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

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

2019-05-07

Contents

Chapter 1

What's inside?

The goal of this tutorial is to take manually gated data from a .wsp flowJo file and use the Bioconductor `openCyto` framework to create an automated gating procedure within R which can be replicated on any dataset. The basic functions used in this tutorial include:

openWorkspace()
parseWorkspace()
gatingTemplate()
gating()
plot()
plotGate()

The example data used in this tutorial is from Colorado State University's Microbiology, Immunology, and Pathology Department. Alternatively, you can input your own flowJo Workspace and follow this tutorial, so long as the file is in .wsp format.

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 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

2.2 Required Packages and Installation

Before getting started, install and load the following libraries into a new R script. As you will see below, this tutorial uses the development version of `openCyto`. Use the following to ensure the correct packages are installed.

RStudio may also prompt you to download XQuartz and XCode, so it may be a good idea to go ahead and also download both before beginning.

To install

```
devtools::install_github("RGLab/openCyto", ref = "trunk")
install.packages("data.table")
install.packages("flowWorkspace")
install.packages("flowCore")
install.packages("flowStats")
install.packages("flowClust")
install.packages("plyr")
```

To load

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

Chapter 3

Working with your Manual Gating Scheme

Current methods for manually gating flow cytometry are both time consuming and costly, making automated gating 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 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
4. Clone data for later automation

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* file, which seems to cause many errors during this analysis. Saving analysis within flowJo is detailed here.

The result of step 1 will be replication of the manual transformation, compensation, and gating from the flowJo workspace saved as an R object. 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 name to open the .wsp file in R, save this as **ws**. Here is an example of saving and opening **wsfile**. Following this step, **ws** will be saved as a flowWorkspace object containing groups of samples.

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

```
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

Once this file exists as an R object, the raw FCS files are then 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. The `parseWorskpace` call requires **ws** (the .wsp workspace that was just read in from flowJo) and the name of the samples to read in. 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. 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 actual 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**.

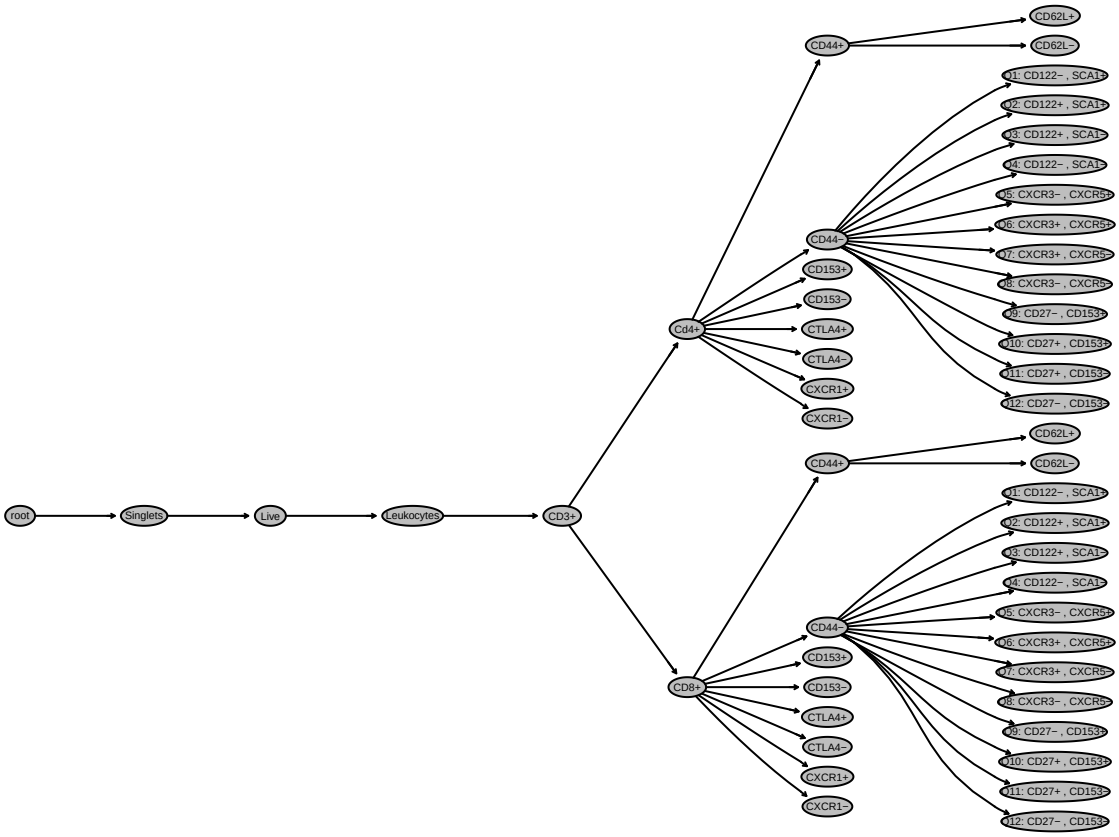
```
gh <- gating_set[[1]]
print(gh)
```

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

3.3.1 plot()

The plot() function will visualize the current gating hierarchy. This can be done for the entire gating hierarchy or a specific population as seen below.

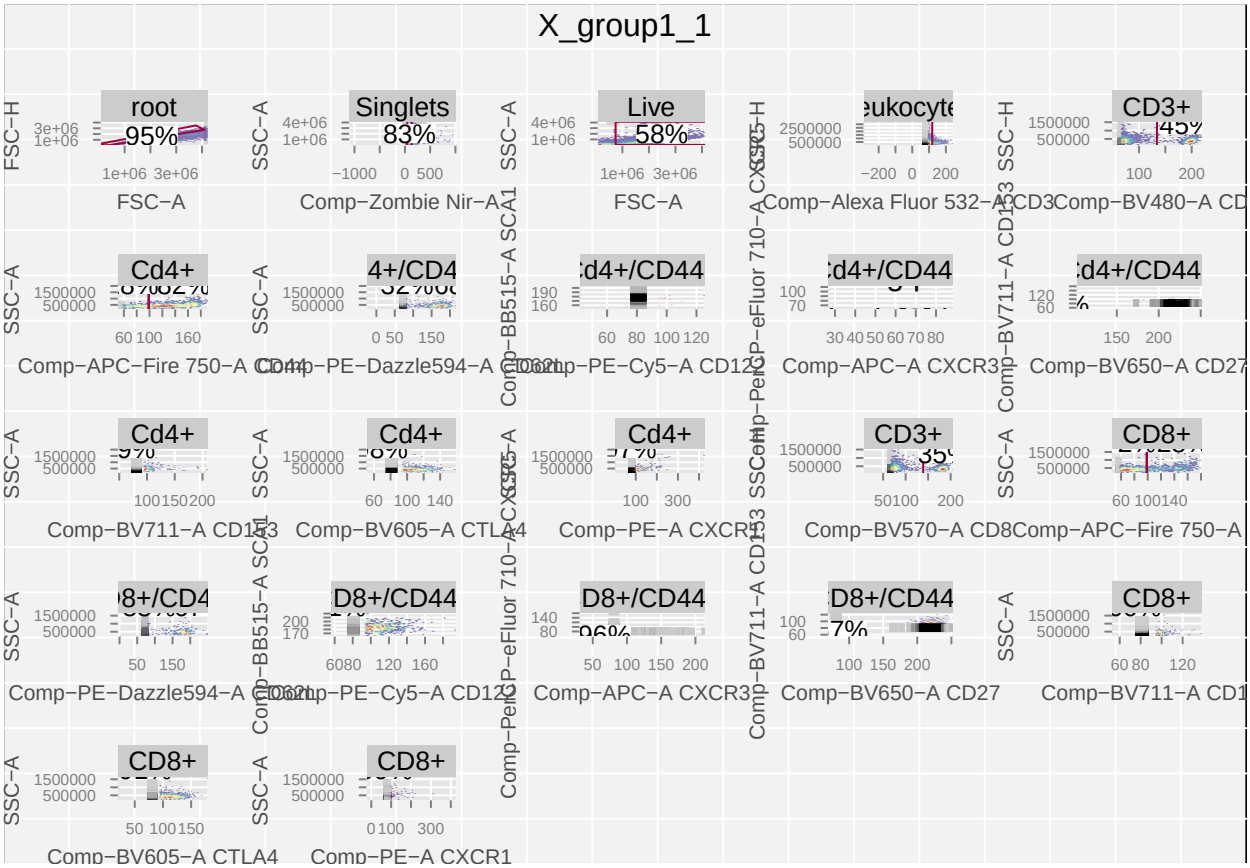
```
plot(gh)
```



3.3.2 plotGate()

The plotGate() function will gate the designated subset of your data according to parameters replicated from flowJo.

```
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.

3.4 Clone

Now, the manual gating scheme has been replicated in R using `openCyto` and verified for consistency with the original flowJo workspace. The final step in this chapter will clone **gating_set** into a new R object named **auto_gating**.

```
auto_gating <- clone(gating_set)
print(auto_gating)
```

```
## A GatingSet with 10 samples
```

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. Gating methods that are support currently by `openCyto` include:

quadrantGate
rangeGate
quantileGate
mindensity
tailgate
cytokine
flowClust
boundary
singletGate
transitional
plolyfunctionalityGate
flowDensity

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. The .csv must contain 10 predefined columns that will be listed below. Most cell population names listed in columns must be uniquely and follow certain guidelines. These include:

unique idenitifer for `alias` *and* `parent` *columns*
no commas in `parent` columns (otherwise `opencyto` will assume the population has multiple parents)
*restrict `pop` column to quadrant-only strings (++,+)

The required 10 columns are explained below.

4.1.1 Template Columns

`alias`- unique name/identifier for cell population
`pop`- +/- pattern to determine which subset or quadrant will be gated
`parent`- unique identifier for the parent population
`dims`- channel or marker names for gating
`gating_method`- gating function (supported options listed above)
`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. `TemplateGen()` will auto-fill 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.

```
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.

4.3 Load .csv into R

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**. The gating template will be saved as a GatingTemplate object.

```
gt <- gatingTemplate("/Users/monhait/Desktop/flow_cyto/automated_gating/data/gating_template/partial.cs
print(gt)
```

Chapter 5

Automate Gating

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.

5.1 Apply Gating

At this point, you will either return to your cloned GatingSet object that contains raw FCS files or bring in new data to be gated using the same parameters. Apply **gt** to the GatingSet object, where x = **gt** and y = data to be gated.

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

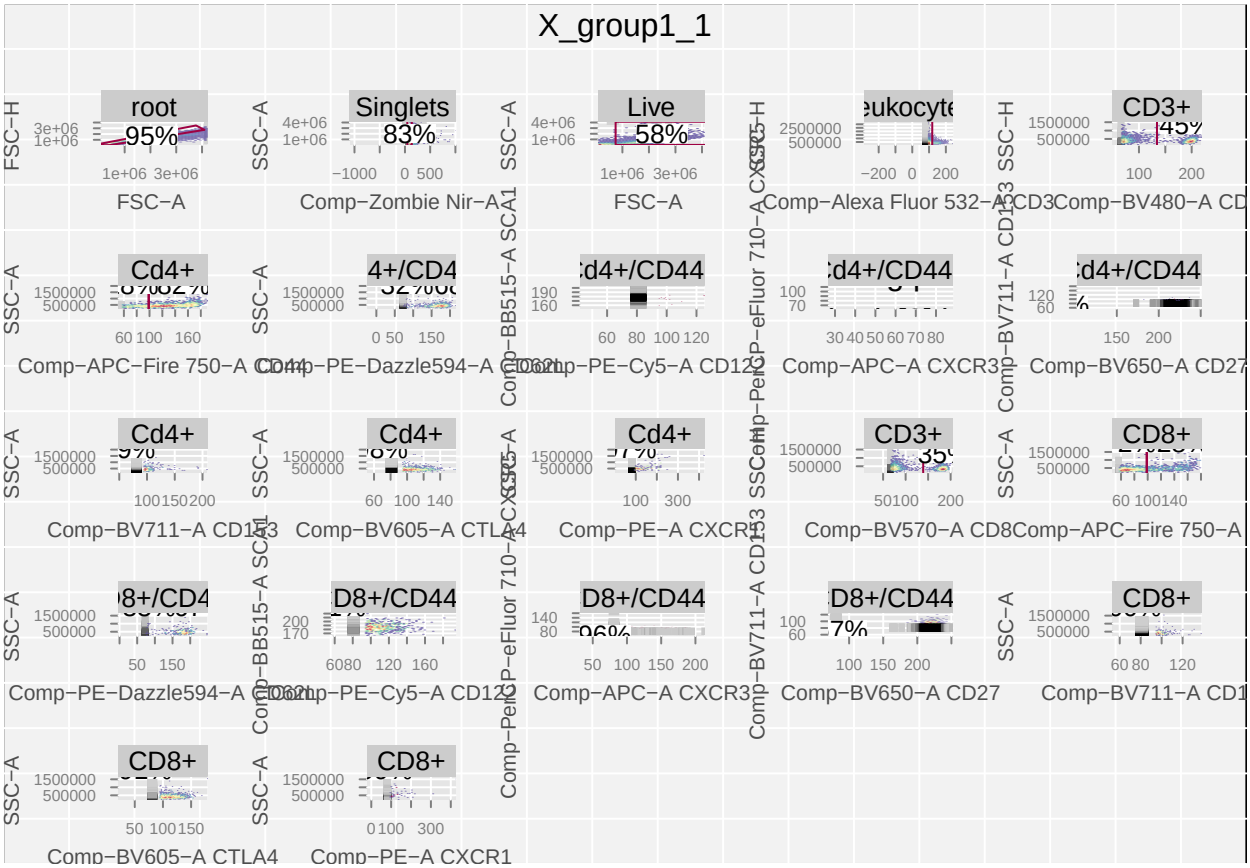
Just as before, plot both the gating hierarchy and the automated gates. You may notice extra nodes have been added to the hierarchy. Chapter 5 will highlight additional cusomization to remove unwanted nodes and improve upon visualization.

5.2 Plot Automated Gating

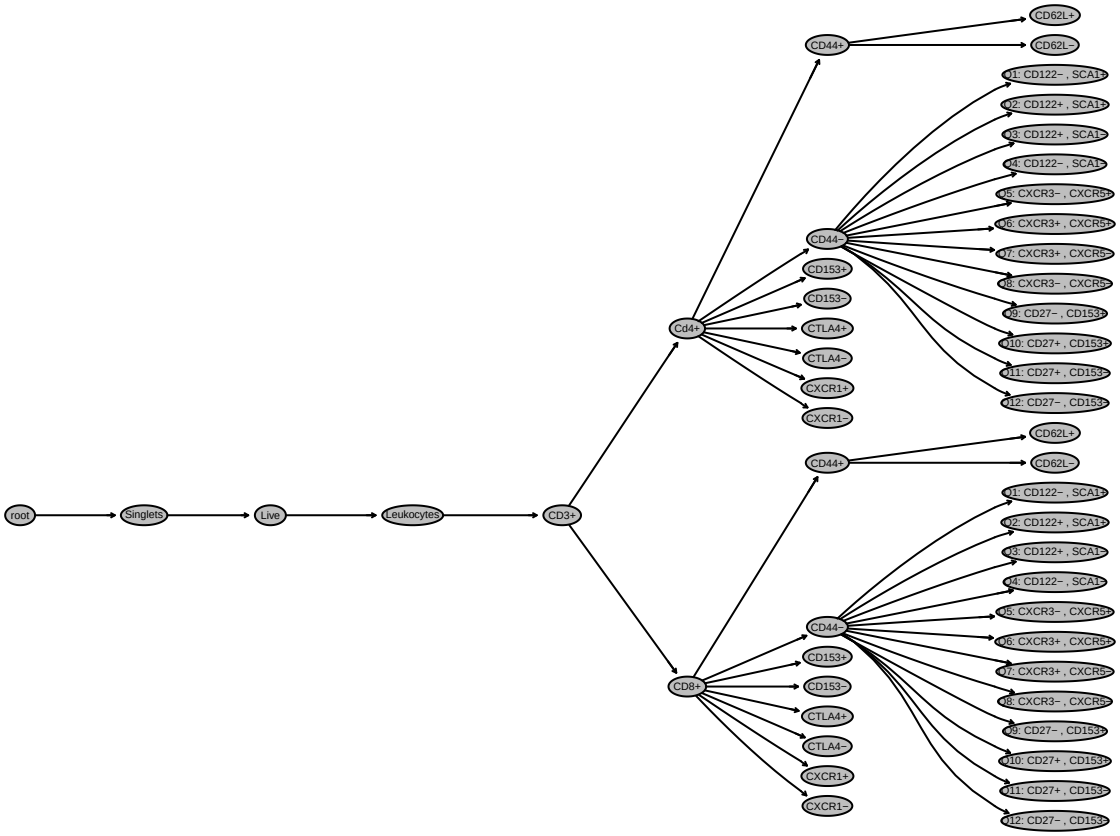
```
plotGate(auto_gating[[1]])
```

```
## Warning in min(x): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x): no non-missing arguments to max; returning -Inf
```




```
plot(auto_gating[[1]])
```



5.3 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(auto_gating,satistic="count"))
```

```
##          name Population     Parent  Count ParentCount
## 1: X_group1_1   Singlets       root 134378      142158
## 2: X_group1_1       Live   Singlets 111695      134378
## 3: X_group1_1 Leukocytes       Live  65104      111695
## 4: X_group1_1       CD3+ Leukocytes   4781       65104
## 5: X_group1_1       Cd4+       CD3+   2151        4781
## 6: X_group1_1 Cd4+/CD44+       Cd4+   1760        2151
```

Frequencies

```
head(getPopStats(auto_gating,satistic="freq"))
```

```
##          name Population     Parent  Count ParentCount
## 1: X_group1_1   Singlets       root 134378      142158
## 2: X_group1_1       Live   Singlets 111695      134378
## 3: X_group1_1 Leukocytes       Live  65104      111695
## 4: X_group1_1       CD3+ Leukocytes   4781       65104
## 5: X_group1_1       Cd4+       CD3+   2151        4781
```

```
## 6: X_group1_1 Cd4+/CD44+       Cd4+   1760        2151
```

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(gs, thisNode, FALSE))
```

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")
```

6.3 Adjusting plots

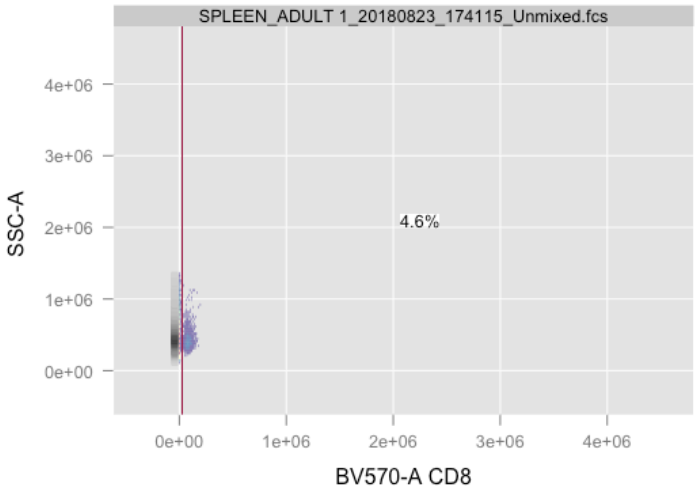
6.3.1 Adjust plot axes

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.

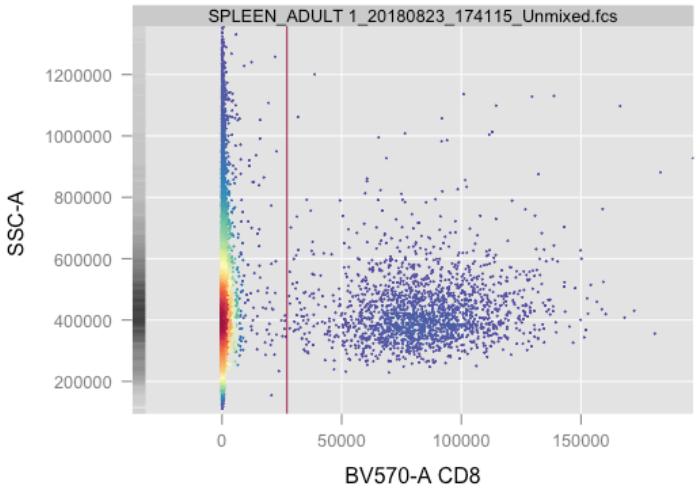
```
flowWorkspace.par.set("plotGate", list(xlim = "data",
                                       ylim = "data"))
```

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

Instrument



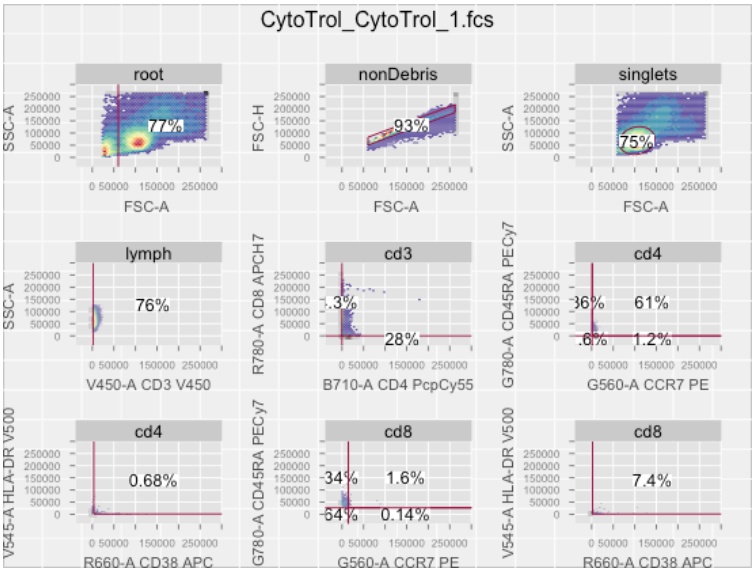
Data



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 bioexponential. Below is a comparison of gates without transformation and gates that have been transformed.

Without Transformation



Transformed