~~~
---
title: "Gating in R"
author: "R. Burke Squires, Radina Droumeva"
date: "7/11/2017"
output: ioslides_presentation
---

LICENSE


```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
~~~

# Clear current workspace, close plots, load libraries

```{r}
rm(list=ls())
graphics.off()
library(flowCore)
library(flowDensity)
library(GEOmap)
```

# Tell R which directory we are working in and load the
transformed flowSet object:

```{r}
setwd('~/Desktop/auto_flow_cyto_analysis/data/')
load('trans.fs.RData')
trans.fs
```

# Let's gate the CD3+ live cells!

```{r}
cd3 <- "R780-A"
dump <- "V450-A"
```

# First create a pooled flowFrame:

```{r}
source("./code/supportCode/support_functions.R")
```

~~~
pooled.frame <- getGlobalFrame(trans.fs)
{plotDens(pooled.frame, c(cd3, dump))
  abline(v = 1, lwd=2, col = "blue")
  abline(h = 1.5, lwd=2, col="blue")}
```

# By looking at this, it looks like CD3 = 1 and Dump Channel =
1.5 are good gates

```{r}
# abline(v = 1, lwd=2, col = "blue")
# abline(h = 1.5, lwd=2, col="blue")
```

# Let's verify by plotting whole flowSet:

```{r}
par(mfrow = c(5, 4), mar = c(3, 3, 2, 1), mgp = c(2, 1, 0))
for (i in 1:length(trans.fs)){
  plotDens(trans.fs[[i]], c(cd3, dump))
  abline(v = 1, lwd=2, col = "blue")
  abline(h = 1.5, lwd=2, col="blue")
}
```

# Looks pretty good. Now let's isolate the viable cells into a
new flowSet

```{r}
viable.fs <- trans.fs
```

## Use fsApply

This time, instead of a for loop, let's use 'fsApply'. It
applies a single function to each flowFrame object. So, we must
first write our very own function! To do this, we must imagine
we are only working with a single flowFrame and write the
algorithm to isolate the viable cells for it:

```{r}
# Code for single flowFrame:
f <- trans.fs[[1]]
cd3pos.indices <- which(exprs(f)[, cd3] > 1)
dumpneg.indices <- which(exprs(f)[, dump] < 1.5)
~~~

~~~~
combined <- intersect(cd3pos.indices, dumpneg.indices)
viable.f <- f[combined]
```

# Is this right? Let's check

```{r}
graphics.off()
plot(exprs(f)[, c(cd3, dump)], pch=".")
points(exprs(viable.f)[, c(cd3, dump)], pch=".", col =
"green4")
```

# See ?chull for an idea of how to plot a gate using lines
instead of just a different colour: run the example in the
helpf file!

```{r}
plotDens(f, c(cd3, dump))
X <- exprs(viable.f)[, c(cd3, dump)]
gate.pts <- chull(X)
gate.pts <- c(gate.pts, gate.pts[1]) # Connect the first and
last point to make gate closed.
lines(X[gate.pts, ], lwd=2, col="blue", lty="dashed")
```

## Functions

This is how you create your own function in R:

1) Give it a name of your choice, make sure it is uncommon
enough that it won't accidentally interfere with existing R
functions! I.e. don't create a function called "read.FCS" or
"plot" or "which"!

2) inside the 'function(*)' put a variable name which will be
only used within the function. I chose 'f', but you can call it
'x' and replace all the 'f's inside the function with 'x's.

```{r}
getViableFrame <- function(f){
  # Manipulate the input 'f':
  cd3pos.indices <- which(exprs(f)[, cd3] > 1)
  dumpneg.indices <- which(exprs(f)[, dump] < 1.5)
  combined <- intersect(cd3pos.indices, dumpneg.indices)
~~~~

```
  viable.f <- f[combined]
  # Finally, return the desired altered version of the input:
  return (viable.f)
}
```

## Execute the function

Execute the function definition to enable it for use. Now try it:

```{r}
viable.f <- getViableFrame(f)
# Is this right? Let's check (again):
graphics.off()
plot(exprs(f)[, c(cd3, dump)], pch=".")
points(exprs(viable.f)[, c(cd3, dump)], pch=".", col =
"green4")
```

# Now we can use fsApply:

```{r}
viable.fs <- fsApply(trans.fs, getViableFrame)
```

# Let's look at the proportions of live cells:

```{r}
live.counts <- fsApply(viable.fs, nrow)/fsApply(trans.fs, nrow)
plot(density(live.counts), main = "Proportion CD3+ viable
cells")
```

## Twin Peaks?

It kind of looks like two peaks in the density. Can we do anything interesting just with this information?

Let's extract the clinical information we have first.

For this workshop, I have selected 20 FCS files for patients which have an event reported -- either death or progression to AIDS. We have the number of days before the event occured. Infact, that information is stored inside the FCS keywords!

~~~~
```{r}
survival <- fsApply(viable.fs, function(x) x@description$`CD
Survival time from seroconversion`)
# Let's convert this to numbers so we can work with them:
survival <- as.numeric(survival)
survival
plot(survival, live.counts, pch = 19)
```

## Try kmeans:

```{r}
km <- kmeans(live.counts, 2)
plot(survival, live.counts, pch = 19, col = km$cluster, main =
"K-means misclassifies 3 samples from each group.")
# Too few samples, but it looks like more of the low-survival
patients have lower CD3+live proportions also.
# For a really good explanation of k-means and hierarchical
clustering, see Andrew Moore's slides:
http://www.autonlab.org/tutorials/kmeans11.pdf
```

## Automated Gating

So far we have preprocessed our data and have now isolated the
live CD3+ cells.

Let's now consider automated gating for the remainder of the
analysis.

Consider CD4 first:

```{r}
cd4 <- "V655-A"
pooled.frame <- getGlobalFrame(viable.fs)
par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), mgp=c(2, 1, 0))
{plotDens(pooled.frame, c(cd3, cd4))
abline(h = cd4.gate, lwd=2, col="blue")}
```

## Try flowDensity -- the function 'deGate'

```{r}
cd4.gate <- deGate(pooled.frame, cd4)
~~~~

~~~
```

## Looks good, how did it do it?

```{r}
deGate(pooled.frame, cd4, graphs=TRUE)wssss
```

## Try for every channel (except scatter, CD3 and the dump
channel):

```{r}
ki67 <- "B515-A"
cd8 <- "V800-A"
cd127 <- "G560-A"
```

##  This time we will record the gates for all channels into a
vector

```{r}
store.gates <- rep(-Inf, 4)
names(store.gates) <- c(cd4, cd8, cd127, ki67)
store.gates
par(mfrow = c(2, 2))
for (chan in c(cd4, cd8, cd127, ki67)){
  plotDens(pooled.frame, c(cd3, chan))
  store.gates[chan] <- deGate(pooled.frame, chan)
  abline(h = store.gates[chan])
}
store.gates
```

# All looks good except for CD127. Let's work on it:

```{r}
par(mfrow = c(5, 4), mar = c(3, 3, 1,1), mgp=c(2, 1, 0))
for (i in 1:length(viable.fs)){
  plotDens(viable.fs[[i]], c("SSC-A", cd127))
}
```



Still not obvious. This is where a control would be necessary!
Let's ignore CD127 from now on.
~~~

Let's at least make sure CD4, CD8 and KI67 work for all samples:

~~~
```{r}
par(mfrow=c(5,4), mar = c(3, 2, 2, 1), mgp=c(2, 1, 0))

for (i in 1:20){
  plotDens(viable.fs[[i]], c(cd4, cd8))
  abline(v = store.gates[cd4], lwd=2, col="blue")
  abline(h = store.gates[cd8], lwd=2, col="blue")
}

par(mfrow=c(5,4), mar = c(3, 2, 2, 1), mgp=c(2, 1, 0))

for (i in 1:20){
  plotDens(viable.fs[[i]], c(cd8, ki67))
  abline(v = store.gates[cd8], lwd=2, col="blue")
  abline(h = store.gates[ki67], lwd=2, col="blue")
}
```
~~~

Looks pretty good. Now what? flowType!

If you don't have flowType, i.e. if library(flowType) doesn't work):

(Ideally you can run this during lunch)

```
#source("http://bioconductor.org/biocLite.R")
#biocLite('rrcov')
#biocLite('codetools')
#biocLite('foreach')
#biocLite('flowMerge')
#biocLite('flowType')
#biocLite('RchyOptimyx')
```

~~~
```{r}
library(flowType)
library(RchyOptimyx)
```

```{r}
~~~

~~~
# For convinience, rename the channels of the flowSet to more
phenotype-friendly names:
colnames(viable.fs) <- c("FSC-A", "FSC-H", "SSC-A", "KI67",
"CD3", "CD8", "CD4", "Dump", "CD127")
```

# Now we run flowType on a single flowFrame to see how it
works. It is hard!
# Frame is your flowFrame object.
# PropMarkers are the indices or markers which you want to
involve in the analysis. For us, CD4 is the 7th marker in the
list above, CD8 is the 6th, and KI67 is the 4th. How you order
them does not matter as long as you are consistent.
# Methods are the gate threshold values in the same order as
above.
# MarkerNames is the full vector of all channel names.
```{r}
ft1 <- flowType(Frame = viable.fs[[1]],
                PropMarkers=c(7, 6, 4),
                Methods='kmeans',
                MarkerNames=colnames(viable.fs))
```
# Examine ft1 -- see what's in there.
# In the console below type f1@ and then press your tab key to
see available components to explore!


# Next, use fsApply to compute all phenotypes for thew hole
flow set.
# Notice that we have our own function defined within the
fsApply call -- you can do this if your function is so short,
that you don't need to define it separately. Also notice the
'/nrow(x)' part -- this is so that instead of cell counts we
get cell proportions. We cannot use cell counts because the
total number of starting cells is different for the different
samples.
```{r}
ft <- fsApply(viable.fs,
              function(x) flowType(x,
                                   PropMarkers=c(7, 6, 4),
                                   Methods='kmeans',

MarkerNames=colnames(viable.fs))@CellFreqs/nrow(x))
rownames(ft) <- sampleNames(viable.fs)
```
~~~

~~~
# We want to identify phenotypes which separate our data into
two groups based on survival times. We can calculate some p-
values to use as a guage on the phenotypes' importance.
# First, let's remove samples with very low live CD3+ counts
(say < 1000 cells):
```{r}
remove.low <- which(as.numeric(fsApply(viable.fs, nrow)) <
1000)
ft <- ft[-remove.low, ]
```

# Keep track of survival time before removing low viable count
samples just in case we need it later
```{r}
full.survival.data <- survival
survival <- survival[-remove.low]
```

# Now identify the patients with survival less than 1000 days
and over (from an earlier plot this looks like the dividing
number!)
```{r}
group1 <- which(survival < 1000)
group2 <- which(survival > 1000)
```

# Calculate the p-values. Here is the p-value for a single
phenotype:
```{r}
one.pval <- t.test(ft[group1, "CD4-CD8+"], ft[group2, "CD4-
CD8+"])$p.value
```

# Use a for loop to calculate the p-values by looping over all
phenotypes.
# Initialize a vector of p-values to be all 1:
```{r}
pvals <- rep(1, ncol(ft)) # ncol is the number of phenotypes!
for (i in 1:ncol(ft)){
  if (sd(ft[, i]) == 0) {# if no variation in the phenotype
measurement, the p-value will be undefined.
    pvals[i] <- 1
  } else {
    pvals[i] <- t.test(ft[group1, i], ft[group2, i])$p.value
~~~

~~~
  }
}
names(pvals) <- colnames(ft)
pvals
```

# Now we can use the p-values as a way to score each
phenotype's importance. However, we want to find the phenotype
with the fewest number of markers (most robust, efficient,
cheap to make into a panel) without losing much of the ability
to separate the patients with low survival time from those with
high survival time. Instead of 'pvals' though, let's use -log10
(pvals). Typically the lower the p-value is, the better the
phenotype is. By taking the -log10 of the p-value as the score,
we can now say the higher the score -- the better the
phenotype.

# First, RchyOptimyx wants us to provide all possible phenotype
combinations in terms of "+" or "-" combinations of markers.
Here we use '0' for a negative expression of a marker, '1' if
the marker is neutral (not involved in the phenotype at all),
and '2' for a positive expression of the marker. 'Signs'
contains this required variable. (very computer science-y, but
we are working on an easier version of flowType and RchyOptimyx
right now!)
```{r}
library(sfsmisc)
Signs <- t(digitsBase(1:(3^3-1), 3, ndigits=3))
rownames(Signs) <- colnames(ft)
colnames(Signs) <- c("CD4", "CD8", "KI67")
```

# Now to run RchyOptimyx. Here are a couple of additional
required parameters:
# startPhenotype: we don't necessarily want RchyOptimyx to
check ALL Possible phenotypes. We can specify one with a low p-
value that we want it to reach by combining its constituent
markers and their espression.
# For example, "012" means CD4-CD8(neutral)KI67+ == CD4-KI67+.
# To get a broader view, we should not simply pick the one with
the lowest value. We can generate a few RchyOptimyx trees and
then merge them into one plot:
```{r}
rch1 <- RchyOptimyx(Signs, -log10(pvals), startPhenotype =
"012", trimPaths=FALSE, pathCount=6)
~~~

~~~~
rch2 <- RchyOptimyx(Signs, -log10(pvals), "002",
trimPaths=FALSE, pathCount=6)
rch3 <- RchyOptimyx(Signs, -log10(pvals), "202",
trimPaths=FALSE, pathCount=6)
merged <- merge(rch1, merge(rch2, rch3)) # Can only merge two
at a time!
```

# Instead of plotting in RStudio, let's write this to a .pdf
file!
# Specify file name:
```{r}
pdf('/home//rguru/Documents/Workshop/rchy.pdf')
# Do your plotting:
plot(merged, phenotypeScores=-log10(pvals))
# Tell R you are done plotting and it is safe to write the
information to the file.
dev.off()
```

# Let's also save the flowType results and p-values:
```{r}
results <- rbind(ft, pvals)
rownames(results)[nrow(results)] <- "P-values (Uncorrected)"
write.csv(results,
file="/home/rguru/Documents/Workshop/results.csv")
```

# Navigate to that folder to see the plot and .csv report!

# See that the 'best' phenotype is KI67+
```{r}
ft[, "KI67+"]*100
par(mfrow=c(1,1))
boxplot(ft[group1, "CD4-KI67+"]*100, ft[group2, "CD4-
KI67+"]*100, boxwex=0.2, labels=c("Group1", "Group2"))
```
# NOT great at all, but to be expected with such a small set of
samples and small set of channels!!

# Just CD3+ counts: (i.e. first remove dead cells, then
calculate CD3+ proportion of live cells.)
```{r}
only.live <- fsApply(trans.fs, function(x) x[which(exprs(x)[,
"V450-A"] < 1.5)])
~~~~

```
cd3counts <- fsApply(only.live, function(x)
length(which(exprs(x)[, "R780-A"] > 1)))/fsApply(only.live,
nrow)
t.test(cd3counts[which(full.survival.data > 1000)],
cd3counts[which(full.survival.data < 1000)])
boxplot(cd3counts[which(full.survival.data > 1000)],
cd3counts[which(full.survival.data < 1000)], boxwex=0.2)
```