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

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

2019-05-13

Contents

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 designed
flowWorkspace	This package allows you to import basic flowJo workspaces into BioConductor and replicate the gating fi
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

```
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

```
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 <- "./tutorial/group1_v_group2.wsp"
```

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

```
print(ws)
```

```
## FlowJo Workspace Version  20.0
## File location:  ./tutorial
## 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 = "./tutorial/group1_v_group2", isNcdf = TRUE, s
```

```
## 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.

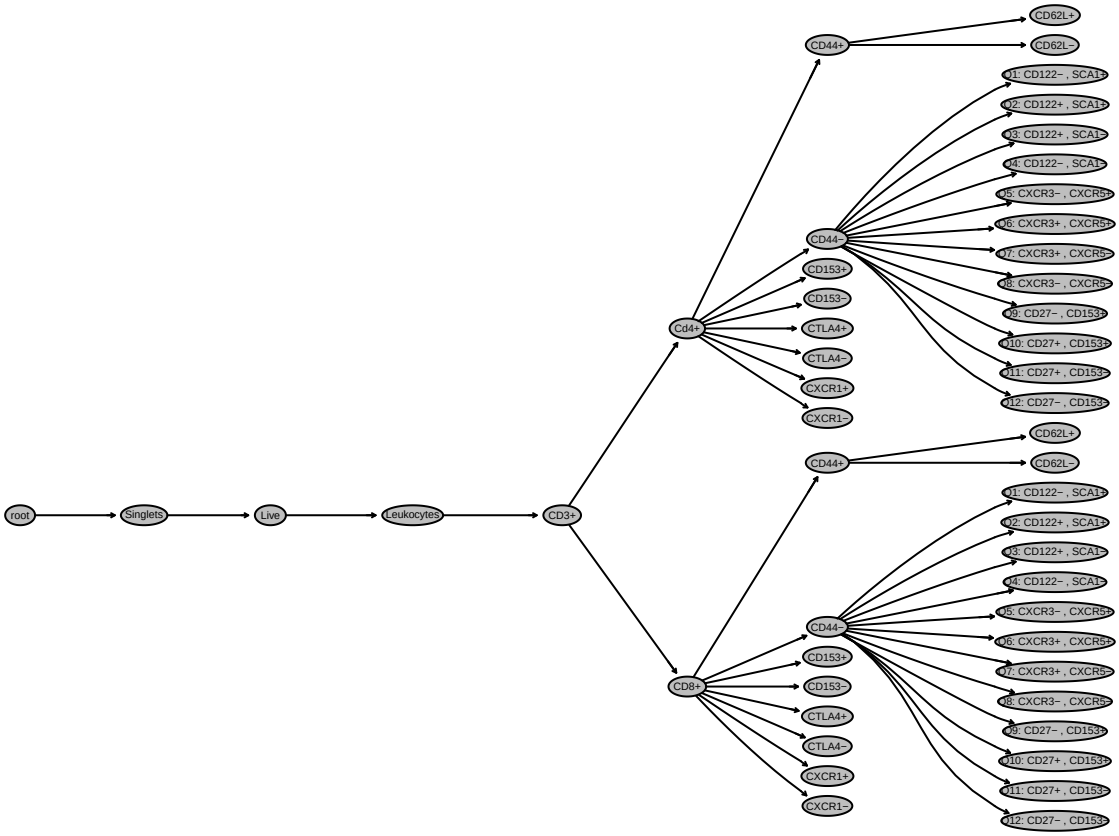
```
gh <- gating_set[[1]]
print(gh)
```

```
## 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.

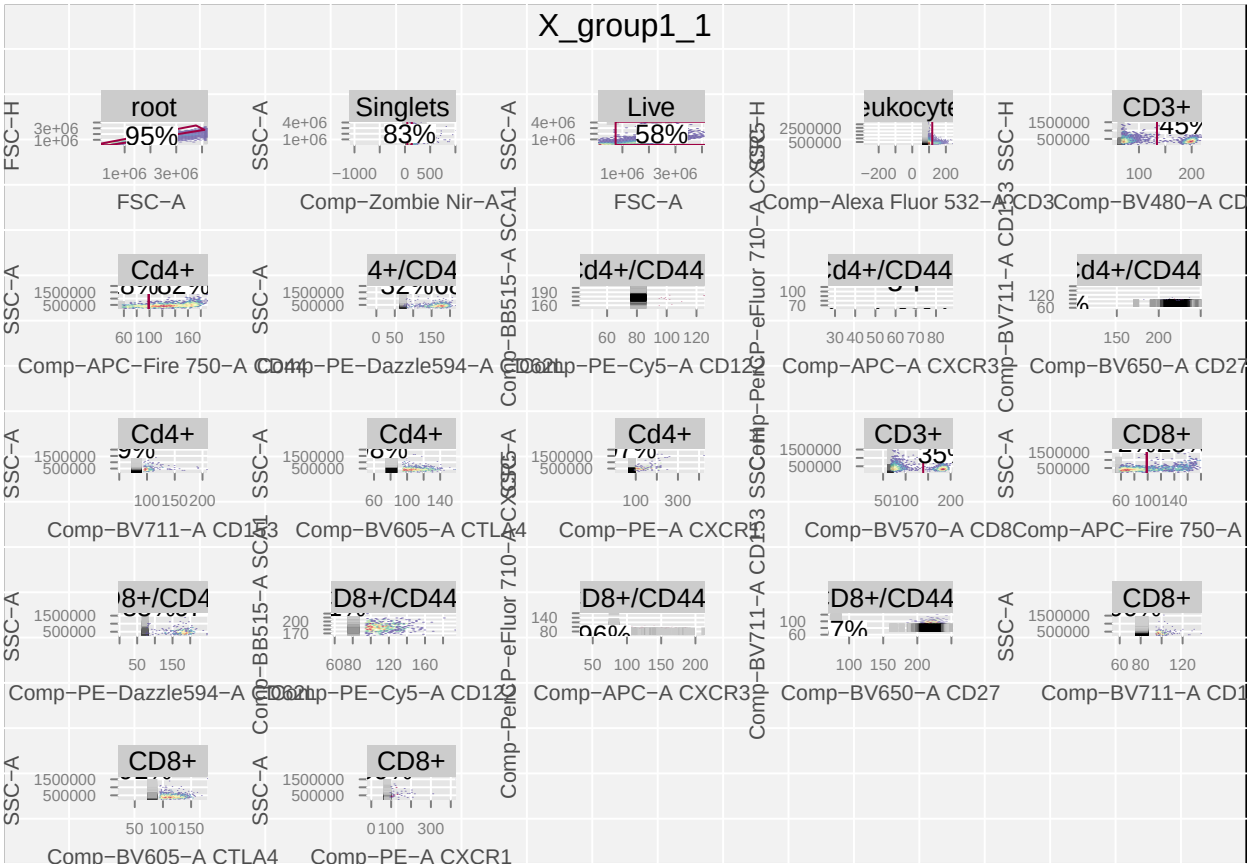
```
plot(gh)
```



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.