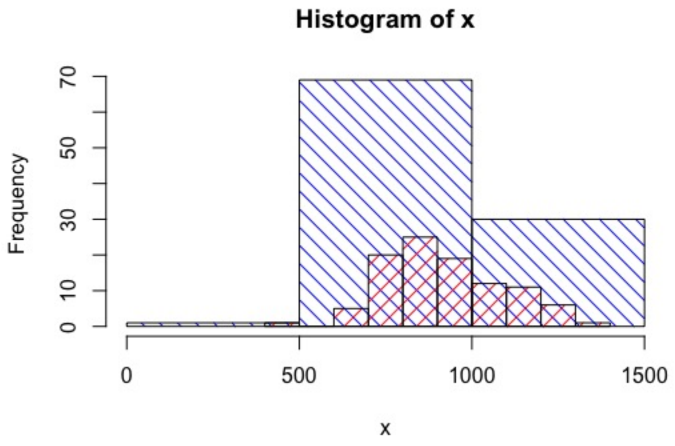

Figure 3: Sample output explore(Nile, 'hist', 7, T), then changed tuning parameter to 3.

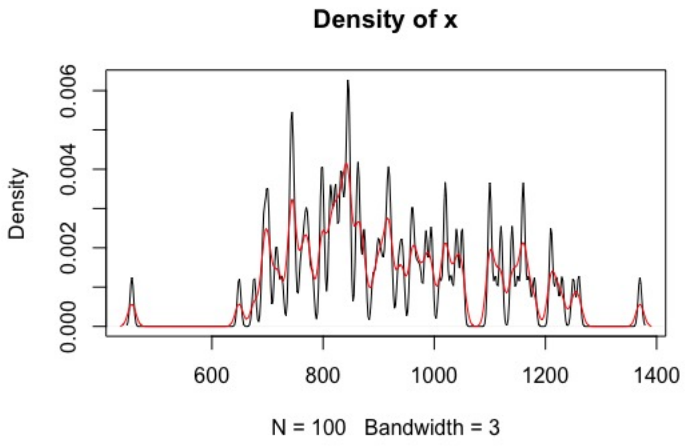

Figure 4: Sample output explore(Nile, 'density', 7, T) and then changed tuning parameter to 3.

2.2 Animation

To implement animations, we simply looped through multiple different tuning parameters, starting from 1 and stopping at 100. At each tuning parameter, the data is plotted and then the tuning parameter is incremented. This animation will not superimpose, even if the user specified to superimpose the other plots. We did not chose to use an external package to implement the animation part of this term project, but came up with a creative solution instead which uses a loop.

2.3 Zooming In and Out

Additionally, we included zooming functionality. This was done by first ordering the data set, then removing the first 10 and the last 10 points of data, in the case of zooming in. The amount of 'zoom in' could be adjusted through the variable 'zoomparam', which was set to 10 as the default. Then in the case of zooming out, we restore the original data set. To do this we keep the original data set stored in a global variable ('ogdata') that is set any time we pass a new data set to our function.

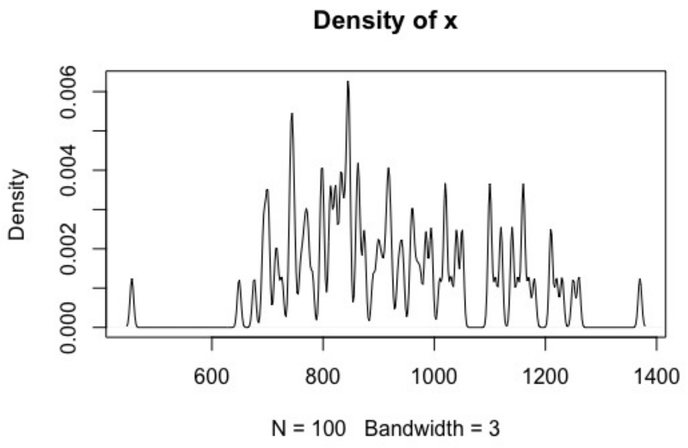

Figure 5: Sample output explore(Nile, 'density', 3, F)

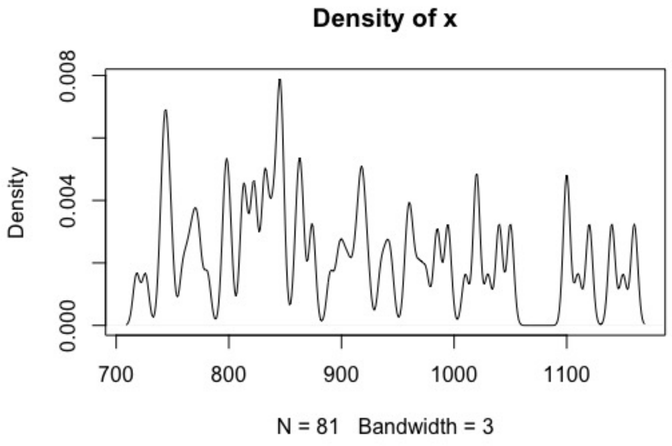

Figure 6: Sample output explore(Nile, 'density', 3, F) zoomed in.

3 Individual Contributions

3.1 Gavin

Gavin worked on debugging the code when errors arose, as well as brainstorming the implementation of various aspects of the code and writing the Graphing section of the report.

3.2 Jeremy

Initially the code was run automatically with the file, and not with a function call to exploreShape, but Jeremy put the code within the exploreShape function such that the loop and prompts went within that itself. Thus, the graphing occurred only when the user prompts it with the function exploreShape with the correct arguments. Jeremy helped with testing of the project code, and pointing out issues in output. Furthermore, Jeremy helped with outlining and organization of the report, writing the introduction as well as adding in the images of output graphs.

3.3 Shayan

Shayan mainly focused on improving the presentability of the project. One of the main areas of focus was making the twoAtATime functionality work properly and easier to understand for users. At one point, we had a problem where upon first calling exploreShape, it would still try to plot two at once using bogus data, so Shayan added code to ensure it would only plot twice after the user gives a new tuning parameter. Also, at first was hard to view two histogram plots at once, so Shayan changed it such that the histograms were colored with diagonal striped lines of different colors and angles. Shayan also made some quality of life improvements like handling bad user input and allowing users to save as many tuning parameters as they want.

3.4 Michael

Michael started off by creating comments and empty functions to get a general idea of which direction to take the Term Project. Michael created all necessary global variables,functions and classes that would be used later in development. Michael coded the functionalities for zoom in/out, plot animations, s3 class densEst and functions. The team would go on to work on top of Michael's code to improve and adjust the project. Michael also helped in adding information to the report.

4 Discussion

4.1 Assumptions and limitations

Our project assumes that the data to be used is the Nile data set provided by the R package. We can also make it so the user would have to provide their own data set. The animation feature of the project only animates up to a bw / break of 100. We could possibly make it so users would be able to set their limit.

4.2 Scope and Style

Throughout the coding process, we tried to provide as many informative comments about each method or variable that we used without being too verbose or ambiguous. We wrote many of the features into separate functions to keep the code clean and legible.

4.3 Edge Cases

4.4 Optimizations and Efficiency

4.5 Personal Reflection

This term project has provided us with an interesting opportunity to create an R package that could be used by someone who was interested in plotting their datasets with different tuning parameters. This project has taught us a lot in the process of team working to building a clean and easy to use end product.

If there was something we would do differently for next time..

Appendices

```
# project start
#x: A numeric vector to be graphed. ex. (3,5,2)
#estMethod: Either 'hist' or 'density'.
#tuning: The intial value of either breaks or bw.
#twoAtATime: If TRUE, always display the current graph superimposed on the previ
choicefour <- T
gmemory <- -1

#for zoom out
ogdata <- Nile

dataset <- Nile

#functions below are for the S3 object , will have functions plot,summary and pr
exploreS3 <- function(x,estMethod,tuningparams,twoAtATime){
  s <- list(data=x,method=estMethod,tuning=tuningparams,two=twoAtATime)
  class(s) <- "densEst"
  return (s)
}
print.densEst <- function(x){
  cat("Method:", x$method,"\n")
  cat("Tuning parameters:", x$tuning,"\n")
  cat("Superimpose plots:", x$two,"\n")
}
summary.densEst <- function(x){
  summary(x$data)
}
plot.densEst <- function(x){
  for (tune in x$tuning){
    if(x$method == 'hist'){
      if(x$two){
        p1 <- hist(x$data,breaks=tune,plot=FALSE)
        plot(p1, col="blue", angle = 135,density=10, main="Histogram of x")
        if (gmemory >= 0) {
          p2 <- hist(x$data,breaks=gmemory,plot=FALSE)
          plot(p2, col="red", add=T, angle = 45,density=10)
        }
        gmemory <<- tune
      }else{
        plot(hist(x,breaks=tuning), main="Histogram of x")
        gmemory <<- tune
      }
      readline(prompt = "Hit enter to view next Plot.")
    }else if(x$method == 'density'){
      if(x$two){
        plot(density(x$data,bw=tune), main="Density of x")
        if (gmemory >= 0) {
          lines(density(x$data,bw=gmemory),col = "red")
        }
        gmemory <<- tune
      }else{
        plot(density(x$data,bw=tune), main="Density of x")
        gmemory <<- tune
      }
      readline(prompt = "Hit enter to view next Plot.")
    }else{
      cat('Error: Was not given a valid Method name.\n')
      break
    }
  }
}

uservector <- function(){
  x <- 0
  myvec = vector()
  while(1){
    n <- readline(prompt = "Input a tuning parameter(type 'quit' to end): ")
    if(grepl("^[0-9]+$",n) | n == "quit") {
      if(n != 'quit'){
        myvec = c(myvec,n)
        x<- x+1
      }else{
        return (as.integer(myvec))
      }
    }
  }
  return (as.integer(myvec))
}
animate <- function(x,estMethod){
  ki <- 1
  cat("Running animation...")
  while(1){
    if(estMethod == 'hist'){
      hist(x,breaks=ki)
    }else if(estMethod == 'density'){
      plot(density(x,bw=ki))
    }
    ki <- ki + 2
    Sys.sleep(0.1)
```

```
      if(ki >= 100){
        break
      }
    }
  }

  newShape <- function(x,estMethod,tuning,twoAtATime){
    if(estMethod == 'hist'){
      if(twoAtATime){
        if(tuning == ''){
          p1 <- hist(x,plot=FALSE)
          plot(p1, col="blue", angle = 135,density=10, main="Histogram of x")
          if (gmemory >= 0) {
            p2 <- hist(x,breaks=gmemory,plot=FALSE)
            plot(p2, col="red", add=T, angle = 45, density=10)
          }
        }else{
          p1 <- hist(x,breaks=tuning,plot=FALSE)
          plot(p1, col="blue", angle = 135,density=10, main="Histogram of x")
          if (gmemory >= 0) {
            p2 <- hist(x,breaks=gmemory,plot=FALSE)
            plot(p2, add=T, col="red", angle=45,density=10)
          }
          gmemory <<- tuning
        }
      }else{
        if(tuning == ''){
          p1 <- hist(x,breaks=tuning,plot=FALSE)
          plot(p1, col="blue", angle = 135,density=10, main="Histogram of x")
        }else{
          p1 <- hist(x,breaks=tuning,plot=FALSE)
          plot(p1, col="blue", angle = 135,density=10, main="Histogram of x")
          gmemory <<- tuning
        }
      }
    }else if(estMethod == 'density'){
      if(twoAtATime){
        if(tuning == ''){
          plot(density(x), main="Density of x")
          if (gmemory >= 0) {
            lines(density(x,bw=gmemory),col = "red")
          }
        }else{
          plot(density(x,bw=tuning), main="Density of x")
          if (gmemory >= 0) {
            lines(density(x,bw=gmemory),col = "red")
          }
          gmemory <<- tuning
        }
      }else{
        plot(density(x,bw=tuning), main="Density of x")
        gmemory <<- tuning
      }
    }else{
      cat('Error: Was not given a valid Method name.\n')
      return(0)
    }
  }

    exploreShape <- function(x,estMethod,tuning,twoAtATime){
    dataset <<- sort(x, decreasing=FALSE)
    ogdata <<- sort(x, decreasing=FALSE)
    while(choicefour){
      newShape(dataset,estMethod,tuning,twoAtATime)
      cat("Please select one of the four options.\n")
      cat("1. Give a new value of the tuning parameter.\n")
      cat("2. Zoom in/out.\n")
      cat("3. Run an animation of tuning parameters.\n")
      cat("4. Quit.\n")
      selectOption <- readline(prompt="Enter a number and press Enter: ")
      if(selectOption == 1){
        tuning <- as.integer(readline(prompt = "Enter a new value of the tuning pa
      }else if(selectOption == 2){
        zoom <- readline(prompt = "Zoom in or Zoom out? ('in' , 'out'):")
        if(zoom == 'in'){
          zoomparam <- 10
          if(length(dataset) < zoomparam){
            cat("You can't zoom in anymore.\n")
          }else{
            cat("Zooming in...\n")
            dataset <<- dataset[zoomparam:(length(dataset)-zoomparam)]
          }
        }else if(zoom == 'out'){
          cat("Zooming out...\n")
          dataset <<- ogdata
        }
      }else if(selectOption == 3){
        #animation
        animate(dataset,estMethod)
      }else if(selectOption == 4){
        cat("Quit\n")
        break
      }
    }
    #after user selects 'Quit'. Get user's tuning parameters
    #save user selected parameters in memory and then
    #returns a vector of integers
    cat('Save some of your tuning parameters!\n')
    userparam = uservector()

    #Create the s3 object that we'll use in the next part
    s3obj <- exploreS3(ogdata,estMethod,userparam,twoAtATime)
    return(s3obj)
  }
```