Wellcome Sanger Institute Cytometry Core Facility: Introduction to R for Flow Cytometry

Contents

R is case sensitive and the red indented text in this handout is code. You need have R installed and I use RStudio to run my scripts.

Working Directory

This is where R looks for data and stores the output. To see where it is type into the console:

```
getwd()
```

To change the working directory type:

```
setwd("c:/myfiles/")
```

Note that forward slashes are used instead of backslashes. Backslashes mean something in R, they are 'escape characters'. For example, \n = new line, \t = tab

Console Window

Think of it as a calculator. Type commands and receive answers. Finish each command by pressing enter. Try:

```
1+1

6*8

5+6*8

4/(4-2)
```

Pay attention to the order of calculations. You can use the up arrow to bring back previous inputs.
Brackets – orders (power) – division – multiplication – addition - subtraction

Objects

Objects are things that mean other things ;) Type:

```
myname <- "Chris"

myname
```

Strings, i.e. text needs to be enclosed in double quotes. Now try:

```
a <- 1

b <- 2

a+b

a+b+myname
```

myname is not a number! To remove an object type:

```
rm(myname)
```

With objects you can be clever and make your life easier, for example:

```
radius <- 10

height <- 15

vol_cyl <- pi*radius^2*height

vol_cyl
```

Or even better you could make a function to do the calculation by just typing vol_cyl:

```
vol_cyl <- function(radius, height){

  volume <- pi*(radius^2)*height

  return(volume)
```

function() tells R that you are creating a repeatable command and the contents of the brackets are what needs to be passed to the function in order for it to work.

Then typing either of these commands will give you the cylinder volume:

```
vol_cyl(radius, height)
vol_cyl(10, 15)
```

You now have the volume of a cylinder. You can change radius and height as you wish and still use the vol_cyl object to calculate the volume. pi is a built in constant.

Data types

Character: "words", "writing"
Numeric: 2, 15.5
Integer: 2L (the L tells R to store this as an integer, i.e. a whole number)
Logical: TRUE, FALSE
Complex: 1+4i (complex number)

```
class(radius)
```

Vectors

A collection of elements of a similar data type.

```
MyVector <- c(10,11,12,13,14,15)
MyVector
```

The c() allows you to input multiple values at ones. Another example:

```
EasyVector <- c(10:14)
EasyVector
```

You can examine and change a vector using:

```
MyVector[3]
length[MyVector]
EasyVector <- c(EasyVector, 15)
EasyVector
```

Try this and think about the result:

```
MyVector*EasyVector
```

Matrices

A matrix is a vector with dimensions.

```
MyMatrix <- matrix(1:50, nrow=10, ncol=5)
MyMatrix
```

Lists

An ordered collection of elements that can be different data types.

```
MyList <- c(name="Chris", number=a, matrix=MyMatrix)
MyList[[2]]
MyList[["number"]]
```

Data Frames

Are the basically tables. Each column can have a different data type.

```
x <- c(10,20,33,44)

y <- c("blue", "green", "purple", NA)

z <- c(FALSE,TRUE,TRUE,FALSE)

mydata <- data.frame(x,y,z)

mydata
```

head() - first 6 rows

tail() - last 6 rows

dim() - dimensions of data frame

nrow() - number of rows

ncol() - number of columns

str() - structure of data frame - name, type and preview of data in each column

names() or **colnames()** - both show the names attribute for a data frame

There are other data types, Google is your friend.

Scripts

A script is a reusable list of commands, think text file. Click File>New>R Script.
This will make an empty file where you can type in your commands. To run them either click Ctrl+Enter on a line, or highlight the section and press Run.

Stats

R contains a number of example data sets. Type data() to see them.
To load a dataset use data(Indometh), to see the data type Indometh, and to see a description use help(Indometh).
The dollar sign ($) allows you to select columns in data frames such as this. Try:

```
mean(Indometh$conc)
```

You can replace mean with mode, sd, max, min etc. For help try help(max).
Help() is your friend.

Plots

There are many ways to plot in R. This is just one way. Better ways include using ggplot2 and, for flow cytometry, flowViz or ggcyto. More on this later.

> hist(Indometh$conc)
>
> hist(Indometh$conc, col="red", breaks=10)
>
> boxplot(Indometh$conc)
>
> plot(Indometh$time, Indometh$conc)

To format your table try adding:

- **xlim, ylim**: range of the x-axis and the y-axis

- **xlab, ylab**: labels for the x-axis and the y-axis

- **main**: main title of the graph

- **pch**: symbol type. See ?points.

- **cex**: symbol size

- **col**: symbol colour by either colour name or index. See colors().

> plot(Indometh$time, Indometh$conc, xlab="Time", ylab="Concentration", main="Pharmacokinetics of Indomethacin", col="red", pch=13)

Export your plot using the export button in the plot window. Or by typing this and checking your working directory:

> dev.copy(png,"myplot.png",width=8,height=6,units="in",res=100)
>
> dev.off()

Always finish with dev.off() when exporting a plot by command line.

Programming

For Loops

'For loops' allow you to repeat sections of codes for a predetermined set of times. The code to be run is enclosed in curly brackets{}, for example:

```
for(i in 1:10){print(i)}
```

While Loops

'While loops' repeat a section of code as long as a condition is met. For example:

```
countdown <- 10

while(countdown>0){

print(countdown)

countdown <- countdown-1}
```

If

If statements respond to certain values.

```
countdown <- 10

while(countdown>0){

  print(countdown)

  countdown <- countdown-1

  if (countdown == 0){ print("Takeoff")}

}
```

Pay attention to the curly brackets!

Apply

The apply family of functions enable you to iterate through and to manipulate matrices, lists, arrays, and data frames. We care about the apply functions because they are used to handle groups of flow cytometry files in R. There are a few different ones: lapply(), sapply(), maaply() and more. Google is your friend, again. We will use fsApply later.

Packages

R has many packages which are pre written sets of R functions that can be used on your data. For example ggplot2 has sophisticated plotting tools and flowCore allows you to analyse flow cytometry data. They are written by the R community and can be found in two places. The general purpose R packages are on CRAN and biological R packages are on BioConductor. To load them:

```
install.packages("ggplot2")
```

Or for Bioconductor:

```
install.packages("BiocManager")

BiocManager::install("flowCore")
```

You may be asked which packages to update, you normally want to type 'a' and enter.

You must then apply the downloaded packages to your R installation using:

```
library(ggplot2)
```

Flow Cytometry

Flow cytometry data is stored in .fcs files which are spreadsheets with headers. The headers contain the keywords and parameter information. The spreadsheet contains the fluorescent information from each event, i.e. cell.

We need to use some community built packages to interact with flow cytometry data. Let's start with flowCore and flowViz, the two "basic" flow cytometry packages.

```
BiocManager::install("flowCore")

BiocManager::install("flowViz")

BiocManager::install("flowAI")

library(flowCore)

library(flowViz)

library(flowAI)
```

Load a single flow cytometry file into a "flowFrame" using:

```
myfile <- "C:/FCSfiles/8peak400v.fcs"

fcsfile <- read.FCS(myfile)
```

Explore the file using:

```
fcsfile

exprs(fcsfile)[1:10,]

summary(fcsfile)

str(keyword(fcsfile))

summary(fcsfile[,7:24])
```

Pay attention to the type of bracket. () is for functions of a package and [] is for a subset of some data.

One of the advantages of using R to analyse your data is the ability to "clean" your data in an automated fashion. Cleaning the data involves removing events that are not representative of the sample because they are low quality. Several packages will do this for you: flowClean, flowAI, and flowCut. Here we will use flowAI, which looks at expression over time to remove events that are outside a set variance, checks the sample acquisition rate to remove events that were acquired when the instrument was not running at its average rate, and it removes events that fall outside the dynamic range of the instrument. Check the documentation for more info.

```
fcsfile_flowAI <- flow_auto_qc(fcsfile)

nrow(fcsfile)

nrow(fcsfile_flowAI)
```

Apply the compensation matrix saved in the file from the acquisition using the spillover() function.

```
fcsfile_comp <-compensate(fcsfile_flowAI, spillover(fcsfile)[[1]])
```

This function searches the keywords for possible spillover matrices. In our case we want the first one so we extract it using [[1]]. If you are not sure run spillover(fcsfile) and look at your options.

Transform the data for plotting. The data is almost always exported as raw and linear.

```
chnls <- colnames(fcsfile_comp[,7:24])

chnls

trans <- estimateLogicle(fcsfile_comp, chnls)

fcsfile_trans <- transform(fcsfile_comp, trans)
```

Plotting

```
plot(fcsfile_trans, c("FSC-A", "SSC-A"))

plot(fcsfile_trans, c("610/20 (561)-A", "450/50 (355)-A"))

plot(fcsfile_trans[,7:24])

plot(fcsfile_trans)

plot(fcsfile_trans, "610/20 (561)-A")

plot(fcsfile_trans, "610/20 (561)-A", breaks=1024)
```

Group of files

Groups of files are known as flowSets in flowCore. You list the files, load them into flowCore and either analyse them individually or as a group.

```
files <- list.files(path="C:/FCSfiles/", pattern=".fcs$")

fs <- read.flowSet(files, path="C:/FCSfiles/")
```

The 'path' parameter is only necessary if you are loading files outside your working directory.

You can explore the data the same way as you would a single file. Pay attention to the double [] as you need both to look at a file in the flowSet. For example:

```
fs

summary(fs_trans[[1]])

head(fsApply(fs_trans, nrow))
```

The fsApply is a tool in flowCore to allow you to iterate through a flowSet, think of it like a for loop.

Clean the data using flowAI.

```
fs_flowAI <- flow_auto_qc(fs)

head(fsApply(fs, nrow))

head(fsApply(fs_flowAI, nrow))
```

Apply the compensation matrix saved in the files from acquisition:

```
comp <-fsApply(flowAI,function(x)spillover(x)[[1]], simplify=FALSE)

fs_comp <-compensate(fs, comp)
```

Transform and plot the flowSet.

```
tf <- estimateLogicle(fs_comp[[1]], channels = colnames(fs[[1]][,7:24]))

fs_trans <- transform(fs_comp, tf)

plot(fs[[1]], c("610/20 (561)-A", "450/50 (355)-A"))

plot(fs_trans[[1]], c("610/20 (561)-A", "450/50 (355)-A"))
```

To plot the flowSet it is easier to use another visualisation package called ggcyto. ggcyto uses the structure of ggplot2 and applies it to flow cytometry files. Look up ggplot2 for more info, especially on how to change the look of your plot.

In theory you should be able to use fsApply to plot using flowViz, but I don't know how ☹

```
BiocManager::install("ggcyto")

library(ggcyto)
```

```
autoplot(fs_trans[[1]], x="610/20 (561)-A", y="450/50 (355)-A", bins = 256)

autoplot(fs_trans, x="610/20 (561)-A", y="450/50 (355)-A", bins = 256)


p <- ggcyto(fs_trans, aes(x = "610/20 (561)-A", y =  "450/50 (355)-A"))

p <- p + geom_hex(bins = 128)

p
```

You use the + operator to add elements and style to your plot.

Gating

You can gate individual files using a flowFrame and whole sets using flowSets. We will be gating a flowSet. To do this we will load the flowWorkspace package, which can also load gating strategies from other analysis programs, though we won't do that today. Gates are known as filters in R.

```
BiocManager::install("flowWorkspace")
library("flowWorkspace")
```

The GatingSet is the holding area for all the gates.

```
gs <- GatingSet(fs_trans)
```

We can make a number of gates, just like on the acquisition software:

rectangleGate : A square

polygonGate : A not so square

polytopeGate : A convex hull of a set of points, possibly in more than two dimensions

ellipsoidGate : Think squishy circle

norm2Filter : Automatically try to find the region that most resembles a bivariate Normal distribution.

kmeansFilter : Automatically try to find a region based on a one dimensional k-means.

```
rg1 <- rectangleGate("FSC-H"=c(100000, Inf), filterId="NonDebris")
add(gs, rg1, parent = "root")
getNodes(gs)
recompute(gs)
autoplot(gs,x = 'FSC-H', y = 'SSC-H', "NonDebris", bins = 256)
```

The sequence here is important once the GatingSet has been created. Define the gate using coordinates, add it to the GatingSet, check it is there, and compute the stats. You then plot to check it all looks good.

If something goes wrong with your gating (filters) try:

```
Rm('singlets', gs)
recompute(gs)
```

We have made a "none debris" gate, now let's repeat for the singlets.

```
rg2 <- rectangleGate("FSC-H"=c(100000, 150000),"FSC-W"=c(50000, 75000))
add(gs, rg2, parent = "NonDebris", name = "singlets")
getNodes(gs)
recompute(gs)
autoplot(gs,x = 'FSC-H', y = 'FSC-W', "singlets", bins = 256)
```

Now lets gate the fluorescent parameters:

```
autoplot(fs_trans[[2]], '530/30 (488)-A')+geom_histogram(binwidth = 0.01)
```

STOP!! This is not going to be pleasant. We have been manually gating this, which is time consuming and unreliable across big flowSets. We need to automate things, but first a quick look at what we can do with filtered (gated) flowSets.

```
plot(gs)
getStats(gs)
```

```
getStats(gs, "singlets", "percent")
autoplot(gs[[1]])
```

To create a subset of the fcs files, i.e. only look at the gated cells, you need to use the getData function.

```
fs_singlets <- getData(gs, "/NonDebris/singlets")
fsApply(fs_singlets, each_col, median)
```

Autogating

There are many autogating tools, some included in flowCore like kmeansFilter and others are in other packages such as flowDensity, flowMeans, openCyto. Which one is best for you depends upon what your needs are. I will briefly use openCyto here to repeat the last two gates in an automated fashion. The only difference is in using fsApply and a function using openCyto to apply the autogating. Here we use flowClust.2d and singletGate.

Let's start again, first clear everything using the brush button in the environment tab   . Then:

```
library("flowCore")
library("flowWorkspace")
library("openCyto")

files <- list.files(path="C:/FCSfiles/", pattern=".fcs$")
fs <- read.flowSet(files, path="C:/FCSfiles/")
tf <- estimateLogicle(fs[[1]], channels = colnames(fs[[1]][,7:24]))
fs_trans <- transform(fs, tf)

gs <- GatingSet(fs_trans)

thisData <- getData(gs)
nonDebris_gate <- fsApply(thisData, function(fr) openCyto:::.flowClust.2d(fr, channels = c("FSC-H","SSC-H")))
add(gs, nonDebris_gate, parent = "root", name = "nonDebris")
recompute(gs)
autoplot(gs,x = 'FSC-H', y = 'SSC-H', "nonDebris", bins = 256)

thisData <- getData(gs, "nonDebris") #get parent data
singlet_gate <- fsApply(thisData, function(fr) openCyto:::.singletGate(fr, channels =c("FSC-H", "FSC-W")))
add(gs, singlet_gate, parent = "nonDebris", name = "singlets")
recompute(gs)
autoplot(gs,x = 'FSC-H', y = 'FSC-W', "singlets", bins = 256)
```

Let's escape autoplot so we can have more control of the plotting. We need to stop the statistics overlapping the events. Note that the nudge_y parameter has the same scale as the plot.

```
p1 <- ggcyto(gs, aes(x = 'FSC-H', y = 'SSC-H')) + geom_hex(bins = 128) + geom_gate("nonDebris") +
geom_stats(nudge_y=50000)
p1
p2 <- ggcyto(gs, aes(x = 'FSC-H', y = 'FSC-W')) + geom_hex(bins = 128) + geom_gate("singlets") +
geom_stats()
p2
```

I am going to stop here due to time constraints. I have purposely not gated fluorescent parameters as the 8 peak beads are a pain to autogate, though the kmeansFilter does work very well, but is very difficult to plot. You would be better off doing an overlay of the clusters. To continue gating just keep repeating the elements above.

Clustering

There are many clustering algorithms. For flow cytometry you will most likely be looking at tSNE or viSNE, flowSOM and phenograph.

https://github.com/jkrijthe/Rtsne

https://github.com/SofieVG/FlowSOM

https://github.com/JinmiaoChenLab/Rphenograph

Let's look at tSNE:

```
install.packages("Rtsne")
library(Rtsne)
set.seed(42)
thisData <- getData(gs, "singlets")
tsne_out <- Rtsne(as.matrix(exprs(thisData[[2]][,7:24])),perplexity=50, ) # Run TSNE
plot(tsne_out$Y, col=exprs(thisData[[2]][,7]))
```

set.seed is to increase reproducibility,

Exporting your data

You want the stats, you desire the pictures, and you need the raw data.

Use exprs() to extract fluorescence data from flowCore to use in other packages.

use write.csv() to save spreadsheets.

Use the package gridExtra to make nice neat lots.

Use ggsave() to export plots as images or use the save button on the plots window.

```
library("gridExtra")
grid.arrange(as.ggplot(p1), as.ggplot(p2), nrow = 2)
g <- arrangeGrob(as.ggplot(p1), as.ggplot(p2), nrow = 2)
ggsave(file="plots.png", g)
write.csv(getPopStats(gs), "stats.csv")
write.csv(exprs(thisData[[1]]), "stats2.csv")
```

Help

Use your favourite internet search engine!

Use the R documentation

Use the package documentation.

Use the ? and ?? commands in R

Use http://stackoverflow.com/

Practice

Analyse your data

Code

```r
library(flowCore)
library(flowViz)
library(flowWorkspace)
library(openCyto)
library(ggcyto)

setwd("C:/yourDATA/")

files <- list.files(path="C:/yourDATA/", pattern=".fcs$")
fs <- read.flowSet(files, path="C:/yourDATA/") #danger point!
tf <- estimateLogicle(fs[[1]], channels = colnames(fs[[1]][,8:13]))
fs_trans <- transform(fs, tf)
gs <- GatingSet(fs_trans)

#gate the main population of events
thisData <- getData(gs)
nonDebris_gate <- fsApply(thisData, function(fr) openCyto:::.mindensity(fr, channels = "FSC-A"))
add(gs, nonDebris_gate, parent = "root", name = "nonDebris")
recompute(gs)

#gate the singlets
thisData <- getData(gs, "nonDebris") #get parent data
singlet_gate <- fsApply(thisData, function(fr) openCyto:::.singletGate(fr, channels =c("FSC-A",
"FSC-W")))
add(gs, singlet_gate, parent = "nonDebris", name = "singlets")
recompute(gs)

#gate the CD45+
thisData <- getData(gs, "singlets") #get parent data
CD45_gate <- fsApply(thisData, function(fr) openCyto:::.mindensity(fr, channels ="APC-A"))
add(gs, CD45_gate, parent = "singlets", name = "CD45")
recompute(gs)

#gate the CD3 then the CD4/8
thisData <- getData(gs, "CD45") #get parent data
CD3_gate <- fsApply(thisData, function(fr) openCyto:::.mindensity(fr, channels ="FITC-A"))
add(gs, CD3_gate, parent = "CD45", name = "CD3")
```

```
recompute(gs)


thisData <- getData(gs, "CD3") #get parent data

Tcell_quad <- fsApply(thisData, function(fr) openCyto:::quadGate.tmix(fr, channels = c("PerCP-
Cy5-5-A", "APC-Cy7-A"), K = 3))

add(gs, Tcell_quad, parent = "CD3", name = c("CD4+","CD8+","--","++"))

recompute(gs)


#export the data

f2<- autoplot(gs,x = 'FSC-A', y = 'SSC-A', "nonDebris", bins = 256)

f3<- autoplot(gs,x = 'FSC-A', y = 'FSC-W', "singlets", bins = 256)

f4<- autoplot(gs, x = 'APC-A', y = 'FSC-A', "CD45", bins = 256)

f5<- autoplot(gs, x = 'FITC-A', y = 'FSC-A', "CD3", bins = 256)

f6<- autoplot(gs, x = "PerCP-Cy5-5-A", y = "APC-Cy7-A", c("CD4+","CD8+","--","++"), bins = 256)


g <- grid.arrange(as.ggplot(f2), as.ggplot(f3),as.ggplot(f4), as.ggplot(f5), as.ggplot(f6),nrow = 2)

ggsave(file="plots.png", g)

write.csv(getPopStats(gs), "stats.csv")
```

Results
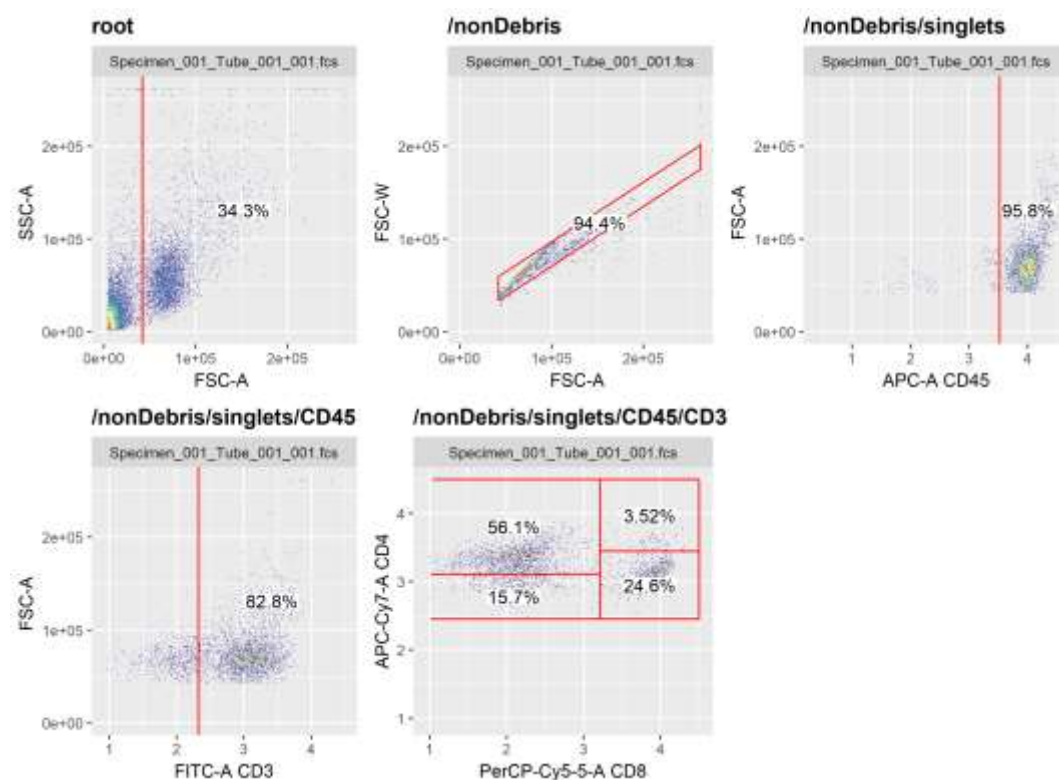
	name	Population	Parent	Count	ParentCount
1	Specimen_001_Tube_001_001.fcs	nonDebris	root	3829	11172
2	Specimen_001_Tube_001_001.fcs	singlets	nonDebris	3614	3829
3	Specimen_001_Tube_001_001.fcs	CD45	singlets	3463	3614
4	Specimen_001_Tube_001_001.fcs	CD3	CD45	2869	3463
5	Specimen_001_Tube_001_001.fcs	CD4+	CD3	1610	2869
6	Specimen_001_Tube_001_001.fcs	CD8+	CD3	101	2869
7	Specimen_001_Tube_001_001.fcs	--	CD3	707	2869
8	Specimen_001_Tube_001_001.fcs	++	CD3	451	2869