Automated Gating of Flow Cytometry Data using the Bioconductor `openCyto` Framework

Nichole Monhait, MPH Candidate
Colorado School of Public Health, Colorado State University

2019-05-12

Contents

1 What's inside? 5

2 Getting Started 7

2.1 Required Packages and Installation 7

3 Working with your Manual Gating Scheme 9

3.1 Read in flowJo file 9

3.2 Parse FCS files 10

3.3 Visualize and Verify 10

4 Create .csv 13

4.1 .csv Gating Template Structure 13

4.2 Creating the Template 17

5 Automate Gating 19

5.1 Load .csv into R 19

5.2 Read in raw FCS files 19

5.3 Apply Gating 19

5.4 Plot Automated Gating 20

5.5 Population Statistics 21

6 Customization 23

6.1 Hiding unwanted nodes 23

6.2 Renaming nodes 24

6.3 Adjusting plots 24

7 Appendix 27

7.1 Function and R Object Definitions 27

Chapter 1

What's inside?

Flow cyometry is a method used to gain understanding of cell samples and populations by quantifying scattered and emitted fluorescent light. Signals are captured and analyzed through use of software programs. Flow cytometry analysis consists of gating, a method that dictates which cells will be further analyzed and which will not. Current methods for flow cytometry gating involve manually drawing gates. This process is both time consuming and costly, making automated gating procedures an appealing option. The `openCyto` package allows users to take manually gated data from flowJo, reproduce those gates in R, and eventually automate the gating process. The goal of this tutorial is to take the user through the process of automated gating analysis.

This tutorial will be useful to anyone who has done manual gating on a sample and wishes to automate the same procedure on additional samples in the future.

The example data used in this tutorial is from Colorado State University's Microbiology, Immunology, and Pathology Department. Alternatively, you can input your own data using the filetypes described in Chapter 3.

Chapter 2

Getting Started

Here is an overview of the process to automate flow cytometry data using R's `openCyto` and what you will need to successfully automate your own flow cytometry analysis. The general steps to accomplish this are as follows:

1. Read in a manually gated flowJo workspace in .wsp file format.
2. Parse raw FCS files from the read in workspace.
3. Visualize the manual gating template and resulting gates to verify gating scheme.
4. Create and read in a .csv gating template.
5. Automate gating.
6. Visualize automated gating template and gates to verify gating scheme.
7. Extract population statistics and relevant information.

This process is completed primarily with the `openCyto` package but calls upon other packages within the Bioconductor `openCyto` framework. Packages needed to complete this tutorial are listed at the end of this chapter. Descriptions of each function and R object used for this analysis are below.

2.1 Required Packages and Installation

2.1.1 Package descriptions

Below is a description of each package used in this analysis. Code to install and use these packages will follow. Package descriptions taken from Bioconductor and CRAN.

Package Name	
openCyto	This package is designe
flowWorkspace	This package allows you to import basic flowJo workspaces into BioConductor and replicate the gating f
flowCore	Pro
flowStats	Methods and functiona
flowClust	Robu
data.table	Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete

2.1.2 Installation

Install the following libraries into a new R script. As you will see below, this tutorial uses the development version of `openCyto`. It is important to use the development version of `openCyto` to remain up to date on any changes made by the developers of `openCyto`. Use the following to ensure the correct packages are installed. Installation will only need to be done once.

To install

```r
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()

BiocManager::install(c("openCyto", "flowWorkspace", "flowCore",
                       "flowStats", "flowClust"))

install.packages("data.table")
devtools::install_github("RGLab/openCyto", ref = "trunk")
```

RStudio may also prompt you to download XQuartz and XCode based on your computer type, so it may be a good idea to go ahead and also download both.

2.1.3 Load packages

Although installation only needs to be done once, packages will need to be reloaded each time you open an R session. At the beginning of each session, run the following code.

To load

```r
library(openCyto)
library(flowWorkspace)
library(data.table)
library(flowCore)
library(flowStats)
library(flowClust)
```

Chapter 3

Working with your Manual Gating Scheme

The first step in this process is to bring a pre-existing flowJo file into R in order to recreate the gating environment. The remainder of this chapter will detail the following:

1. Read in flowJo .wsp file
2. Parse FCS files
3. Visualize and verify manual gates

3.1 Read in flowJo file

Within flowJo, tranformation, compensation, and gating can be saved as either *.xml* or *.wsp* filetypes. This tutorial will only detail steps from a *.wsp* filetype saved from flowJo. Note that many other tutorials begin from a *.xml* filetype. Saving analysis within flowJo is detailed here. Your *.wsp* file will contain samples and groups to be added to the Workspace in R, all gates and analyses, and compensation matrices. Importantly, the *.wsp* will not save your FCS files. Rather, the path to your files will be saved and can be adjusted later within R.

Before you begin, be sure you have loaded the required packages outlined in the previous chapter.

Once all packages are loaded, save the *.wsp* file path as an R object called **wsfile**. Next, use `openWorkspace()` with your R object created in the prior step to open the *.wsp* file in R. Save this as an R object. Here, this was saved as **ws** and is of flowJoWorkspace class. Here is an example of saving and opening your *.wsp* filetype in R. Please ensure that **ws** is saved as a flowWorkspace object containing groups of samples before proceeding.

```
wsfile <- "/Users/monhait/Desktop/group1_v_group2.wsp"
```

```
ws <- openWorkspace(wsfile)
```

```
print(ws)
```

```
## FlowJo Workspace Version  20.0
## File location:  /Users/monhait/Desktop
## File name:  group1_v_group2.wsp
## Workspace is open.
```

```
##
## Groups in Workspace
##          Name Num.Samples
## 1 All Samples          10
## 2     Samples          10
```

3.2 Parse FCS files

The next step is to read in raw FCS files. FCS files contain data from the cytometer. Standards for FCS files are listed here.

Raw FCS files are read using the `parseWorkspace` function. This function will read the FCS files and transform, compensate, and gate according to parameters defined from the *.wsp* flowJo workspace, which is now saved as an R object of class flowWorkspace. The `parseWorkspace` call requires the object that results from running `openWorkspace`. Here, we named this object `ws`. The function `parseWorkspace()` also requires the name of the samples to read in. To list sample names, use the `getSampleGroups()` function on your flowWorkspace class object. Other options may be customized based on particular needs. A new R object named `gating_set` is then created and will be a GatingSet object. The `isNcdf = TRUE` call saves this output to disk rather that into memory because the files are large. Here is an example of parsing FCS files. As this function runs, you will see several messages appear as the FCS files are loaded and the manual gating scheme is replicated. After this, `attributes()` is used to examine the data.

```
gating_set <- parseWorkspace(ws, name = "Samples", path = "/Users/monhait/Desktop/data/group1_v_group2"
```

```
## windows version of flowJo workspace recognized.
## version X
```

```
attributes(gating_set)
```

3.3 Visualize and Verify

It is helpful to now visualize both the gating template and gates on a subset of the data in order to verify the gating scheme. This will ensure consistency between the flowJo workspace and the manual gates recreated in R. First, save a subset of the `gating_set` as follows. The following saves the first FCS file of **gating_set** as **gh**. Since each FCS file corresponds to an individual experiment, this saves the first experiment of the group.

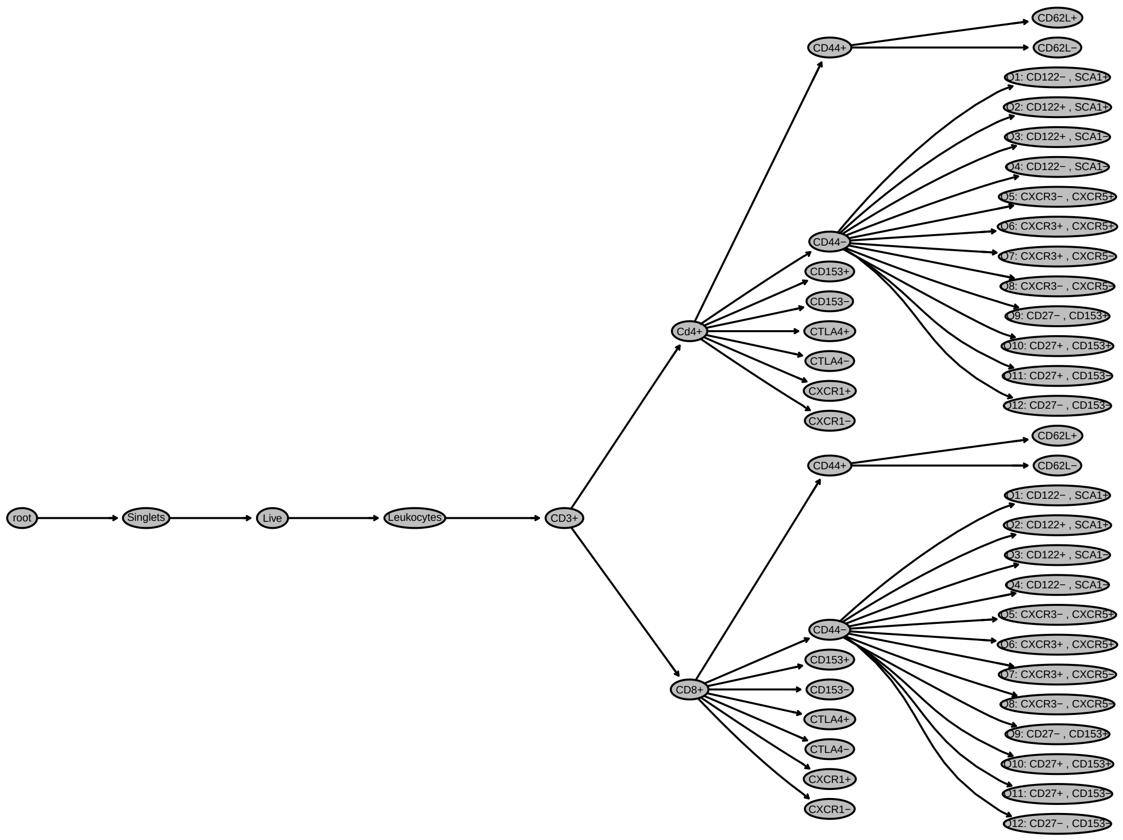
```
gh <- gating_set[[1]]
print(gh)
```

```
## Sample:  X_group1_1
## GatingHierarchy with  51  gates
```

3.3.1 plot()

The plot() function will visualize the current gating hierarchy when applied to an object of class GatingHierarchy. This can be done for the entire gating hierarchy or a specific population as seen below.

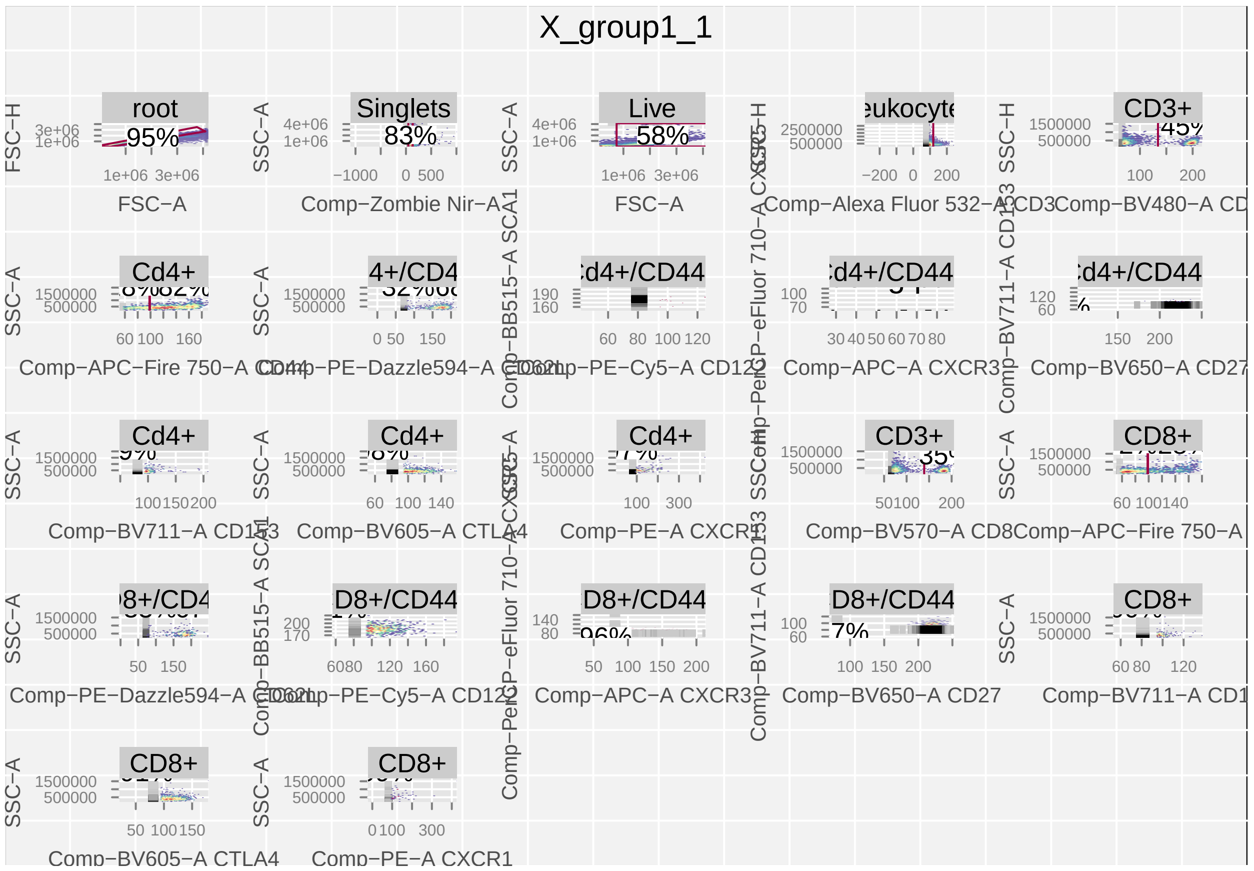
```
plot(gh)
```



3.3.2 plotGate()

The plotGate() function will gate the designated subset of your data according to parameters replicated from flowJo. This must also be called on an object of class GatingHierarchy.

```
flowWorkspace.par.set("plotGate", list(xlim = "data",
                                       ylim = "data"))
plotGate(gh)
```

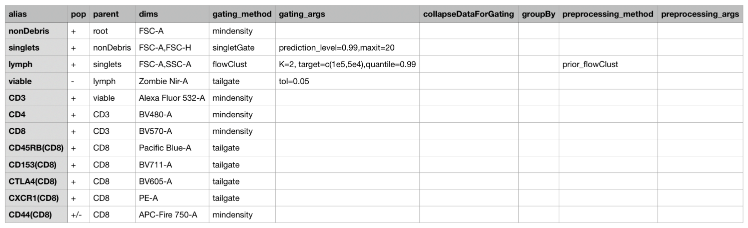


**Note the use of `flowWorkspace.par.set()` here. Chapter 5 of this tutorial will discuss customizations such as this one.

Chapter 4

Create .csv

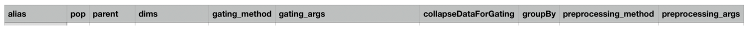
The creation of a .csv gating template is arguably the most important step to automating flow cytometry analysis. The .csv template that you create will tell `openCyto` how to gate your data. Included with this tutorial is a partial .csv template that can be used to gate the sample data. Generally, .csv templates will look like this:



alias	pop	parent	dims	gating_method	gating_args	collapseDataForGating	groupBy	preprocessing_method	preprocessing_args
nonDebris	+	root	FSC-A	mindensity					
singlets	+	nonDebris	FSC-A,FSC-H	singletGate	prediction_level=0.99,maxit=20				
lymph	+	singlets	FSC-A,SSC-A	flowClust	K=2, target=c(1e5,5e4),quantile=0.99			prior_flowClust	
viable	-	lymph	Zombie Nir-A	tailgate	tol=0.05				
CD3	+	viable	Alexa Fluor 532-A	mindensity					
CD4	+	CD3	BV480-A	mindensity					
CD8	+	CD3	BV570-A	mindensity					
CD45RB(CD8)	+	CD8	Pacific Blue-A	tailgate					
CD153(CD8)	+	CD8	BV711-A	tailgate					
CTLA4(CD8)	+	CD8	BV605-A	tailgate					
CXCR1(CD8)	+	CD8	PE-A	tailgate					
CD44(CD8)	+/-	CD8	APC-Fire 750-A	mindensity					

4.1 .csv Gating Template Structure

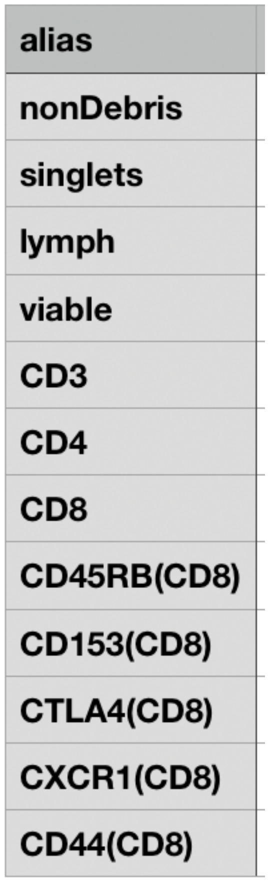
In the gating template, each row corresponds to a single cell population and the method used to gate that population. When read into R, the .csv will direct gating based on parameters listed in each row and column. The .csv must contain 10 predefined columns as seen here:



alias	pop	parent	dims	gating_method	gating_args	collapseDataForGating	groupBy	preprocessing_method	preprocessing_args

4.1.1 `alias`

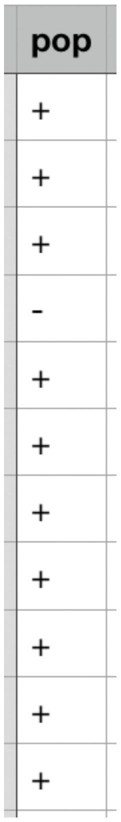
The first column must be titled `alias`. This is where you will put your cell population names. Remember, each row corresponds to a single cell population. Population names in the `alias` column must be unique.



alias
nonDebris
singlets
lymph
viable
CD3
CD4
CD8
CD45RB(CD8)
CD153(CD8)
CTLA4(CD8)
CXCR1(CD8)
CD44(CD8)

4.1.2 `pop`

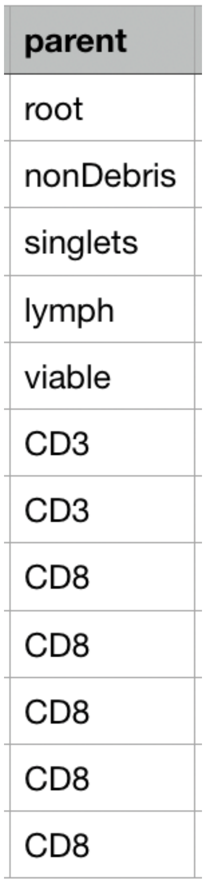
The second column must be titled `pop`. This column will contain a `+` or `-` to designate which subset or quadrant will be gated. A `+` will gate the positive subset while a `-` will gate the negative. This column can only contain strings of `+` and `-`, so do not use any characters as separators for quadrant gates.



pop
+
+
+
-
+
+
+
+
+
+
+

4.1.3 `parent`

The third column must be titled `parent`. This column refers to the parent cell population, or where the current cell population originates from. Similar to the `alias` column, `parent` names must be unique. This column cannot contain any commas, otherwise `openCyto` will assume the population has multiple parents and you will get an error message.



parent
root
nonDebris
singlets
lymph
viable
CD3
CD3
CD8
CD8
CD8
CD8
CD8

4.1.4 Remaining template columns

`dims`- channel or marker names for gating
`gating_method`- gating function (supported options listed above)
quadrantGate
rangeGate
quantileGate
mindensity
tailgate
cytokine
flowClust
boundary
singletGate
transitional
plolyfunctionalityGate
flowDensity `gating_args`- arguments to be passed to gating function `collapseDataforGating`- data is collapsed and replicated across all samples
`groupBy`- used to group samples into unique combinations
`preprocessing_method`- preprocessing function
`preprocessing_args`- arguments for preprocessing function

4.2 Creating the Template

The gating template can be created manually or assisted by the use of the `templateGen()` function. The function `templateGen()` will input the `alias`, `pop`, `parent`, and `dims` columns and the rest must be completed manually. To use `templateGen()`, you must input a GatingHierarchy object. In this example, that is **gh**, the subset created from **gating_set**.

```
gt <- templateGen(gh)
head(gt)
```

```
##        alias        pop                               parent
## 1   Singlets   Singlets                                 root
## 2       Live       Live                            /Singlets
## 3 Leukocytes Leukocytes                       /Singlets/Live
## 4       CD3+       CD3+            /Singlets/Live/Leukocytes
## 5       CD8+       CD8+       /Singlets/Live/Leukocytes/CD3+
## 6     CXCR1-     CXCR1- /Singlets/Live/Leukocytes/CD3+/CD8+
##                           dims gating_method gating_args
## 1                  FSC-A,FSC-H          <NA>        <NA>
## 2            Comp-Zombie Nir-A          <NA>        <NA>
## 3                  FSC-A,SSC-A          <NA>        <NA>
## 4 Comp-Alexa Fluor 532-A,SSC-H          <NA>        <NA>
## 5           Comp-BV570-A,SSC-H          <NA>        <NA>
## 6                    Comp-PE-A          <NA>        <NA>
##   collapseDataForGating groupBy preprocessing_method preprocessing_args
## 1                  <NA>    <NA>                 <NA>               <NA>
## 2                  <NA>    <NA>                 <NA>               <NA>
## 3                  <NA>    <NA>                 <NA>               <NA>
## 4                  <NA>    <NA>                 <NA>               <NA>
## 5                  <NA>    <NA>                 <NA>               <NA>
## 6                  <NA>    <NA>                 <NA>               <NA>
```

The auto-filled template will generate within the R Console and can then be saved locally with the following code. You will see that all columns besides the first four will contain NA values. These are the values that must be manually input to complete the .csv.

```
write.csv(gt, "gt.csv")
```

If you choose to create the gating template manually, the same conventions must be followed. Start with a blank spreadsheet. Next, fill in the 10 required column names. From there, use the manual gating hierarchy to fill in each cell population `alias`. Fill in the remainder accordingly.

There will likely be troubleshooting involved in this process. This is a great place to start if you're seeking more information on the gating template. The `openCyto` GitHub page is also very responsive to issues posted.

Chapter 5

Automate Gating

5.1 Load .csv into R

As noted in the previous chapter, there is a sample gating template titled *partial.csv* with the sample data. This may serve as a guide to creating your own. When the .csv gating template is complete, it is then read into R and saved as **`gt`**, a gatingTemplate object.

```
gt <- gatingTemplate("/Users/monhait/Desktop/partial_gt.csv")
```

The flow cytometry equipment at CSU will compensate and transform the data automatically. Other tutorials may highlight the steps to compensate and transform data, but these are not relevant to CSU at this moment. In the event that equipment changes, it may be necessary to complete compensation and transformation steps to prepare data. More on the current equipment used as CSU here.

5.2 Read in raw FCS files

Now that the GatingTemplate object has been loaded into R, you will need to load in raw FCS files to perform the automated gating on. For gating, these files must be in a GatingSet object type, which requires the following steps. When the path is saved using `list.files`, a character matrix of file names will be saved. Next, `read.ncdfFlowSet` will save FCS files as a ncdfFlowSet obect. The `GatingSet` function will then save the FCS files as a GatingSet object. In this form, the FCS files can be input and gated.

```
fcs_files <- list.files(path = "/Users/monhait/Desktop/data/group1_v_group2", full.names = TRUE)
ncfs  <- read.ncdfFlowSet(files = fcs_files)
gs_auto <- GatingSet(ncfs)
```

5.3 Apply Gating

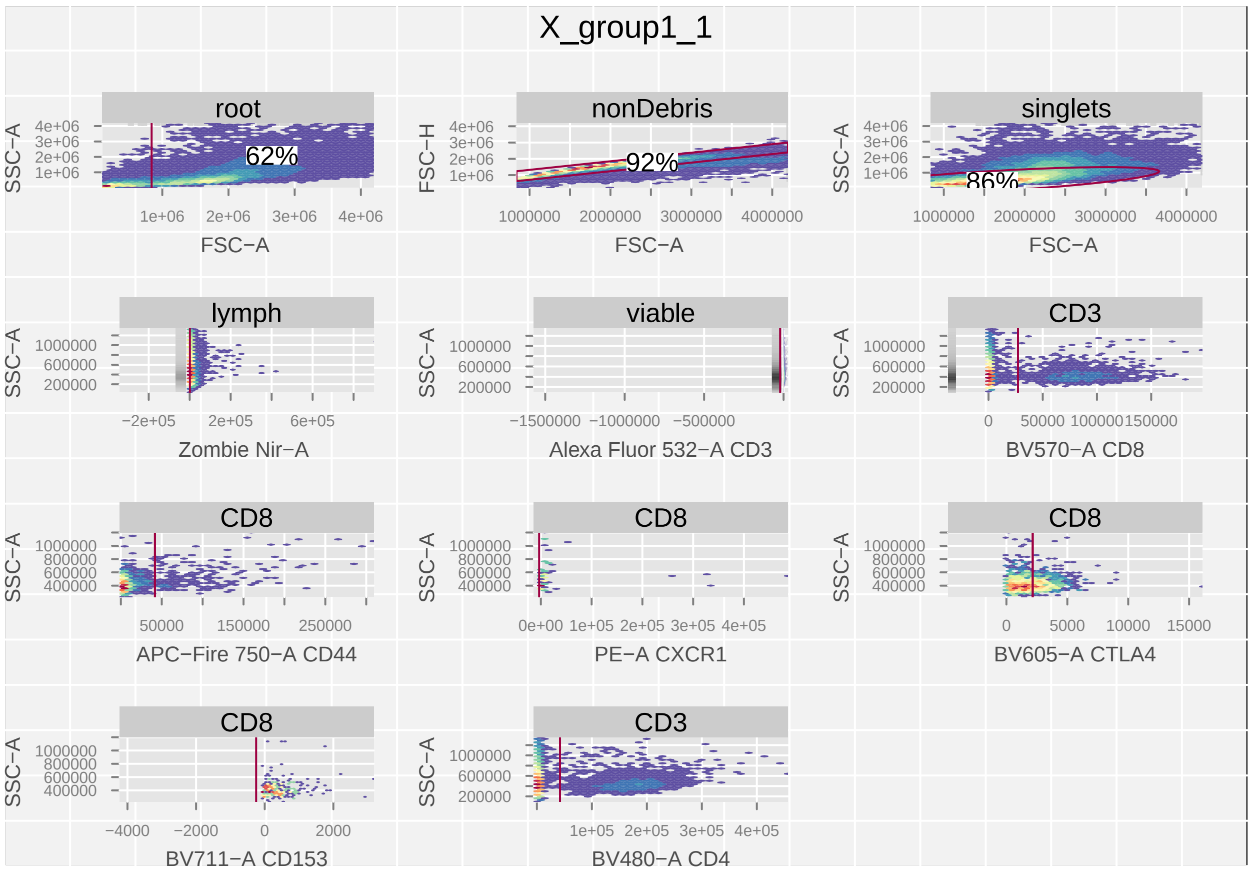
At this point, you now have GatingTemplate and GatingSet object to be used for gating. Apply your GatingTemplate object to the GatingSet object, where x = GatingTemplate object and y = GatingSet object.

```
gating(x = gt, y = gs_auto)
```

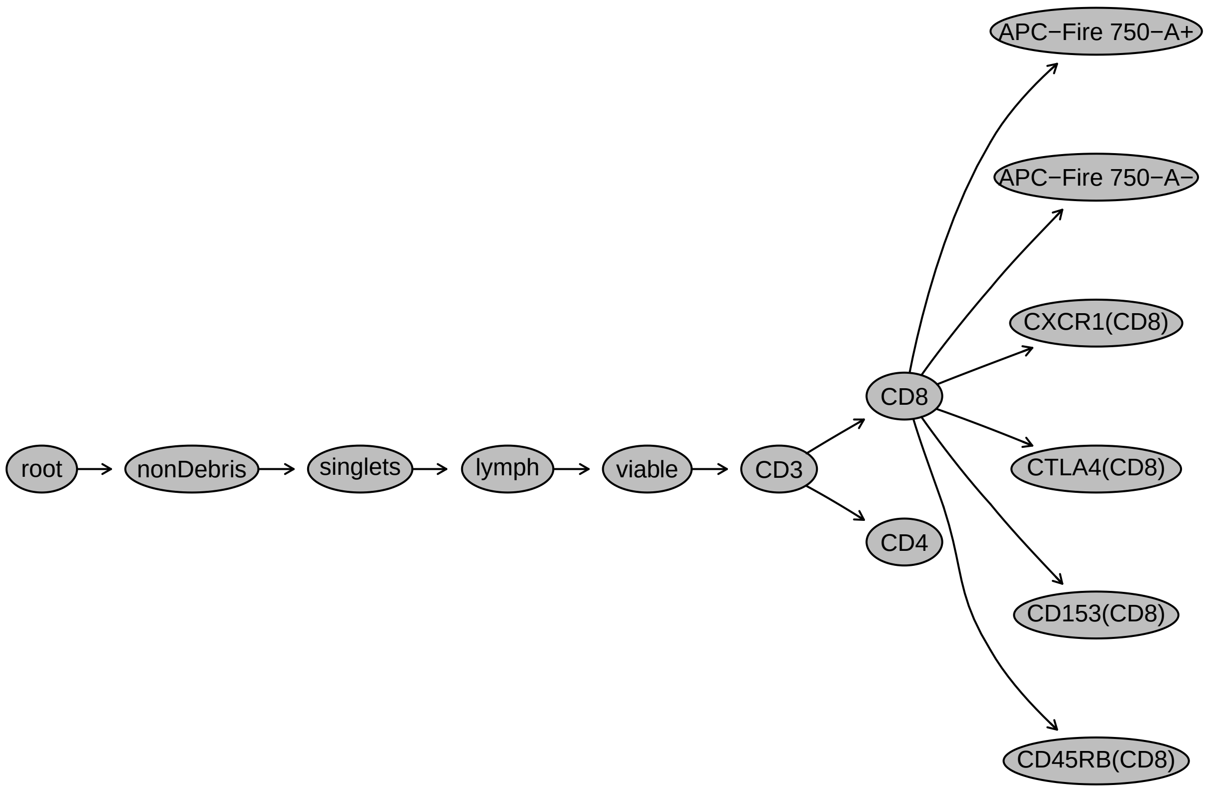
5.4 Plot Automated Gating

Just as before, plot both the gating hierarchy and the automated gates. You may notice extra nodes have been added to the hierarchy. Chapter 5 will highlight additional cusomization to remove unwanted nodes and improve upon visualization.

```
plotGate(gs_auto[[1]])
```



```
plot(gs_auto[[1]])
```



5.5 Population Statistics

Both counts and frequencies can be generated for analysis. This can be generate based on the analysis completed in R, or pulled directly from flowJo. To pull from flowJo, simply at `flowJo=TRUE` to either code chunk below.

Counts

```
head(getPopStats(gs_auto,statistic="count"))
```

```
##           name Population    Parent Count ParentCount
## 1: X_group1_1  nonDebris      root 87715      142158
## 2: X_group1_1   singlets nonDebris 80775       87715
## 3: X_group1_1      lymph  singlets 69710       80775
## 4: X_group1_1     viable     lymph 40982       69710
## 5: X_group1_1        CD3    viable 40937       40982
## 6: X_group1_1        CD8       CD3  1888       40937
```

Frequencies

```
head(getPopStats(gs_auto,statistic="freq"))
```

```
##           name Population    Parent Count ParentCount
## 1: X_group1_1  nonDebris      root 87715      142158
## 2: X_group1_1   singlets nonDebris 80775       87715
## 3: X_group1_1      lymph  singlets 69710       80775
## 4: X_group1_1     viable     lymph 40982       69710
## 5: X_group1_1        CD3    viable 40937       40982
## 6: X_group1_1        CD8       CD3  1888       40937
```

Chapter 6

Customization

It is possible that additional customization may be necessary when working with the `openCyto` framework. Below are three common customizations that will be outlined in this chapter.

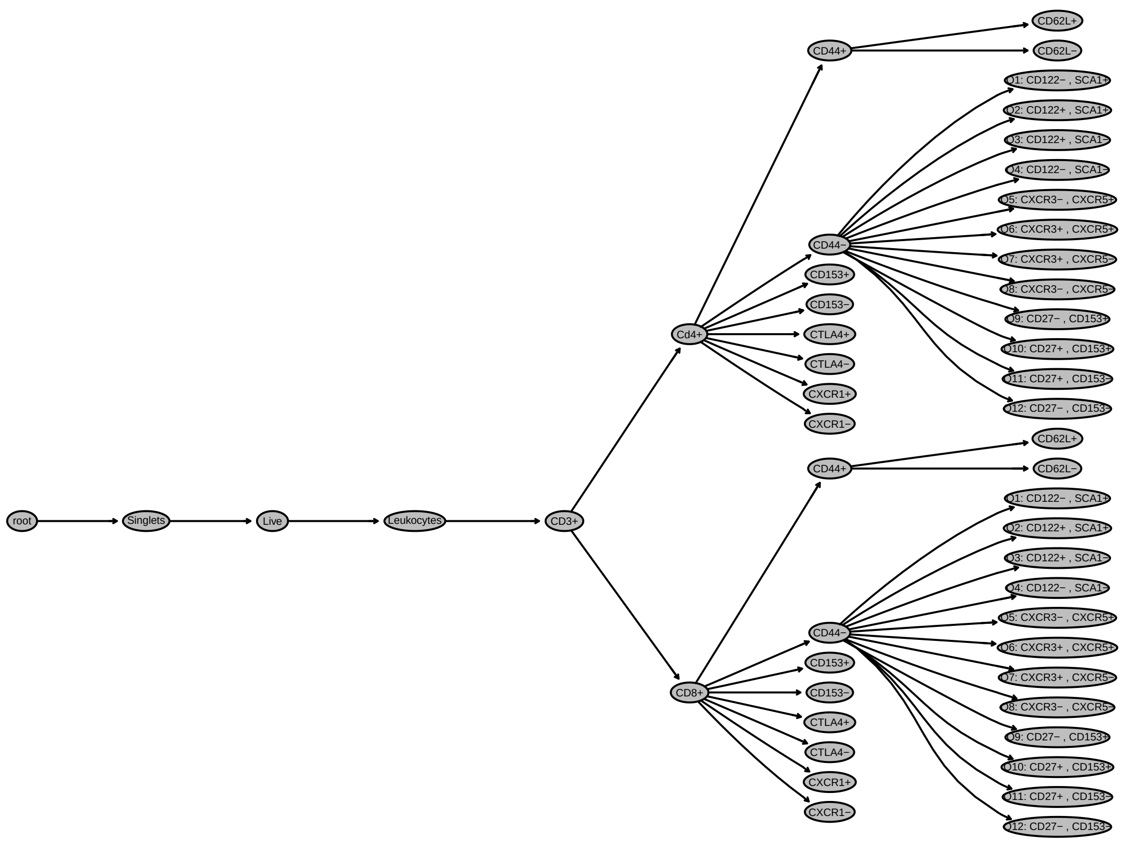
1. Hiding unwanted nodes
2. Renaming nodes
3. Adjusting plots

6.1 Hiding unwanted nodes

When automating analysis, there may be nodes that were not predefined in the .csv gating template or nodes that may not be of interest in your particular analysis. Plotting the gating hierarchy using the `plot()` function will display this and then nodes can be hidden based on need with the following code. Below is an example of a "full" gating hierarchy and then the same hierarchy with the CD3+ node removed.

Full Hierarchy

```
plot(gh)
```



CD3+ Removed Hierarchy

To remove nodes, first save the unwanted nodes as an R object named **nodesToHide**. Next, use the code following the `lapply()` function, only replacing **gs** with your GatingSet object name.

```
nodesToHide <- "CD3+"
lapply(nodesToHide, function(thisNode)setNode(gh, thisNode, FALSE))
```

```
## [[1]]
## NULL
```

6.2 Renaming nodes

Rename nodes based on your preferences with the following code. Within the `setNode` function, the first input is the current cell population name and the second is the desired change.

```
setNode("Live", "Viable")
plot(gh)
```

6.3 Adjusting plots

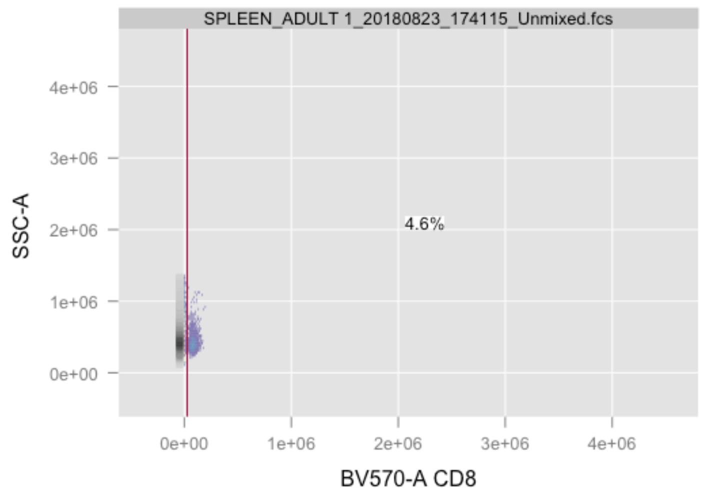
6.3.1 Adjust plot axes

As seen in chapter 2, it may be necessary to adjust the plot axes in order to best view the gates. This is done using the code below. Setting xlim and ylim to “data” adjusts plot based on the actual data range, rather than instrument specifications. Custom ranges can also be input numerically.

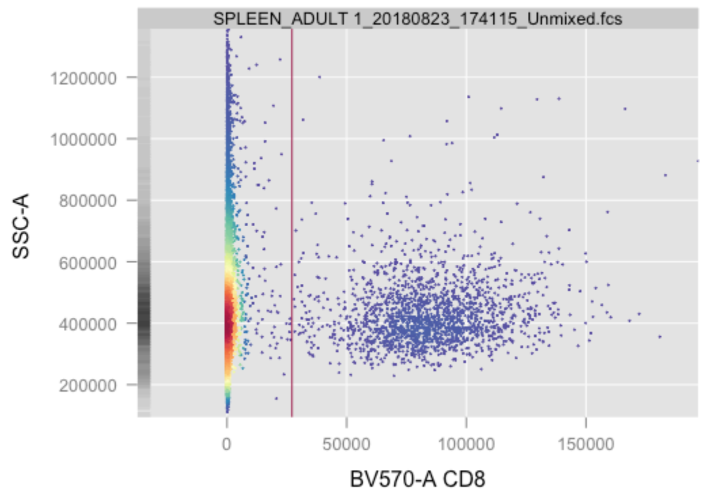
```
flowWorkspace.par.set("plotGate", list(xlim = "data",
                                        ylim = "data"))
```

Here is a comparison of xlim and ylim set as “instrument” and then “data”.

Instrument



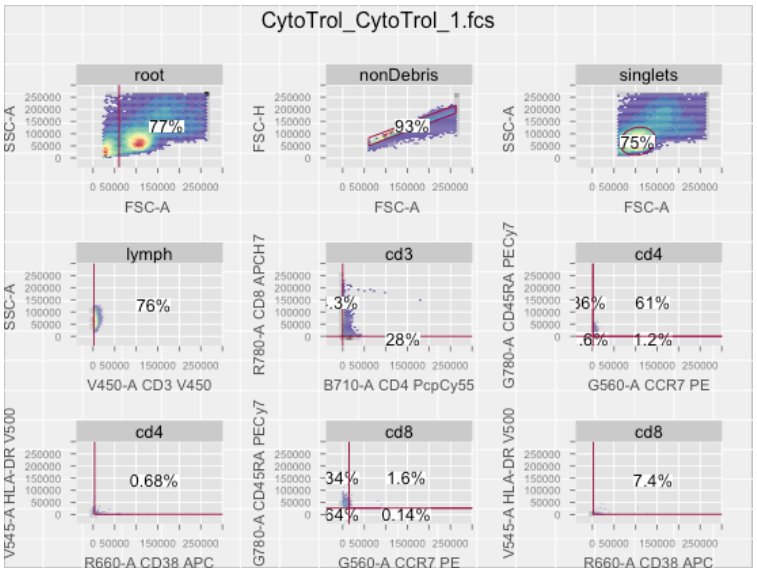
Data



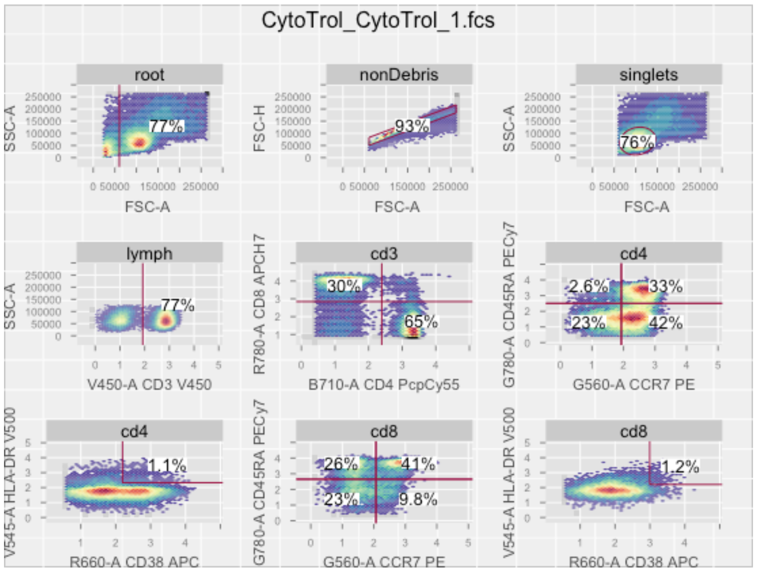
6.3.2 Transform data for better visualization

Although data will not be altered in any way, transformation may allow for better visualization. The most common form of transformation for flow cytometry analysis is biexponential. Below is a comparison of gates without transformation and gates that have been transformed.

Without Transformation



Transformed



Chapter 7

Appendix

7.1 Function and R Object Definitions

Function/Object Name	Definition
wsfile	flowJo .wsp file location
openWorkspace()	function used to read in **wsfile**
ws	read in data from flowJo
parseWorkspace()	function to extract FCS files from **ws**
gating_set	parsed FCS files to be gated
clone()	function used to create a clone of **gating_set**
gh	subset of gating_set
gt	.csv gating template
templateGen()	function used to generate a .csv template from existing manual gates
gatingTemplate()	function used to read in .csv template
gating()	function used to apply gates to a gating set
plot()	function to visualize gating tree
plotGate()	function to visualize gates