Clean up gating

Anja Bråthen Kristoffersen

08 March 2022

This document describes the different lines of the file “example clean up gating alle filer.R”. To perform your own gating save the file “example clean up gating alle filer.R” from github <https://github.com/folkehelseinstituttet/cytof> give it a new name and change the necessary paths and values throughout the document to use it on your own data.

You start with defining a variable data\_path that contains the path of the folder where all your uncleaned files are saved. I prefer using the function path from the library fs. When using a function from a given library the way to get the function is by library\_name::function\_name so in this case fs::path

data\_path <- fs::path("C:", "CyToF data", "immun\_aga", "panel1 RAA")

which gives the path

data\_path

## C:/CyToF data/immun\_aga/panel1 RAA

Now define the paths to the folder where you want to save

* the new files that are clean up gated and
* figures of the gating performance.

Do also define the path where the files script used are saved.

outDataPath <- fs::path("C:", "CyToF data", "immun\_aga", "panel 1 cleaned data")  
out\_result <- fs::path("C:", "CyToF data", "immun\_aga", "figures panel 1 cleaning")  
scriptPath <- fs::path("C:", "CyToF data", "fra github")

I have already made many functions that can be used for reading, transforming, gating and plotting in R. These functions can be found on github <https://github.com/folkehelseinstituttet/cytof>. Save the files gating\_functions.R, ploting\_functions, read\_data\_function.R, analysis\_functions.R and transformation\_functions.R into your script folder and source them into R so that you might use them later

source(fs::path(scriptPath, "gating\_functions.R"))  
source(fs::path(scriptPath, "ploting\_functions.R"))  
source(fs::path(scriptPath, "read\_data\_functions.R"))  
source(fs::path(scriptPath, "transformation\_functions.R"))  
source(fs::path(scriptPath, "analysis\_functions.R"))

file.info() and list.files() are functions in R that can be used to find the name of all files in one folder

fcs\_files <- fs::path(data\_path, rownames(file.info(list.files(data\_path))))

to see which files that is included in the variable fcs\_files:

fcs\_files

## C:/CyToF data/immun\_aga/panel1 RAA/Ko\_Ma\_A\_70\_578K1\_Panel1.fcs  
## C:/CyToF data/immun\_aga/panel1 RAA/Ko\_Ma\_A\_72\_582K1\_Panel1.fcs  
## C:/CyToF data/immun\_aga/panel1 RAA/M\_Fe\_B\_20\_T1\_FHI155\_Panel1.fcs  
## C:/CyToF data/immun\_aga/panel1 RAA/M\_Fe\_B\_43\_T1\_FHI004\_Panel1.fcs  
## C:/CyToF data/immun\_aga/panel1 RAA/Ref1\_Panel1\_230621.fcs  
## C:/CyToF data/immun\_aga/panel1 RAA/S\_ Fe\_B\_22\_T1\_FHI111T1\_Panel1.fcs

basename() is a function that give you the name of the file without the path. By using the function grepl() we choose only the files that contains .fcs.

files\_to\_open <- basename(fcs\_files)  
files\_to\_open <- files\_to\_open[grepl(".fcs", files\_to\_open)]

setwd() is a function who set the working directory. Here we want the working directory to be where the uncleaned cytof files are. dirname() gives the directory/folder of one file, and fcs\_files[1] gives the first file in the vector fcs\_files.

setwd(dirname(fcs\_files[1]))

The variable file\_names are made as a shorter version of the variable files\_to\_open (without “.fcs”) by using the function gsub(), the function gsub() first takes the pattern that you want to substitute and then the pattern you want to replace it with in the vector which is the last variable in the function call. This is done to have shorter unique names on each dataset/file for later use.

file\_names <- gsub(".fcs", "", files\_to\_open)

The number of .fcs files in the variable vector file\_names are found by the function length()

n\_files <- length(file\_names)

Here a vector filenumbers containing 1, 2, 3, …. to n\_files are made for later use

filenumbers <- 1:n\_files  
filenumbers

## [1] 1 2 3 4 5 6

Two empty matrixes to take care of how many percentages of each file that are lost by each gating and in total of all previous gating are defined as percent\_lost\_each\_gating and percent\_lost\_from\_full\_dataset, respectively. These data.frames are given the column name of the gating and the rownames of the files that are gated.

percent\_lost\_each\_gating <- as.data.frame(matrix(NA, ncol = 9, nrow = n\_files))  
percent\_lost\_from\_full\_dataset <- as.data.frame(matrix(NA, ncol = 9, nrow = n\_files))  
  
colnames(percent\_lost\_each\_gating) <- c("Ce140Di", "Residual", "Center", "Offset", "Width",  
 "Event\_length", "Pt194Di", "Ir191Di", "Ir193Di")  
  
colnames(percent\_lost\_from\_full\_dataset) <- colnames(percent\_lost\_each\_gating)  
  
rownames(percent\_lost\_each\_gating) <- file\_names  
rownames(percent\_lost\_from\_full\_dataset) <- rownames(percent\_lost\_each\_gating)

plotSignal() is a function that has to be made to the dataset that are used. It tells how to plot the files you use. Here I have 6 files and I want to plot them with 2 rows and 3 columns. The variable plot\_list is a list that I will produce later that contain a plot per file. To write comments into an R code you use # in the beginning of the line. I have here written a version of this function that plots 66 plots in 6 rows and 11 columns.

plotSignal <- function(plot\_list){  
 g <- gridExtra::grid.arrange(plot\_list[[1]], plot\_list[[2]], plot\_list[[3]], plot\_list[[4]], plot\_list[[5]],  
 plot\_list[[6]], ncol = 3, nrow = 2)  
 return(g)  
}  
  
# plotSignal <- function(plot\_list){  
# g <- gridExtra::grid.arrange(plot\_list[[1]], plot\_list[[2]],   
# plot\_list[[3]], plot\_list[[4]], plot\_list[[5]],  
# plot\_list[[6]], plot\_list[[7]], plot\_list[[8]],   
# plot\_list[[9]], plot\_list[[10]], plot\_list[[11]],   
# plot\_list[[12]], plot\_list[[13]], plot\_list[[14]],   
# plot\_list[[15]], plot\_list[[16]], plot\_list[[17]],   
# plot\_list[[18]], plot\_list[[19]], plot\_list[[20]],  
# plot\_list[[21]], plot\_list[[22]], plot\_list[[23]],   
# plot\_list[[24]], plot\_list[[25]], plot\_list[[26]],  
# plot\_list[[27]], plot\_list[[28]], plot\_list[[29]],  
# plot\_list[[30]], plot\_list[[31]], plot\_list[[32]],   
# plot\_list[[33]], plot\_list[[34]], plot\_list[[35]],  
# plot\_list[[36]], plot\_list[[37]], plot\_list[[38]],   
# plot\_list[[39]], plot\_list[[40]], plot\_list[[41]],   
# plot\_list[[42]], plot\_list[[43]], plot\_list[[44]],   
# plot\_list[[45]], plot\_list[[46]], plot\_list[[47]],   
# plot\_list[[48]], plot\_list[[49]], plot\_list[[50]],  
# plot\_list[[51]], plot\_list[[52]], plot\_list[[53]],   
# plot\_list[[54]], plot\_list[[55]], plot\_list[[56]],   
# plot\_list[[57]], plot\_list[[58]], plot\_list[[59]],   
# plot\_list[[60]], plot\_list[[61]], plot\_list[[62]],   
# plot\_list[[63]], plot\_list[[64]], plot\_list[[65]],  
# plot\_list[[66]], ncol = 11, nrow = 6)  
# return(g)  
# }

You could either look at all files at the same time or divide it into blocks of files. Here is an example with all files. If memory problems, it is possible to divide the files to clean into groups using a loop.

Read all files in data\_path into a list object fcs\_data by using the function read\_some\_data\_from\_folder(). This function returns two elements the dataset and the file\_name in the same order as the data is read. I choose to store this in two separate variables and remove the variable with both objects by using the function rm().

fcs\_data\_with\_info <- read\_some\_data\_from\_folder(data\_path, file\_number = filenumbers)  
fcs\_data <- fcs\_data\_with\_info$fcs\_data  
file\_names <- factor(fcs\_data\_with\_info$file\_names, levels = fcs\_data\_with\_info$file\_names)  
rm(fcs\_data\_with\_info)

By using the function get\_params\_fcs\_data() the parameters of the first element of the list fcs\_data are obtained. By writing the variable name params we get to see what this variable contains.

params <- get\_params\_fcs\_data(fcs\_data[[1]])  
params

## name desc range minRange maxRange  
## $P1 Time <NA> 2097152 0 2097151  
## $P2 Event\_length <NA> 4096 0 4095  
## $P3 Y89Di 89Y\_CD45 16384 0 16383  
## $P4 Pd102Di 102Pd 4096 0 4095  
## $P5 Pd104Di 104Pd 4096 0 4095  
## $P6 Pd105Di 105Pd 4096 0 4095  
## $P7 Cd106Di 106Cd\_CD57 65536 0 65535  
## $P8 Pd108Di 108Pd 32768 0 32767  
## $P9 Cd110Di 110Cd 4096 0 4095  
## $P10 Cd111Di 111Cd\_CD19 32768 0 32767  
## $P11 Cd112Di 112Cd\_CD5 32768 0 32767  
## $P12 Cd113Di 113Cd\_CD8 16384 0 16383  
## $P13 Cd114Di 114Cd\_HLADR 65536 0 65535  
## $P14 Cd116Di 116Cd\_CD3 65536 0 65535  
## $P15 Sn120Di 120Sn 32768 0 32767  
## $P16 I127Di 127I 16384 0 16383  
## $P17 Xe131Di 131Xe 4096 0 4095  
## $P18 Cs133Di 133Cs 4096 0 4095  
## $P19 Ba138Di 138Ba 32768 0 32767  
## $P20 Ce140Di 140Ce 8192 0 8191  
## $P21 Pr141Di 141Pr\_CCR6 32768 0 32767  
## $P22 Nd142Di 142Nd\_KLRG1 32768 0 32767  
## $P23 Nd143Di 143Nd\_CD127 65536 0 65535  
## $P24 Nd144Di 144Nd\_CD15 8192 0 8191  
## $P25 Nd145Di 145Nd\_CD4 65536 0 65535  
## $P26 Nd146Di 146Nd\_IgD 65536 0 65535  
## $P27 Sm147Di 147Sm\_CD11c 4096 0 4095  
## $P28 Nd148Di 148Nd\_CD16 65536 0 65535  
## $P29 Sm149Di 149Sm\_CD25 4096 0 4095  
## $P30 Nd150Di 150Nd\_CD134\_OX40 16384 0 16383  
## $P31 Eu151Di 151Eu\_CD123 8192 0 8191  
## $P32 Sm152Di 152Sm\_TCRgd 4096 0 4095  
## $P33 Eu153Di 153Eu\_CCR4 8192 0 8191  
## $P34 Sm154Di 154Sm\_TIGIT 4096 0 4095  
## $P35 Gd155Di 155Gd\_CD45RA 4096 0 4095  
## $P36 Gd156Di 156Gd\_CXCR3 32768 0 32767  
## $P37 Gd158Di 158Gd\_CD27 8192 0 8191  
## $P38 Tb159Di 159Tb\_IgG 65536 0 65535  
## $P39 Gd160Di 160Gd\_CD28 4096 0 4095  
## $P40 Dy161Di 161Dy\_CD160 16384 0 16383  
## $P41 Dy162Di 162Dy\_CD95 8192 0 8191  
## $P42 Dy163Di 163Dy\_CRTH2 4096 0 4095  
## $P43 Dy164Di 164Dy\_CD161 16384 0 16383  
## $P44 Ho165Di 165Ho\_CD85j 65536 0 65535  
## $P45 Er166Di 166Er\_TCRVa7.2 4096 0 4095  
## $P46 Er167Di 167Er\_CCR7 4096 0 4095  
## $P47 Er168Di 168Er\_ICOS 65536 0 65535  
## $P48 Tm169Di 169Tm\_NKG2A 4096 0 4095  
## $P49 Er170Di 170Er\_CD169 16384 0 16383  
## $P50 Yb171Di 171Yb\_CXCR5 4096 0 4095  
## $P51 Yb172Di 172Yb\_CD38 4096 0 4095  
## $P52 Yb173Di 173Yb\_CD141 32768 0 32767  
## $P53 Yb174Di 174Yb\_CD279\_PD-1 4096 0 4095  
## $P54 Lu175Di 175Lu\_CD14 8192 0 8191  
## $P55 Yb176Di 176Yb\_CD56 4096 0 4095  
## $P56 BCKG190Di 190BCKG 4096 0 4095  
## $P57 Ir191Di 191Ir\_DNA1 32768 0 32767  
## $P58 Ir193Di 193Ir\_DNA2 65536 0 65535  
## $P59 Pt194Di 194Pt\_Cisplatin 131072 0 131071  
## $P60 Pt195Di 195Pt\_Cisplatin 32768 0 32767  
## $P61 Pt198Di 198Pt 4096 0 4095  
## $P62 Pb208Di 208Pb 65536 0 65535  
## $P63 Bi209Di 209Bi\_CD11b 32768 0 32767  
## $P64 Center <NA> 16384 0 16383  
## $P65 Offset <NA> 4096 0 4095  
## $P66 Width <NA> 4096 0 4095  
## $P67 Residual <NA> 4096 0 4095

Now that we have read all the data into R we are ready for gating so that we are left with only live cells.

## Beads gating

Beads channels can be “Ce140Di”, “Eu151Di”, “Eu153Di”, “Ho165Di” or “Lu175Di” or maybe something else in your dataset. Here we use the function grepl() to get the correct name used in params. the string “140|151|153|165|175” tells grepl() to look for 140, 151, 153, 165 or 175 in the column name of the data.frame params. R is case sensitive, so it is important to write the name correctly.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#gating on Beads ----  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
bead\_channels <- as.character(params$name[grepl("140|151|153|165|175", params$name)])  
bead\_channels

## [1] "Ce140Di" "Eu151Di" "Eu153Di" "Ho165Di" "Lu175Di"

here we choose to do the beads gating on Ce140Di, first making the list beads\_data where I have arc\_sinh transformed the signal data.

x <- "Ce140Di"  
beads\_data <- arc\_sinh\_transform\_selected\_channels(fcs\_data = fcs\_data, channels = bead\_channels)

The variable beads\_data is a list that contain as many elements as fcs file used

length(beads\_data)

## [1] 6

where each element in the list contains a matrix with the observation of that file. By using the function head() we can look at the six first rows in the first element ([[1]]) in the list beads\_data

head(beads\_data[[1]])

## Ce140Di Eu151Di Eu153Di Ho165Di Lu175Di Time  
## 1 0.00000000 0.0000000 0.000000 0.4203946 0.08786318 0.039  
## 2 0.00000000 0.2816456 0.000000 0.4150252 0.00000000 2.331  
## 3 6.95949234 6.7431394 6.953601 6.9163646 6.92496600 2.969  
## 4 0.16807481 0.0000000 1.556992 0.0000000 0.07307996 3.945  
## 5 0.08145197 0.0000000 0.000000 2.3374715 0.05895377 6.198  
## 6 0.00000000 0.4823033 3.496447 0.2487503 0.15427276 6.979

by using the function number\_of\_events\_raw\_data() we get a vector with number of events in the dataset

number\_of\_events\_raw\_data <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
number\_of\_events\_raw\_data

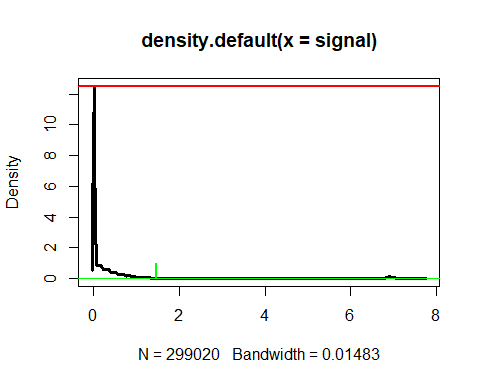
## Ko\_Ma\_A\_70\_578K1\_Panel1 Ko\_Ma\_A\_72\_582K1\_Panel1   
## 299020 300000   
## M\_Fe\_B\_20\_T1\_FHI155\_Panel1 M\_Fe\_B\_43\_T1\_FHI004\_Panel1   
## 298690 300000   
## Ref1\_Panel1\_230621 S\_ Fe\_B\_22\_T1\_FHI111T1\_Panel1   
## 300000 300000

since it takes time and memory to plot many events, we here choose to randomly plot 10000 events per file. This events are found by the function random\_events() which gives a list of as many elements that you have files. Each element of the list contains a vector with 10000 numbers representing the events to plot.

random\_events\_for\_plotting <- random\_events(number\_of\_events\_raw\_data, n = 10000)

When performing beads gating the purpose is to exclude events with high signal.

Here the function find\_gate\_perc\_height\_upper\_noise() takes the dataset that you want to make the gates based on, here beads\_data in column channel x, and a value upper\_perc\_height, here 0.001, the proportion height of the max of the density curve. The max of the density curve is here shown with the red line. while 0.001 times this height is shown with the green line. The gate should be placed where the green horizontal line crosses the density curve. Here shown by the green vertical line. The function find\_gate\_perc\_height\_upper\_noise() take a list of matrices and return a vector in the same length as the list, i.e. one gate per file.



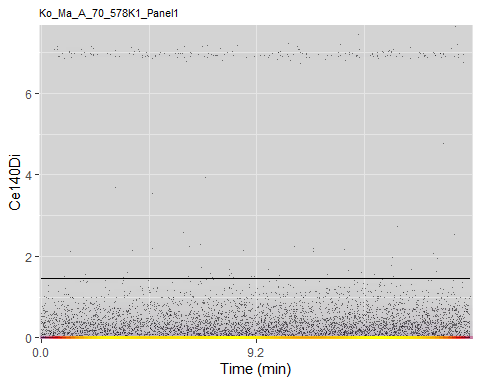
Here we use the function find\_gate\_perc\_height\_upper\_noise() to find upper\_gate for noise gating based on the channel Ce140Di. The upper\_gate values are stored in the variable upper\_gate\_Ce140Di.

upper\_gate\_Ce140Di <- find\_gate\_perc\_height\_upper\_noise(data = beads\_data, channel = x, upper\_perc\_height = 0.001)

Then we make a time signal plot where the signal is plotted against time and the gate is shown. Even though we here only make the plot for those events we want to plot, saved in random\_events\_for\_plotting, the gate is made on all datapoints.

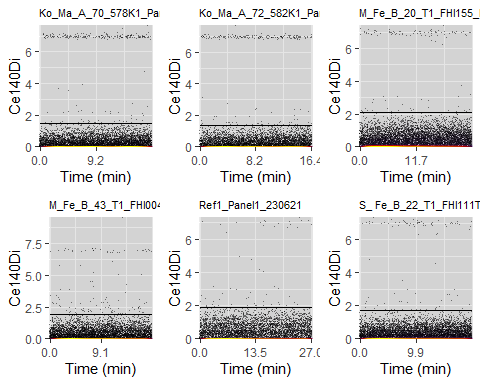
time\_signal\_plots <- time\_signal\_plot(data = beads\_data, random\_events = random\_events\_for\_plotting,   
 channel = x, plot\_title = file\_names, upper\_gate = upper\_gate\_Ce140Di)  
   
time\_signal\_plots[1] # to see first plot

## [[1]]



All the plots can be plotted together by using the function plotSignal(). I prefer to save this plot to a tiff file instead of plotting it in R. In that way you have saved it to later evaluations.

plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

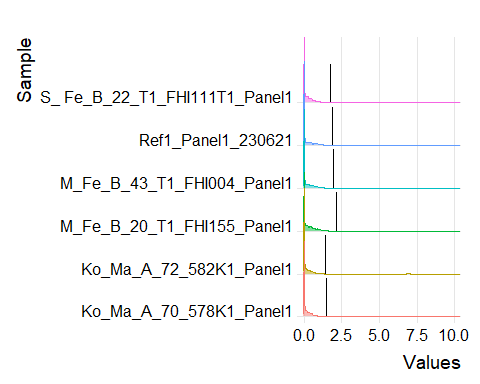
I prefer to save this plot to a tiff file instead of plotting it in R. In that way you have saved it to later evaluations. To run the lines remove “#”.

# tiff(fs::path(outFigPath, paste0("Signal\_fig1\_bead\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()

The gate could also be plotted in a density plot made by the function density\_plot(). Again you could either see it directly in the RStudio or save it to a tiff file. If saving it, you do not have to run the script again to see what you did.

density\_plots <- density\_plot(data = beads\_data, channel = x, plot\_title = file\_names, upper\_gate = upper\_gate\_Ce140Di)  
density\_plots # to see the plots

## Picking joint bandwidth of 0.0181



# tiff(fs::path(outFigPath, paste0("fig1\_bead\_gating.tiff")))  
# density\_plots  
# dev.off()

Before continuing with the script, it should be decided if the gate is ok or something should be adjusted. If ok, proceed using the gate and adjust the dataset by excluding the events with signal above upper\_gate\_Ce140Di.

* The function events\_to\_keep\_after\_gating() find which events to keep per dataset.
* The function update\_data\_based\_on\_events\_to\_keep() update the fcs\_data
* The number of events per file after gating are found by number\_of\_events()
* The percentages lost by this gating and for full dataset is calculated and saved in the matrices percent\_lost\_from\_full\_dataset and percent\_lost\_from\_full\_dataset
* A new vector of random\_events\_for\_plotting is made by the function random\_events()

events\_to\_keep\_after\_gating <- events\_to\_keep(data = beads\_data, channel = "Ce140Di", upper\_gate = upper\_gate\_Ce140Di)  
   
 #overwrite the raw dataset (will take lot of space if we make one new each time, and do not need it)  
 fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_gating)  
 number\_of\_events\_after\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_each\_gating[as.character(file\_names),"Ce140Di"] <- number\_of\_events\_after\_gating/number\_of\_events\_raw\_data \* 100   
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Ce140Di"] <- number\_of\_events\_after\_gating/number\_of\_events\_raw\_data \* 100   
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_after\_gating)  
   
   
 number\_of\_events\_after\_beads\_gating <- number\_of\_events\_after\_gating

The percent of events lost for this gating is saved in the matrix percent\_lost\_each\_gating, and can be printed here as

percent\_lost\_each\_gating[as.character(file\_names),"Ce140Di"]

## [1] 97.64029 97.96200 98.92430 99.04367 99.41833 99.05067

Since we are finished working with beads\_data, remove them to save space in memory

rm(beads\_data)

# clean\_up\_data

* the clean up channel names can be found by using grep of the string “Center|Offset|Width|Residual|Event|Ir191|Ir193|Pt195Di|Pt194Di”
* a arc\_sinh transformed dataset with this channels are made using the function arc\_sinh\_transform\_selected\_channels()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #clean\_up\_data  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 clean\_up\_channels <- as.character(params$name[grep("Center|Offset|Width|Residual|Event|Ir191|Ir193|Pt195Di|Pt194Di", params$name)]) #Pt194Di og 195 tilsvarer Cis  
 clean\_up\_data <- arc\_sinh\_transform\_selected\_channels(fcs\_data = fcs\_data, channels = clean\_up\_channels)

to be able to calculate the number of events lost by this gating we make the variable number\_of\_events\_before\_clean\_up\_gating using the function number\_of\_events()

number\_of\_events\_before\_clean\_up\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)

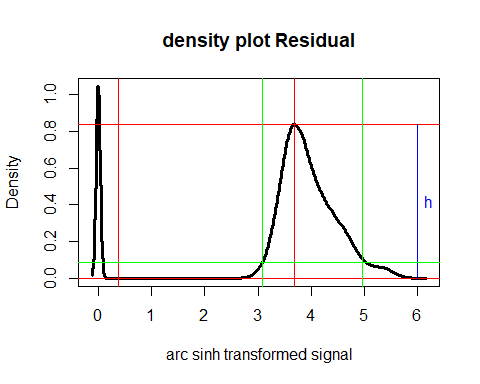
# Gating on Residual

The first clean up gating is on residuals.

First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on Residual+ ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_residual\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_residual\_gating)

Here the function find\_gaussian\_gates\_second\_top() takes the dataset that you want to make the gates based on, here clean\_up\_data in column channel “Residual”, and two values lower\_perc\_height and upper\_perc\_height, here 10 and 10, the percentage height of the max of the density curve. The maximum and minimum of the density curve is here shown with the red line, while the height between these lines are shown in blue. 0.1 times this height is shown with the green line. The gate should be placed where the green horizontal line crosses the density curve. Here shown by the green vertical lines. The function find\_gaussian\_gates\_second\_top() take a list of matrices and return two vectors which both has the same length as the list, i.e. two gates per file.



The function find\_gaussian\_gates\_second\_top() are used to find lower and upper gate for the positiv signal of the residuals. It takes the variables

* data: here the arc sinh transformed clean\_up\_data are used
* channel: which marker/channel you want to gate, here “Residual”.
* lower\_gate\_percent: tells where the lower gate is placed. See figure and explanation above, here 10 %
* upper\_gate\_percent: tells where the upper gate is placed. See figure and explanation above, here 10 %

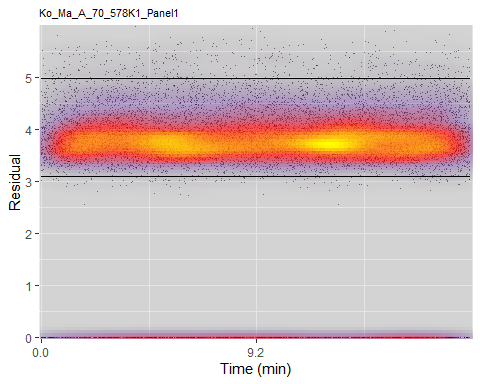
R code to fine the lower and upper residual gates:

residual\_gates <- find\_gaussian\_gates\_second\_top(data = clean\_up\_data, channel = "Residual", lower\_gate\_percent = 10, upper\_gate\_percent = 10)

make a time signal plot where the signal is plotted against time and the gate is shown.

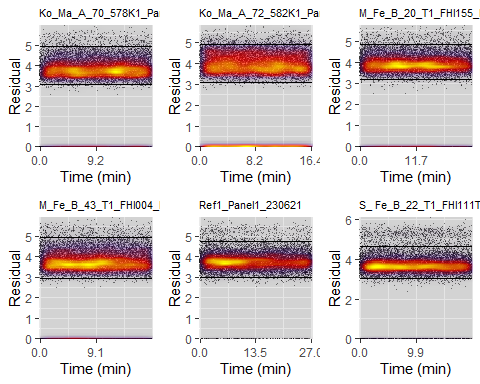
time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Residual", plot\_title = file\_names, lower\_gate = residual\_gates$lower\_gate, upper\_gate = residual\_gates$upper\_gate)  
   
time\_signal\_plots[1] # to see first plot

## [[1]]



All the plots can be plotted together by using the function plotSignal().

plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

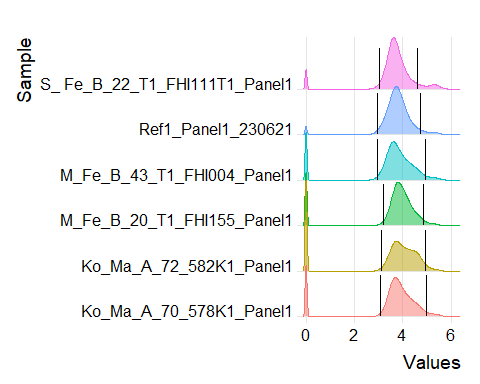
I prefer to save this plot to a tiff file instead of plotting it in R. In that way you have saved it to later evaluations.

# tiff(fs::path(outFigPath, paste0("Signal\_fig2\_residal\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()

The gate could also be plotted in a density plot made by the function density\_plot(). Again, you could either see it directly in the RStudio or save it to a tiff file. If saving it, you do not have to run the script again to see what you did.

density\_plots <- density\_plot(data = clean\_up\_data, "Residual", plot\_title = file\_names, lower\_gate = residual\_gates$lower\_gate, upper\_gate = residual\_gates$upper\_gate)  
density\_plots # to see the plots

## Picking joint bandwidth of 0.0369



# tiff(fs::path(outFigPath, paste0("fig1\_bead\_gating.tiff")))  
# density\_plots  
# dev.off()

Before continuing with the script, it should be decided if the gate is ok or something should be adjusted. If ok, proceed using the gate and adjust the dataset by excluding the events that do not have a signal between lower and upper residual gates.

* The function events\_to\_keep\_after\_gating() find which events to keep per dataset.
* The function update\_data\_based\_on\_events\_to\_keep() update the fcs\_data and the clean\_up\_data
* The number of events per file after gating are found by number\_of\_events()
* The percentages lost by this gating and for full dataset is calculated and saved in the matrices percent\_lost\_from\_full\_dataset and percent\_lost\_from\_full\_dataset
* A new vector of random\_events\_for\_plotting is made by the function random\_events()

events\_to\_keep\_after\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Residual", lower\_gate = residual\_gates$lower\_gates,  
 upper\_gate = residual\_gates$upper\_gate)  
  
 fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_gating)  
 clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_gating)  
 number\_of\_events\_after\_residual\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Residual"] <- number\_of\_events\_after\_residual\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
 percent\_lost\_each\_gating[as.character(file\_names),"Residual"] <- number\_of\_events\_after\_residual\_gating/number\_of\_events\_before\_residual\_gating \* 100 #percent remaining from bead gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names),"Residual"]

## [1] 84.19908 76.51028 83.83834 85.21528 92.03366 86.20066

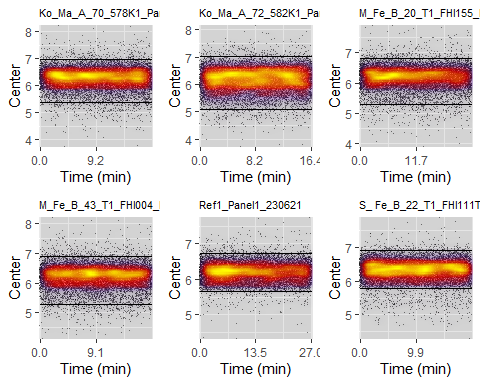
# Gating on Center

The second clean up gating is on center.

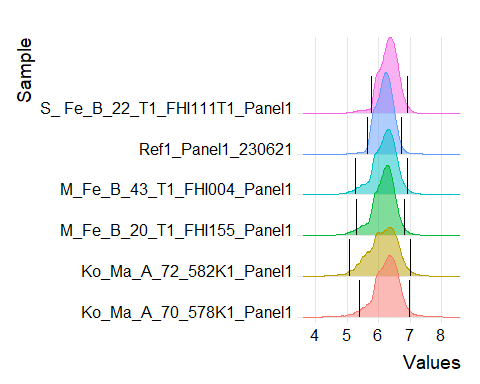
First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on Center+ ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_center\_gating <- number\_of\_events\_after\_residual\_gating  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_center\_gating)

Center is gated in the same way as Residuals by using the function find\_gaussian\_gates\_second\_top(). Also, here we have chosen to use 10% gaussian gates for both lower and upper gate. This must be adjusted to what level you want to use for your data. Plots are made and if wanted saved in the same way as for Residuals.



FALSE TableGrob (2 x 3) "arrange": 6 grobs  
FALSE z cells name grob  
FALSE 1 1 (1-1,1-1) arrange gtable[layout]  
FALSE 2 2 (1-1,2-2) arrange gtable[layout]  
FALSE 3 3 (1-1,3-3) arrange gtable[layout]  
FALSE 4 4 (2-2,1-1) arrange gtable[layout]  
FALSE 5 5 (2-2,2-2) arrange gtable[layout]  
FALSE 6 6 (2-2,3-3) arrange gtable[layout]



and the number of events to keep based on the center gating are updated together with the datasets fcs\_data and clean\_up\_data. Also, the number\_of\_events\_after\_center\_gating and percent lost with this gating is updated similar to the Residual gating.

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Center"]

## [1] 93.97960 95.73499 94.05182 94.40642 95.84000 93.99095

# Gating on Offset

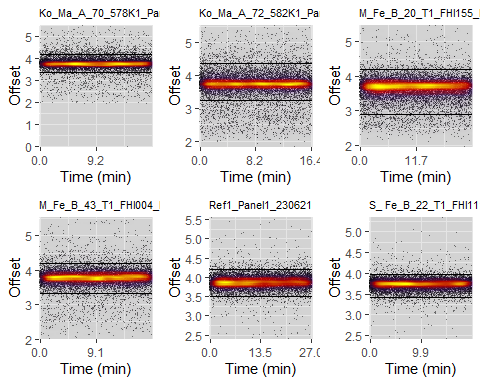
The third clean up gating is on offset.

First the number of events left before this gating are found and a new set of random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#gating on Offset+ ----  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
number\_of\_events\_before\_offset\_gating <- number\_of\_events\_after\_center\_gating  
random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_offset\_gating)

Also offset is gated in the same way as Residuals and Center by using the function find\_gaussian\_gates\_second\_top(). But here we have chosen to include more of the data using 7 % gaussian gates for both lower and upper gate. This must be adjusted to what level you want to use for your data. Plots are made and if wanted saved in the same way as for Residuals.

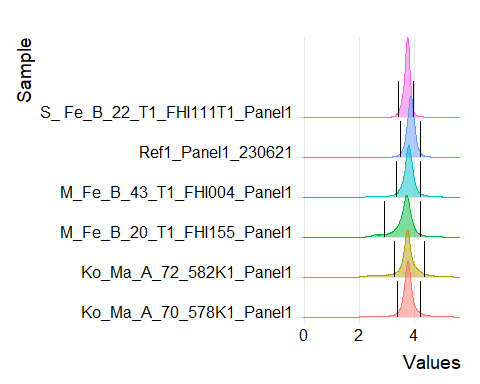
#update lower\_gate\_percent, upper\_gate\_percent  
offset\_gates <- find\_gaussian\_gates\_second\_top(data = clean\_up\_data, channel = "Offset", lower\_gate\_percent = 7, upper\_gate\_percent = 7)  
  
time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Offset", plot\_title = file\_names, lower\_gate = offset\_gates$lower\_gate, upper\_gate = offset\_gates$upper\_gate)  
plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig4\_offset\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()  
  
density\_plots <- density\_plot(data = clean\_up\_data, "Offset", plot\_title = file\_names, lower\_gate = offset\_gates$lower\_gate, upper\_gate = offset\_gates$upper\_gate)  
density\_plots

## Picking joint bandwidth of 0.0133



# tiff(fs::path(outFigPath, paste0("fig4\_offset\_gating.tiff")))  
# print(density\_plots)  
# dev.off()

The number of events to keep based on the offset gating are updated together with the datasets fcs\_data and clean\_up\_data, number\_of\_events\_after\_offset\_gating and percent lost with this gating is updated similar to the previous gatings.

events\_to\_keep\_after\_offset\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Offset", lower\_gate = offset\_gates$lower\_gate, upper\_gate = offset\_gates$upper\_gate)  
  
fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_offset\_gating)  
clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_offset\_gating)  
number\_of\_events\_after\_offset\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
percent\_lost\_from\_full\_dataset[as.character(file\_names),"Offset"] <- number\_of\_events\_after\_offset\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
percent\_lost\_each\_gating[as.character(file\_names),"Offset"] <- number\_of\_events\_after\_offset\_gating/number\_of\_events\_before\_offset\_gating \* 100 #percent remaining from center gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Offset"]

## [1] 88.16176 84.09016 90.47548 88.65620 94.19027 93.78788

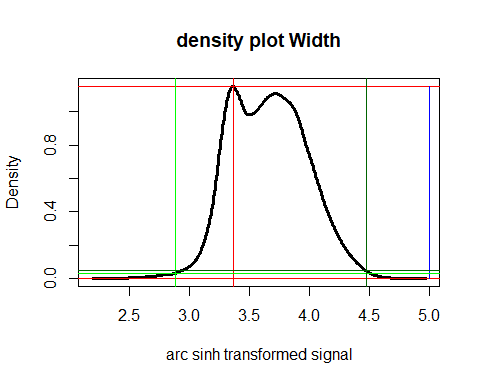
# Gating on Width

The fourth clean up gating is on Width

First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#gating on Width+ ----  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
number\_of\_events\_before\_width\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_width\_gating)

Here the function find\_gaussian\_gates\_highest\_top() takes the dataset that you want to make the gates based on, here clean\_up\_data in column channel “Width”, and two values lower\_perc\_height and upper\_perc\_height, here 0.025 and 0.04, the proportion height of the max of the density curve. The max of the density curve is here shown with the red line. while 0.025 times this height is shown with the green line, and 0.4 times the height is shown with a dark green line. The gate should be placed where the green horizontal line crosses the density curve. Here shown by the green vertical line. The function function find\_gaussian\_gates\_highest\_top() take a list of matrices and return two vector in the same length as the list, i.e. two gate per file.



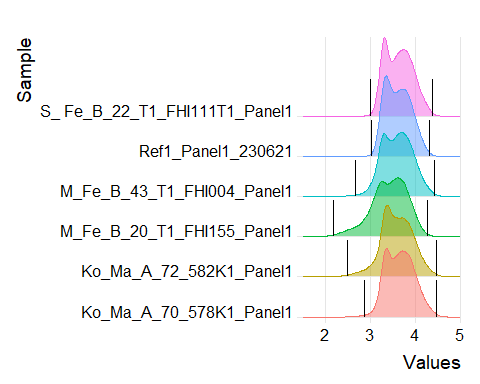
The find\_gaussian\_gates\_highest\_top() are used to find lower gate for the positiv signal of the width. It takes the variables

* data: here the arc sinh transformed clean\_up\_data are used
* channel: which marker/channel you want to gate, here “Width”.
* lower\_gate\_percent: tells where the lower gate is placed. See figure and explanation above, here 2.5 %
* upper\_gate\_percent: tells where the upper gate is placed. See figure and explanation above, here 4 %

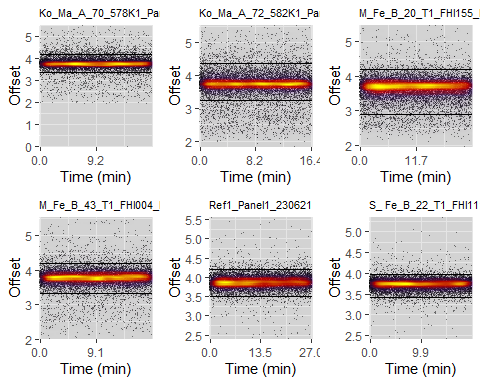
R code to fine the lower and upper residual gates:

#update lower\_gate\_percent, upper\_gate\_percent  
 width\_gates <- find\_gaussian\_gates\_highest\_top(data = clean\_up\_data, channel = "Width", lower\_gate\_percent = 2.5, upper\_gate\_percent = 4)  
  
 density\_plots <- density\_plot(data = clean\_up\_data, "Width", plot\_title = file\_names, lower\_gate = width\_gates$lower\_gate, upper\_gate = width\_gates$upper\_gate)  
 density\_plots

## Picking joint bandwidth of 0.0256



# tiff(fs::path(outFigPath, paste0("fig5\_width\_gating.tiff")))  
 # print(density\_plots)  
 # dev.off()  
 #time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Width", plot\_title = file\_names, lower\_gate = width\_gates$lower\_gate, upper\_gate = width\_gates$upper\_gate)  
 #time\_signal\_plots # to see all plots  
 #time\_signal\_plots[1] # to see first plot  
 plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig5\_width\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()

The number of events to keep based on the width gating are updated together with the datasets fcs\_data and clean\_up\_data, number\_of\_events\_after\_offset\_gating and percent lost with this gating is updated similar to the previous gatings.

events\_to\_keep\_after\_width\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Width", lower\_gate = width\_gates$lower\_gate, upper\_gate = width\_gates$upper\_gate)  
#   
 fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_width\_gating)  
 clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_width\_gating)  
 number\_of\_events\_after\_width\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Width"] <- number\_of\_events\_after\_width\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
 percent\_lost\_each\_gating[as.character(file\_names),"Width"] <- number\_of\_events\_after\_width\_gating/number\_of\_events\_before\_width\_gating \* 100 #percent remaining from offset gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Width"]

## [1] 99.11921 99.14537 99.25568 99.22188 99.54155 99.61027

# Gating on Event

The fifth clean up gating is on event.

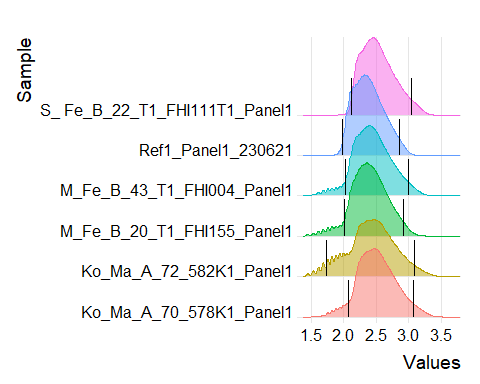
First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on Event+ ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_event\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_event\_gating)

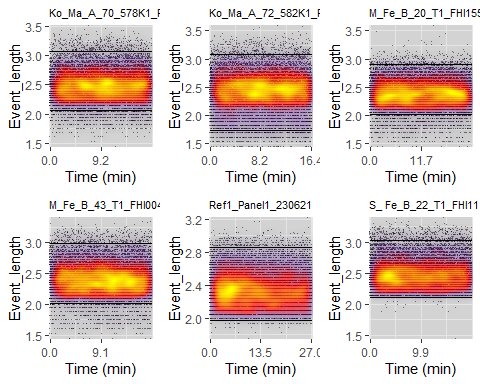
Also event is gated by using the function find\_gaussian\_gates\_second\_top(). But here we have chosen to include less of the data using 20 % gaussian gates for both lower and upper gate. This must be adjusted to what level you want to use for your data. Plots are made and if wanted saved in the same way as for Residuals.

#update lower\_gate\_percent, upper\_gate\_percent  
 EventGates <- find\_gaussian\_gates\_second\_top(data = clean\_up\_data, channel = "Event\_length", lower\_gate\_percent = 20, upper\_gate\_percent = 20)  
 density\_plots <- density\_plot(data = clean\_up\_data, "Event\_length", plot\_title = file\_names, lower\_gate = EventGates$lower\_gate, upper\_gate = EventGates$upper\_gate)  
 density\_plots

## Picking joint bandwidth of 0.022



# tiff(fs::path(outFigPath, paste0("fig6\_event\_gating.tiff")))  
 # print(density\_plots)  
 # dev.off()  
 time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Event\_length", plot\_title = file\_names, lower\_gate = EventGates$lower\_gate, upper\_gate = EventGates$upper\_gate)  
 #time\_signal\_plots # to see all plots  
 #time\_signal\_plots[1] # to see first plot  
   
 plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig6\_event\_gating.tiff")))  
 # plotSignal(plot\_list = time\_signal\_plots)  
 # dev.off()

The number of events to keep based on the event gating are updated together with the datasets fcs\_data and clean\_up\_data, number\_of\_events\_after\_offset\_gating and percent lost with this gating is updated similar to the previous gatings.

events\_to\_keep\_after\_event\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Event\_length", upper\_gate = EventGates$upper\_gate)  
 #percent\_to\_keep\_this\_gating(kept\_events = events\_to\_keep\_after\_event\_gating, file\_names = file\_names)  
  
fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_event\_gating)  
 clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_event\_gating)  
 number\_of\_events\_after\_event\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Event\_length"] <- number\_of\_events\_after\_event\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
 percent\_lost\_each\_gating[as.character(file\_names),"Event\_length"] <- number\_of\_events\_after\_event\_gating/number\_of\_events\_before\_event\_gating \* 100 #percent remaining from width gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Event\_length"]

## [1] 95.79171 96.33974 96.58029 96.20826 96.96987 95.34054

# Gating on CIS

The sixth clean up gating is on CIS, live cells.

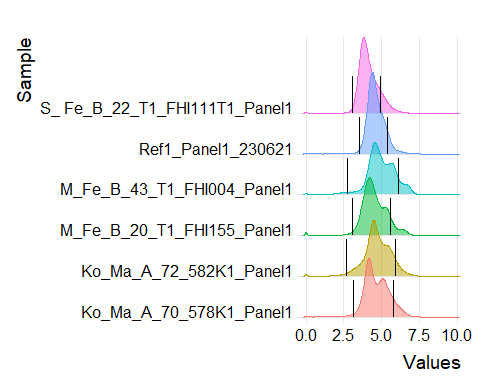
First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on Live Dead---- Cis,  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_cis\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_cis\_gating)

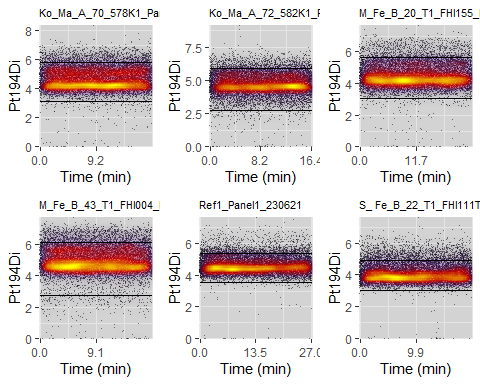
Also CIS is gated by using the function find\_gaussian\_gates\_second\_top(). But here we have chosen to use 5 % gaussian gates for the lower gate and 20 % for the upper gate. This must be adjusted to what level you want to use for your data. Plots are made and if wanted saved in the same way as for Residuals.

#update lower\_gate\_percent, upper\_gate\_percent  
 cis\_gates <- find\_gaussian\_gates\_second\_top(data = clean\_up\_data, channel = "Pt194Di", lower\_gate\_percent = 5, upper\_gate\_percent = 30)  
# if you want to overwrite the gate found this could be done like this:  
 # tvungetLavereCISgate <- 0.5 #sett inn NA hvis du heller vil bruke gaussian gate for CIS. evt annet tall..  
 # if(!is.na(tvungetLavereCISgate)){  
# cis\_gates$lower\_gates <- rep(tvungetLavereCISgate, length(cis\_gates$lower\_gates))  
# }  
  
 density\_plots <- density\_plot(data = clean\_up\_data, "Pt194Di", plot\_title = file\_names, lower\_gate = cis\_gates$lower\_gate, upper\_gate = cis\_gates$upper\_gate)  
 density\_plots

## Picking joint bandwidth of 0.0582



# tiff(fs::path(outFigPath, paste0("fig7\_cis\_gating.tiff")))  
 # print(density\_plots)  
 # dev.off()  
 time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Pt194Di", plot\_title = file\_names, lower\_gate = cis\_gates$lower\_gate, upper\_gate = cis\_gates$upper\_gate)  
 #time\_signal\_plots # to see all plots  
 #time\_signal\_plots[1] # to see first plot  
 #time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Pt194Di", plot\_title = file\_names, lower\_gate = NA, upper\_gate = NA)  
 #time\_signal\_plots # to see all plots  
   
 plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig7\_cis\_gating.tiff")))  
 # plotSignal(plot\_list = time\_signal\_plots)  
 # dev.off()

The number of events to keep based on the cis gating are updated together with the datasets fcs\_data and clean\_up\_data, number\_of\_events\_after\_offset\_gating and percent lost with this gating is updated similar to the previous gatings.

events\_to\_keep\_after\_cis\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Pt194Di", lower\_gate = cis\_gates$lower\_gate, upper\_gate = cis\_gates$upper\_gate)  
  
  
 fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_cis\_gating)  
 clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_cis\_gating)  
 number\_of\_events\_after\_cis\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Pt194Di"] <- number\_of\_events\_after\_cis\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
 percent\_lost\_each\_gating[as.character(file\_names),"Pt194Di"] <- number\_of\_events\_after\_cis\_gating/number\_of\_events\_before\_cis\_gating \* 100 #percent remaining from event gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Pt194Di"]

## [1] 88.36916 89.78884 88.24484 87.34849 89.18104 85.48538

# gating on DNA1, Ir 191

This is the last and seventh clean up gating.

First the number of events left before this gating are found and a new set random cells for plotting made.

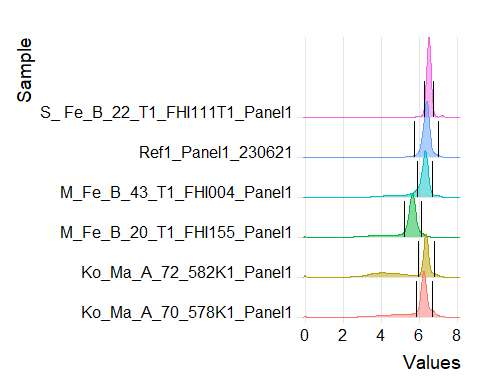
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on DNA1, Ir191Di+ ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_Ir191Di\_gating <- number\_of\_events(fcs\_data, file\_names = file\_names)  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_Ir191Di\_gating)

Also, here gaussian gate for the second top is found, but instead of using the variables lower\_gate\_percent and upper\_gate\_percent another variable is used. This is perc\_include. In this version of the function, we assume the signals to come from two normal distribution and that we want to find the gate that include 99 % of the normal distribution with the highest top. This way of finding the gates involves some computation and will take some time. The best approach is to test on some files and then run it for all files later.

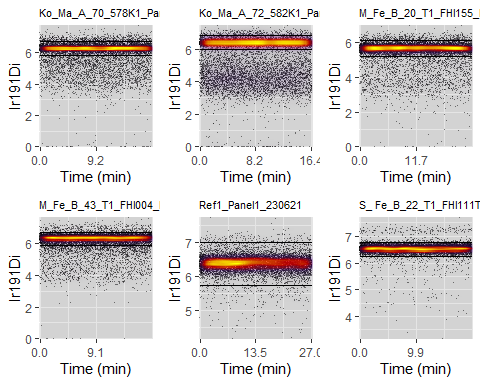
Ir191di\_gates <- find\_gaussian\_gates\_second\_top(data = clean\_up\_data, channel = "Ir191Di", lower\_gate\_percent = NA, upper\_gate\_percent = NA, perc\_included = 0.99, main\_top\_to\_left = F)

FALSE VGLM linear loop 1 : loglikelihood = -7995.823  
FALSE VGLM linear loop 2 : loglikelihood = -7442.573  
FALSE VGLM linear loop 3 : loglikelihood = -7370.461  
FALSE VGLM linear loop 4 : loglikelihood = -7357.152  
FALSE VGLM linear loop 5 : loglikelihood = -7352.573  
FALSE VGLM linear loop 6 : loglikelihood = -7350.897  
FALSE VGLM linear loop 7 : loglikelihood = -7350.259  
FALSE VGLM linear loop 8 : loglikelihood = -7350.013  
FALSE VGLM linear loop 9 : loglikelihood = -7349.917  
FALSE VGLM linear loop 10 : loglikelihood = -7349.88  
FALSE VGLM linear loop 11 : loglikelihood = -7349.865  
FALSE VGLM linear loop 12 : loglikelihood = -7349.859  
FALSE VGLM linear loop 13 : loglikelihood = -7349.857  
FALSE VGLM linear loop 14 : loglikelihood = -7349.856  
FALSE VGLM linear loop 15 : loglikelihood = -7349.856  
FALSE VGLM linear loop 1 : loglikelihood = -11242.68  
FALSE VGLM linear loop 2 : loglikelihood = -11155.96  
FALSE VGLM linear loop 3 : loglikelihood = -11143.28  
FALSE VGLM linear loop 4 : loglikelihood = -11140.32  
FALSE VGLM linear loop 5 : loglikelihood = -11139.54  
FALSE VGLM linear loop 6 : loglikelihood = -11139.34  
FALSE VGLM linear loop 7 : loglikelihood = -11139.29  
FALSE VGLM linear loop 8 : loglikelihood = -11139.28  
FALSE VGLM linear loop 9 : loglikelihood = -11139.27  
FALSE VGLM linear loop 10 : loglikelihood = -11139.27  
FALSE VGLM linear loop 1 : loglikelihood = -6456.714  
FALSE VGLM linear loop 2 : loglikelihood = -5498.344  
FALSE VGLM linear loop 3 : loglikelihood = -5388.993  
FALSE VGLM linear loop 4 : loglikelihood = -5378.259  
FALSE VGLM linear loop 5 : loglikelihood = -5375.42  
FALSE VGLM linear loop 6 : loglikelihood = -5374.668  
FALSE VGLM linear loop 7 : loglikelihood = -5374.478  
FALSE VGLM linear loop 8 : loglikelihood = -5374.431  
FALSE VGLM linear loop 9 : loglikelihood = -5374.42  
FALSE VGLM linear loop 10 : loglikelihood = -5374.417  
FALSE VGLM linear loop 11 : loglikelihood = -5374.416  
FALSE VGLM linear loop 12 : loglikelihood = -5374.416  
FALSE VGLM linear loop 1 : loglikelihood = -5254.295  
FALSE VGLM linear loop 2 : loglikelihood = -4959.776  
FALSE VGLM linear loop 3 : loglikelihood = -4942.553  
FALSE VGLM linear loop 4 : loglikelihood = -4939.944  
FALSE VGLM linear loop 5 : loglikelihood = -4939.456  
FALSE VGLM linear loop 6 : loglikelihood = -4939.366  
FALSE VGLM linear loop 7 : loglikelihood = -4939.35  
FALSE VGLM linear loop 8 : loglikelihood = -4939.347  
FALSE VGLM linear loop 9 : loglikelihood = -4939.347  
FALSE VGLM linear loop 10 : loglikelihood = -4939.346  
FALSE VGLM linear loop 1 : loglikelihood = -45937.12  
FALSE Applying Greenstadt modification to 10000 matrices  
FALSE VGLM linear loop 2 : loglikelihood = -1445858  
FALSE Taking a modified step.....  
FALSE VGLM linear loop 2 : loglikelihood = -45750.5  
FALSE Applying Greenstadt modification to 10000 matrices  
FALSE VGLM linear loop 3 : loglikelihood = -39445.66  
FALSE Taking a modified step.......  
FALSE VGLM linear loop 3 : loglikelihood = -45730.61  
FALSE VGLM linear loop 4 : loglikelihood = -40732.52  
FALSE VGLM linear loop 5 : loglikelihood = -35728.82  
FALSE VGLM linear loop 6 : loglikelihood = -30729.66  
FALSE VGLM linear loop 7 : loglikelihood = -25727.16  
FALSE VGLM linear loop 8 : loglikelihood = -20764.67  
FALSE VGLM linear loop 9 : loglikelihood = -15861.01  
FALSE VGLM linear loop 10 : loglikelihood = -11128.58  
FALSE VGLM linear loop 11 : loglikelihood = -6806.858  
FALSE VGLM linear loop 12 : loglikelihood = -3381.716  
FALSE VGLM linear loop 13 : loglikelihood = -1509.868  
FALSE VGLM linear loop 14 : loglikelihood = -1054.556  
FALSE VGLM linear loop 15 : loglikelihood = -1032.61  
FALSE VGLM linear loop 16 : loglikelihood = -1032.563  
FALSE VGLM linear loop 17 : loglikelihood = -1032.563  
FALSE VGLM linear loop 1 : loglikelihood = -841.9967  
FALSE VGLM linear loop 2 : loglikelihood = 2045.32  
FALSE VGLM linear loop 3 : loglikelihood = 3533.864  
FALSE VGLM linear loop 4 : loglikelihood = 4457.501  
FALSE VGLM linear loop 5 : loglikelihood = 4763.386  
FALSE VGLM linear loop 6 : loglikelihood = 4824.827  
FALSE VGLM linear loop 7 : loglikelihood = 4835.939  
FALSE VGLM linear loop 8 : loglikelihood = 4836.305  
FALSE VGLM linear loop 9 : loglikelihood = 4836.308  
FALSE VGLM linear loop 10 : loglikelihood = 4836.308

density\_plots <- density\_plot(data = clean\_up\_data, "Ir191Di", plot\_title = file\_names, lower\_gate = Ir191di\_gates$lower\_gate, upper\_gate = Ir191di\_gates$upper\_gate)  
  
density\_plots



# tiff(fs::path(outFigPath, paste0("fig8\_Ir191\_gating.tiff")))  
# print(density\_plots)  
# dev.off()  
time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Ir191Di", plot\_title = file\_names, lower\_gate = Ir191di\_gates$lower\_gate, upper\_gate = Ir191di\_gates$upper\_gate)  
#time\_signal\_plots # to see all plots  
#time\_signal\_plots[1] # to see first plot  
  
  
plotSignal(plot\_list = time\_signal\_plots)



FALSE TableGrob (2 x 3) "arrange": 6 grobs  
FALSE z cells name grob  
FALSE 1 1 (1-1,1-1) arrange gtable[layout]  
FALSE 2 2 (1-1,2-2) arrange gtable[layout]  
FALSE 3 3 (1-1,3-3) arrange gtable[layout]  
FALSE 4 4 (2-2,1-1) arrange gtable[layout]  
FALSE 5 5 (2-2,2-2) arrange gtable[layout]  
FALSE 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig8\_Ir191\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()

events\_to\_keep\_after\_Ir191Di\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Ir191Di", lower\_gate = Ir191di\_gates$lower\_gate, upper\_gate = Ir191di\_gates$upper\_gate)  
 #percent\_to\_keep\_this\_gating(kept\_events = events\_to\_keep\_after\_Ir191Di\_gating, file\_names = file\_names)  
  
  
fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_Ir191Di\_gating)  
clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_Ir191Di\_gating)  
number\_of\_events\_after\_Ir191Di\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
percent\_lost\_from\_full\_dataset[as.character(file\_names),"Ir191Di"] <- number\_of\_events\_after\_Ir191Di\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
percent\_lost\_each\_gating[as.character(file\_names),"Ir191Di"] <- number\_of\_events\_after\_Ir191Di\_gating/number\_of\_events\_before\_Ir191Di\_gating \* 100 #percent remaining from cis gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Ir191Di"]

## [1] 72.24777 57.64281 78.87213 76.80667 96.30529 92.68086

# gating on DNA1, Ir 193

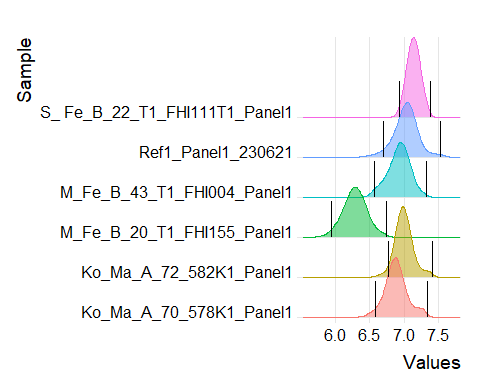
It is optional to also gate on the second DNA1 channel. First the number of events left before this gating are found and a new set random cells for plotting made.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #gating on DNA1, Ir193Di+ ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 number\_of\_events\_before\_Ir193Di\_gating <- number\_of\_events(fcs\_data, file\_names = file\_names)  
 random\_events\_for\_plotting <- random\_events(number\_of\_events\_before\_Ir193Di\_gating)

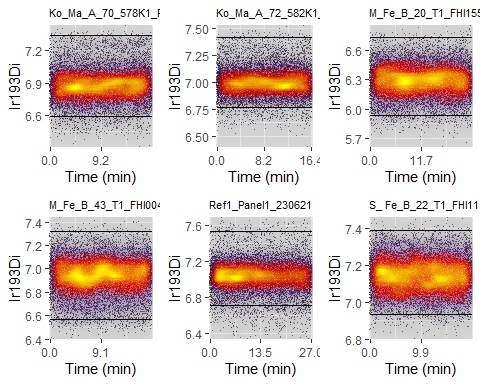
Here we use another function called find\_gaussian\_gates\_highest\_top() which again take the variables lower\_gate\_percent and upper\_gate\_percent.

#update lower\_gate\_percent, upper\_gate\_percent  
 Ir193di\_gates <- find\_gaussian\_gates\_highest\_top(data = clean\_up\_data, channel = "Ir193Di", lower\_gate\_percent = 15, upper\_gate\_percent = 5)  
  
  
density\_plots <- density\_plot(data = clean\_up\_data, "Ir193Di", plot\_title = file\_names, lower\_gate = Ir193di\_gates$lower\_gate, upper\_gate = Ir193di\_gates$upper\_gate)  
  
density\_plots

## Picking joint bandwidth of 0.0122



# tiff(fs::path(outFigPath, paste0("fig9\_Ir193\_gating.tiff")))  
# print(density\_plots)  
# dev.off()#density\_plots  
  
time\_signal\_plots <- time\_signal\_plot(data = clean\_up\_data, random\_events = random\_events\_for\_plotting, channel = "Ir193Di", plot\_title = file\_names, lower\_gate = Ir193di\_gates$lower\_gate, upper\_gate = Ir193di\_gates$upper\_gate)  
#time\_signal\_plots # to see all plots  
#time\_signal\_plots[1] # to see first plot  
  
  
  
plotSignal(plot\_list = time\_signal\_plots)



## TableGrob (2 x 3) "arrange": 6 grobs  
## z cells name grob  
## 1 1 (1-1,1-1) arrange gtable[layout]  
## 2 2 (1-1,2-2) arrange gtable[layout]  
## 3 3 (1-1,3-3) arrange gtable[layout]  
## 4 4 (2-2,1-1) arrange gtable[layout]  
## 5 5 (2-2,2-2) arrange gtable[layout]  
## 6 6 (2-2,3-3) arrange gtable[layout]

# tiff(fs::path(outFigPath, paste0("Signal\_fig9\_Ir193\_gating.tiff")))  
# plotSignal(plot\_list = time\_signal\_plots)  
# dev.off()

events\_to\_keep\_after\_Ir193Di\_gating <- events\_to\_keep(data = clean\_up\_data, channel = "Ir193Di", lower\_gate = Ir193di\_gates$lower\_gate, upper\_gate = Ir193di\_gates$upper\_gate)  
 #percent\_to\_keep\_this\_gating(kept\_events = events\_to\_keep\_after\_Ir193Di\_gating, file\_names = file\_names)  
  
  
 fcs\_data <- update\_data\_based\_on\_events\_to\_keep(data = fcs\_data, kept\_events = events\_to\_keep\_after\_Ir193Di\_gating)  
 clean\_up\_data <- update\_data\_based\_on\_events\_to\_keep(data = clean\_up\_data, kept\_events = events\_to\_keep\_after\_Ir193Di\_gating)  
 number\_of\_events\_after\_Ir193Di\_gating <- number\_of\_events(data = fcs\_data, file\_names = file\_names)  
 percent\_lost\_from\_full\_dataset[as.character(file\_names),"Ir193Di"] <- number\_of\_events\_after\_Ir193Di\_gating/number\_of\_events\_raw\_data \* 100 #percent remaining from total  
 percent\_lost\_each\_gating[as.character(file\_names),"Ir193Di"] <- number\_of\_events\_after\_Ir193Di\_gating/number\_of\_events\_before\_Ir193Di\_gating \* 100 #percent remaining from cis gating

The percent of events lost for this gating is

percent\_lost\_each\_gating[as.character(file\_names), "Ir193Di"]

## [1] 95.86377 95.71781 96.71938 97.03659 94.01092 97.31072

The last ting to do is to save the clean files.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 #save fcs\_data ----  
 #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
 flowCore::write.flowSet(fcs\_data, outdir = outDataPath, filename = as.character(file\_names))

## C:/CyToF data/immun\_aga/panel 1 cleaned data

and the amount of the files lost during gating.

write.csv2(percent\_lost\_each\_gating, fs::path(out\_result, "percent\_kept\_each\_gatingnr2.csv"))  
 write.csv2(percent\_lost\_from\_full\_dataset, fs::path(out\_result, "percent\_kept\_from\_full\_datasetnr2.csv"))