Peak Detection and Statistical Analysis of Karyotypic Variation from Flow Cytometry Data

by

Margot J M Henry

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfilment of the requirements of the degree of

Master of Science

Department of Statistics
University of Manitoba
Winnipeg

Abstract

Karyotypic variation is observed in fungal microbial populations isolated from ecological, clinical, and industrial environments and is also hallmark of many types of cancer. In order to characterize and understand the dynamics of karyotype subpopulations, we require an unbiased computational method to identify different subpopulations and quantify the number of cells within them. Flow cytometry is the gold standard method to measure genome size from each cell populations of interest. Cells within a population are typically measured from all phases of the cell cycle (G0/G1 prior to DNA replication, S phase during replication, and G2/M when cells have doubled their DNA but haven't yet divided). Mathematical models can be fit to the distribution of genome sizes to determine the base ploidy of the population. These algorithms only work for single ploidy populations. When there are multiple subpopulations of mixed ploidy, the researcher must manually divide the original population into subpopulations prior to analysis. This is subject to considerable bias and is not feasible when there are multiple subpopulations.

We developed an unbiased method to quantify karyotypic variation in populations from flow cytometry data and will release an open-source Bioconductor package, *ploidyPeaks*. The existing *flowCore* Bioconductor package was used

to load flow cytometry data from reference cell populations with known and variable karyotypes into the R programming language.

We used reference populations with known ploidy to determine a threshold for single ploidy population and flag the rest as possible mixed ploidy populations. We implemented a peak detection algorithm to identify G0/G1 and G2/M populations, we identified karyotypic subpopulations for mixed populations, and applied a nonlinear least squares to test how well data from each population fit the Dean-Jett-fox cell cycle models to provide a confidence term.

Our method improves on existing algorithms by providing a measure of model fit, and the ability to quantify populations that contain multiple karyotypic subpopulations in an unbiased manner.

Acknowledgment Page

I would like to express my gratitude towards my advisor, Dr. Aleeza Gerstein, for her ongoing support, guidance, and patience. I have learned so much from being your first Co-op student to your first statistics Masters student. You created such a welcoming environment that made me want to come back and do a Masters.

Thank you to the members of MicroStats lab for the friendship and always willing to offer a helping hand throughout my time in the lab. Thank you to Thanh Hoang for the code review.

I am thankful to the committee members Dr. Max Turgeon, Dr. Saumen Mandal and Dr. Pingzhao Hu for their time and advice. An extra special thank you to Dr. Max Turgeon for all his help and advice on building an R package.

I would like to thank the support of the Natural Sciences and Engineering Research Council of Canada and the Visual and Automated Disease Analytics (VADA) graduate training program for their financial assistance.

Thank you to my family and friends for their support and for always providing me with study snack.

Dedication Page

Dédié à ma famille

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Genome size variation in cell populations

Populations of cells are heterogeneous, since variation is continually introduced through mutation with every cell division. Although the majority of mutations are phenotypically neutral, a small but significant fraction of mutations cause either deleterious or beneficial phenotypes in the cells in which they arise. The rate at which variant cells are selected for or against is related to the effect size of the mutation and the population size. A considerable amount of effort in recent years has examined how different factors influence the rate that new genomic mutations arise, the distribution of their effect sizes, and how these factors influence the rate at which they either spread or are lost from the population (Rice et al. (2015); Lang et al. (2013)). In contrast, the role and trajectory of karyotypic mutations—those that increase or decrease the number of chromosomes (aneuploidy mutations) or alter the number of chromosome

sets (ploidy mutations) have been less well studied.

Karyotypic variation is commonly observed in fungal microbial populations isolated from a wide variety of contexts (Gerstein and Sharp (2021)). Populations of the budding yeast, *Saccharomyces cerevisiae*, a model eukaryote (Gerstein and Sharp (2021)), that contain individuals that differ in base ploidy have been isolated from all of its' known niches including the ecological environment (Ezov et al. (2006)), clinically (Zhu et al. (2016)), and industrially (Peter et al. (2018)). Similar within-population variation from many different contexts has also been observed in many additional species that span the fungal tree of life (Gerstein and Sharp (2021)). In some cases, karyotypic mutations have been shown to have a large effect on phenotype and fitness (Gerstein and Sharp (2021)). However, many of the studies that have characterized natural isolates have not done so systematically, and it is not always possible to recreate the relevant environmental condition in the lab to compare between cells of differing karyotype to understand their effect.

Adaptive laboratory evolution (ALE) studies have emerged as a powerful way to test how different factors influence the phenotypic effect and eventual evolutionary fate of karyotypic variants. In ALE, the researcher typically sets up a series of replicate populations from the same ancestor that are then evolved in the lab to a specific environment. ALE evolution experiments initiated from different fungal microbes adapting to diverse environments have found both ploidy increases and decrease (Harari et al. (2018)). In all cases, there is a

period of time after the new karyotypic mutation appears in the population when the population is heterogeneous for karyotype, i.e., when multiple different subpopulations are observed. In order to understand the dynamics of karyotype subpopulations and their role in adapting populations, we require a method that can identify different subpopulations and quantify the number of cells within them.

1.2 Flow cytometry

Flow cytometry is the gold standard method to measure genome size (as a proxy for karyotype) from populations of cells. A population of cells from different stages of the cell cycle are stained with a fluorescent dye that binds to DNA. The flow cytometer is used to measure genome size from typically ∼10 000 cells in a population by passing them one-by-one across a laser. Cells are present in all phases of the cell cycle: G0/G1 (which we will refer to as G1) prior to DNA replication, S phase during replication, and G2/M (which we will refer to as G2) when cells have double the DNA but haven't yet divided. The flow cytometer scatters light though all suspended particles, the light scatters are read into a detector that can visualize the light, and this information is passed on to an analyzer that converts them to electrical signals.

Multiple parameters are measured for each cell. Physical characteristics such as cell size and shape are measured based on light scatter, and genome size, which corresponds to the amount of dye in the nucleus, is measured by the

fluorescence intensity. Forward scatter (FCS) measures the light scatter in the same direction as the laser, providing information on cell size, while side scatter (SSC) measures scatter on a 90° angle, providing information on cell granularity (McKinnon (2018)). When the cells get passed through the laser, it pulses in specific fluorescence channels that are named based on machine settings, and the height (-H), area (-A), and width (-W) are recorded. The fluorescence intensity parameters are typically labelled by arbitrary fluorescence/laser numbers (e.g., FL1, FL2, etc.) or the name of the dye used to stain the DNA (e.g., FITC, PI, SYBR, SYTOX etc.).

Genome size correlates with the fluorescence intensity of the G1 phase cells. Genome size from cell populatitons of interest can be compared to the G1 mean from control populations where the ploidy is known. Software such as Flow-Jo (Treestar) can be used to fit a built-in cell cycle algorithm to the fluorescence intensity data to determine G1 mean. There are two different cell cycle algorithms that are built-in to Flow-Jo to measure genome size, Watson Pragmatic (Watson et al. (1987)) and Dean-Jett-Fox (Fox (1980)). They have generally the same process; they both assume that cells in the G1 and G2 clusters are Gaussian distributed but vary in how they fit the distribution of S phase cells.

With fungal microbial populations, the two models typically give nearly identical results (Figure 1.1 A). However, neither is able to analyze populations that contain multiple subpopulations of mixed ploidy, unless the user manually

sets up gates to divide the original population by eye. Although that can work to some degree some of the time, it is very time consuming, is subject to bias (you will only find the populations that you have identified by eye), and is not feasible when there are multiple subpopulations (e.g., Figure 1.1 B). The current software also does not provide any information on doublets. Doublets are an excess of cells that happen when two cells pass through the laser at the same time and can be mistaken for a cell population (G1 or G2) of high genome size (Bagwell et al. (2020)). Doublets typically appear around the G1+G2 range and sometimes around the G2+G2 range. Doublets around the G2+G2 range can be easily mistaken for a G2 peak and can cause errors in the analysis as they can lead to the improper amount of subpopulations and therefore must be accounted for. There is currently no statistical confidence to let the user know if the the G1/G2 clusters are properly paired together and if the microbial sample is correctly classified as a single or multi-karyotype population.

1.3 Thesis Objective

Our objective was to develop an open-source method to quantify karyotypic variation in populations from DNA fluorescence intensity data. The target audience is researchers who want to analyze their data in an unbiased manner, without having to be experts in coding. To accomplish this, we developed a Bioconductor package, *ploidyPeaks*, that uses peak detection algorithms based

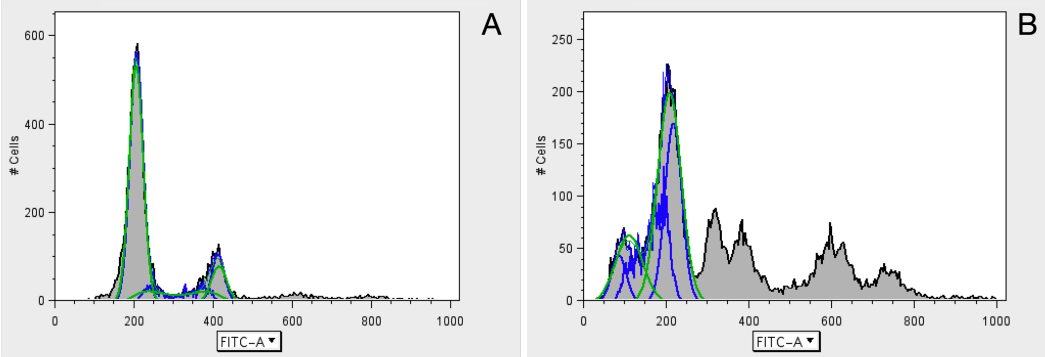

Figure 1.1: The existing workflow in Flow-Jo. A) Cell-cycle analysis for a single population. The blue fitted line represents the Watson-Pragmatic algorithm, the green fitted line is the Dean-Jett-Fox algorithm. B) Cell-cycle analysis for a multi-karyotype population. Example of multi-karyotype populations that are not amenable to manual analysis. The Watson-Pragmatic algorithm and the Dean-Jett-Fox algorithm did not perform well.

on the histograms of DNA fluorescence intensity data to identify G1 and G2 peaks. We consider every local maxima in the fluorescence intensity data to be a possible peak, and are able to conduct cell cycle analyses on samples that contain multiple subpopulations without requiring user input. With this package, we provide an unbiased quantitative method that will enable future studies that seek to better understand of the role of karyotypic subpopulations in evolution.

1.4 Thesis Outline

The thesis proceeds as follows. Chapter 2 discusses the development of the *ploidyPeaks* package. We will talk about the current state of the field and existing BioConductor packages that work with DNA fluorescence intensity

data. We will then take you through the functions that make up *ploidyPeaks*, such as the gating process, the visualization of the data, and the analysis of DNA fluorescence intensity data, including a discussion of the different steps and decisions that were considered during package development. The use and production of mock data with a known number of subpopulations to test the functions is explained and used to demonstrate how certain functions work. Chapter 3 will then provide results from a use case example that used the package. Chapter 4 includes a discussion of the work presented and potential future directions. The thesis appendix includes the package vignette, comprehensive document that explains the package and provides the user with a step by step guide on how to analyze DNA fluorescence intensity data using *ploidyPeaks*. The information in the vignette overlaps entirely with text within the thesis; it is included here for interest, as it will be a major public deliverable produced from the thesis.

Chapter 2

Development of the *ploidyPeaks* Package

In this chapter we present the process of building our package *ploidyPeaks*. *ploidyPeaks* builds off of existing packages that work with DNA fluorescence intensity data. This chapter will walk you through the functions we created, the use of the existing packages and the decisions behind our methodology.

2.1 Current packages

BioConductor is an open source software repository that contains over 2140 different packages created to facilitate analysis of biological data. The goal of BioConductor is to provide statistical and graphical methods for analysis. The majority of BioConductor packages are created as techniques such as R packages, and focus on analysis of often large datasets generated from DNA

microarrays, genome sequencing, flow cytometry (Gentleman et al. (2004)). *flowCore* is a BioConductor package that provides S4 data structures and fundamental functions to handle flow cytometry data from *.fcs* files, the standard format for flow cytometry data (Ellis et al. (2022)). A sample is a single *.fcs*, and samples can contain either a single population (a single G1/G2 pairing) or multiple subpopulations(multiple G1/G2 pairings). An S4 data structure provides functionality object oriented programming and it has slots to store data information (Ellis et al. (2022)). *flowCore* has over 100 functions to bring in, manipulate, transform, and save flow cytometry data in the R programming language. To get flow cytometry data into R, the `read.FCS()` (one *.fcs*, which *flowCore* terms flow frame) and `read.flowSet()` (multiple *.fcs*, which *flowCore* terms a flow set) imports the data, check if the data is in a proper *.fcs* file, and saves them in a S4 structure. There are many functions to transformation to the data in various ways (e.g., such as arc sinh, truncate, log). There are also functions to subset the data and apply a gate to the data. A gate is a boundary that is applied to a plot to subset the cells within the boundary for further analysis. Within *flowCore* `rectangleGate()` creates a rectangular gate and `Subset()` creates a new dataset from the data that is within the gate. `write.FCS()` saves the newly manipulated, transformed or subset flow cytometry data.

ggcyto is a package that is used to visualize flow cytometry data using *ggplot2* functions in different ways, one of the main functions to do this is `autoplot()`

which creates a univariate or bivariate density plot from the flow cytometry data (Phu et al. (2018)). This package has many functions to enhance visuals by adding many features comparable to *ggplot2*, e.g., `as.ggplot()` transforms the data into a regular *ggplot* object, `geom_gate()` allows the user to visualize a gate on their plotted data, `geom_stats()` applies a label on the graph for a certain statistic. Other functions are used to scale the values on the axis, the rotation of the axis labels, or to change the axis limits.

A few packages exist that can analyze single-population ploidy data. *flowClust* is a package that uses a model-based clustering approach to determine the number of clusters within the sample. The package estimates the number of clusters by Bayesian Information Criterion and the Integrated Completed Likelihood (Lo et al. (2009)). *flowClust* provides the user with the total number of clusters within the sample but does not provide any G1/G2 subpopulation pairings. A second package *flowPloidy* analyzes flow cytometry histograms of DNA fluorescence intensity data to determine sample ploidy (Smith (2022)). This package uses a non-linear regression model to find the cell populations that contribute to the total sample. The G1 and G2 peaks are modelled as normal distributions and the regression model estimates the means of each. The function `FlowHist()` creates a histogram and applies the model. `FlowHist()` requires some user input on population information. The function assumes there are 2 subpopulations, if there are more or fewer the user must indicate otherwise. There is a logical parameter called *g2*, default is true, to indicate if

each sample has a G2 peak. There are some samples that will only have one peak (no G2 peak), which will require requires the user to change the parameter. The vignette talks about local minima traps, which is when the non-linear regression model misclassifies a local minima as the G1 or G2 peak (Smith (2018)). The solution was the creation of a function called `browseFlowHist()` a graphical user interface (GUI) that allows the user to manually correct and review the model applied to the histograms. If the model failed or did not correctly identify the peaks, this function allows the users to use their mouse and click on the GUI to properly identify the peaks.

Each sample is a different from one to another. We cannot assume that the number of peaks, number of subpopulations, whether there are doublets, will be the same for the samples. We sought an algorithm that was able to find peaks in all the samples, but is also able to identify additional peaks corresponding to the (potential) subpopulations and doublets. The algorithm also needs to be able to find the subpopulation pairings (the G1/G2 pair that make up a subpopulation) to determine the number of subpopulations within each sample. In addition, we wanted an method that would provide a measure of fit of a model for each population to create a confidence term. This will give the user a sense of how well the algorithm performed. *ploidyPeaks* identifies the numbers of peaks using peak detection algorithm, finds the G1/G2 pairing utilizing the methods of Watson Pragmatic and Dean-Jett-Fox, and creates a confidence term by applying the Dean-Jett-Fox cell cycle algorithm using

Nonlinear least squares to get a residual standard error, which will indicate how well our model fit the data.

2.2 ploidyPeaks

2.2.1 Preparing the Data

The functions `read.FCS()` and `read.flowSet()` from the *flowCore* package are used to read the DNA fluorescence intensity data(*.fcs*) into R as an S4 object. We access the raw data slot, which is a numerical vector of the fluorescent dye value measured for each cell in each sample. Each experiment has typically many samples, `read.FCS()` reads in a single sample while `read.flowSet()` reads in multiple samples.

2.2.2 Gating

Gating is typically the first step for analysis of DNA fluorescence intensity data. Improper gating can lead to computational errors and incorrect results; if you do not remove the debris,which includes inert particles such as dust that are introduced during the cell preparation process as well as large clumps of cells that stick together which can cause an excess of cells along the plot margins, there is a possibility that it can appear as a false population. On the other hand, if you remove too many cells, there is a possibility that you removed part of (or even a whole) population of interest. Following standard

flow cytometry practices, gating in *ploidyPeaks* is accomplished by creating boundaries a bivariate density plot with SSC-A on the y-axis and the fluorescent dye intensity on the x-axis. This creates a rectangle that will retain the cells within the boundaries and remove all other cells from the subsequent analyses (Figure 2.1). The default setting is to include the majority of points, excluding those only truly on the margins.

We wrote two gating functions: `rectGateFlowFrame()` applies the gate on a single sample, while `rectGateFlowSet()` applies the gate to a whole data set, i.e, a series of multiple flow frames. We use `rectangleGate()` and `Subset()` from the *flowCore* package to create the rectangular boundary and subset the cells within the boundary. `write.FCS()` was used to save the gated data for further analysis. We used *ggcyto*'s `autoplot()` to visualize the bivariate density plot, `geom_gate()` to apply a red rectangle on the plot to visualize the gate, and `as.ggplot()` to use `ggplot()`'s feature of graphing two plots side by side.

2.2.3 Peak Detection

The univariate graph for a sample is a histogram of the underlying cell-level data. They have the fluorescence intensity (corresponding to genome size/amount of DNA) on the x-axis, and the cell counts on the y-axis, peaks indicate when the cell count is high which typically corresponds to the G1 and G2 phases, where cells spend the majority of their time. We need a method to work on the

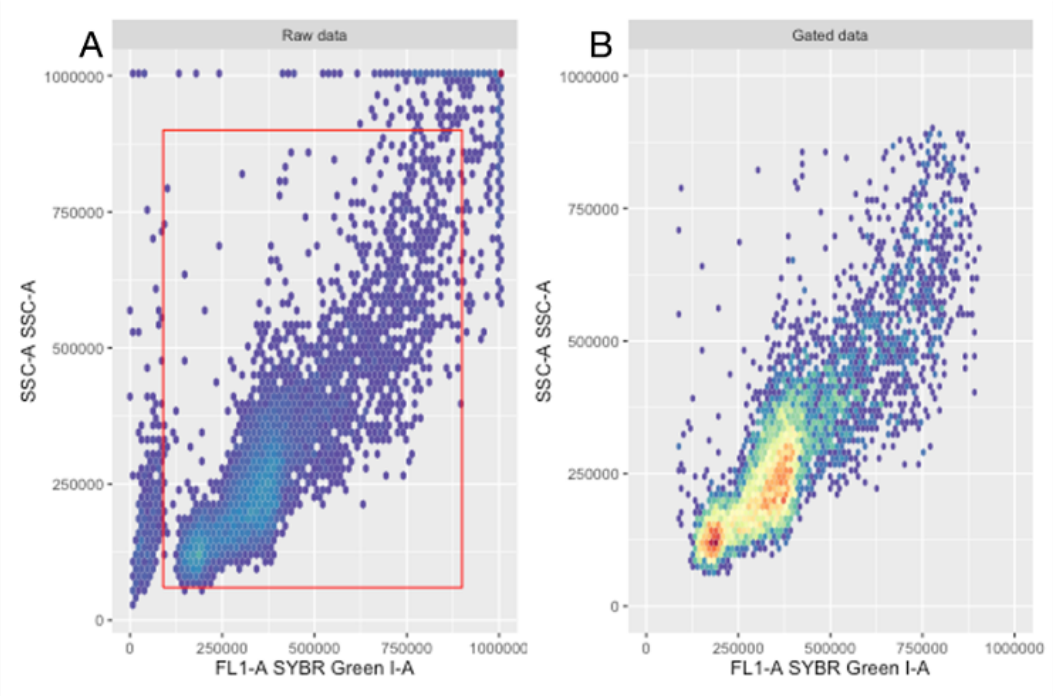

Figure 2.1: The x-axis is the fluorescence intensity data (the DNA was stained with dye called Sybr Green which was read in the FL1-A channel), the y-axis is the side scatter area(SSC-A). Each point on the bivariate plot represents a cell and the density is indicated by a red (high density) to blue (low density) gradient. A) The red rectangle is the gate applied to the sample, the cells outside the gate are removed. B) The cells within the gate are retained and used for further analysis.

underlying cell-level data to identify the cell clusters that represent the peaks in the univariate histogram. The method needs to be able to distinguish between each individual G1 and G2 peak (as well as peaks that do not correspond to cell cycle phases such as doublets or small peaks that arise due to stochasticity in data collection), find the G1/G2 paired subpopulations within the data, and able to identify peaks of all shapes and sizes since this is not always

consistent.(Figure 2.2).

We investigated a number of different methods to identify subpopulations, since this is the critical step of what we set out to achieve.

We started with machine learning clustering algorithms, and initially considered K-means and DBSCAN. Central to K-means clustering algorithm is the input of the number of clusters, which is referred to as k. We used the silhouette method to determine the optimal number of clusters for each sample. The silhouette index is calculated as

$$S(i) = \frac{B(i) - A(i)}{max_i\{A(i), B(i)\}} \tag{2.1}$$

Where $A(i)$ is the average difference of all the points that x_i belongs to and $B(i)$ is the lowest average difference of x_i to any other cluster of which x_i is not a member, and the average silhouette is the average of all $S(i)$ (Rousseeuw (1987)). The average silhouette value is between [-1,1], where 1 means the data points within the cluster are similar and far away from the other clusters, 0 meaning there are overlapping clusters, and -1 being low of discriminatory power. When finding the optimal number of clusters, we iterated through $k = 1, \cdots, 15$ and the k with the highest average silhouette value was chosen for the K-means algorithm for that sample. When we applied the silhouette method to datasets where we knew the number of subpopulations within the sample, we found that the results were inconsistent. We used sample with known number

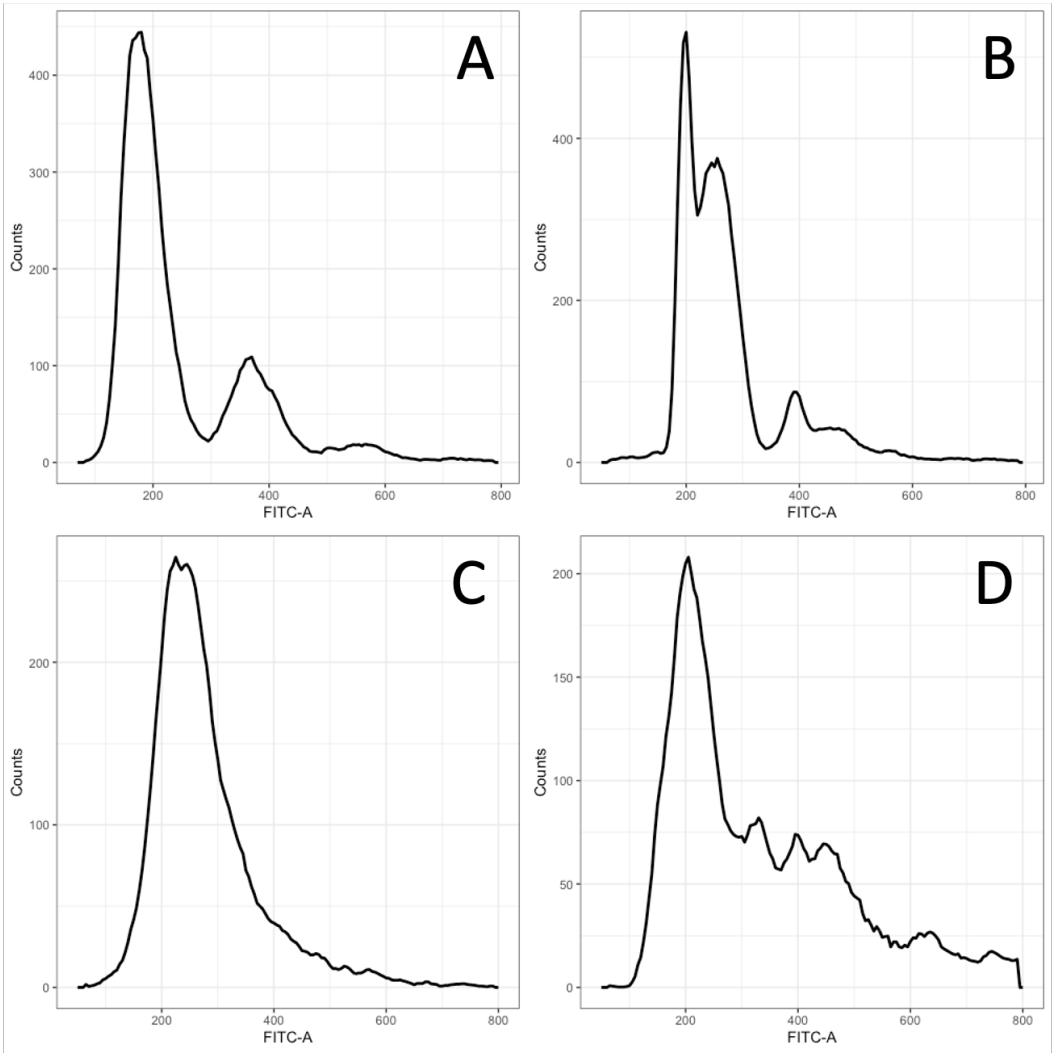

Figure 2.2: Examples of different shapes and sizes of flow cytometry peaks. A) Some samples have very obvious tall peaks, B) some have multiple tall G1 peaks and small G2 peaks, C) some will only have a single distinguishable peak, D) some have odd shapes. We sought to find an algorithm that can best work with data that varies in shape and sizes.

of subpopulations and some with the same number of subpopulation would yield a different optimal number of clusters (Figure 2.3).

We next tried to find the optimal number of clusters using the gap statistic method (Tibshirani et al. (2001)). The gap statistic method calculates the logarithm within-sum-of-squares (WSS) and compares it to the average WSS from simulated datasets created by Monte Carlo sampling. We iterated through $k = 1, \cdots, 15$, and for different numbers of simulated datasets (B). This method took a significant amount of time to run. We ran the gap statistics method on our testing dataset with different B values. When B = 1, creating one simulated dataset, it took 54 seconds, when B = 5 it took 3.49 minutes to run, when B = 10 it took 8.13 minutes to run, and when B = 17, it took 17.64 minutes to run for a single sample. This process has to iterate over many samples, with many datasets containing over 100 samples. Since the gap statistic used significant computational resources we did not use this method.

We next tried DBSCAN, a density-based unsupervised clustering algorithm. DBSCAN requires an input for *epsilon*, the radius of the neighborhood, and *minimum points*, which is the minimum number of points required in the neighborhood (Ester et al. (1996)). Unlike K-means, DBSCAN does not require a specified number of clusters.

Similar to K-means, we got inconsistent results with samples with the sample that have the same number of subpopulation using an epsilon value of $10, \cdots, 40$ and a minimum points value of $2, \cdots, 5$. In addition, DBSCAN was not able to identify a single peak per cluster. We applied DBSCAN on different samples from our testing dataset and the algorithm would categorize

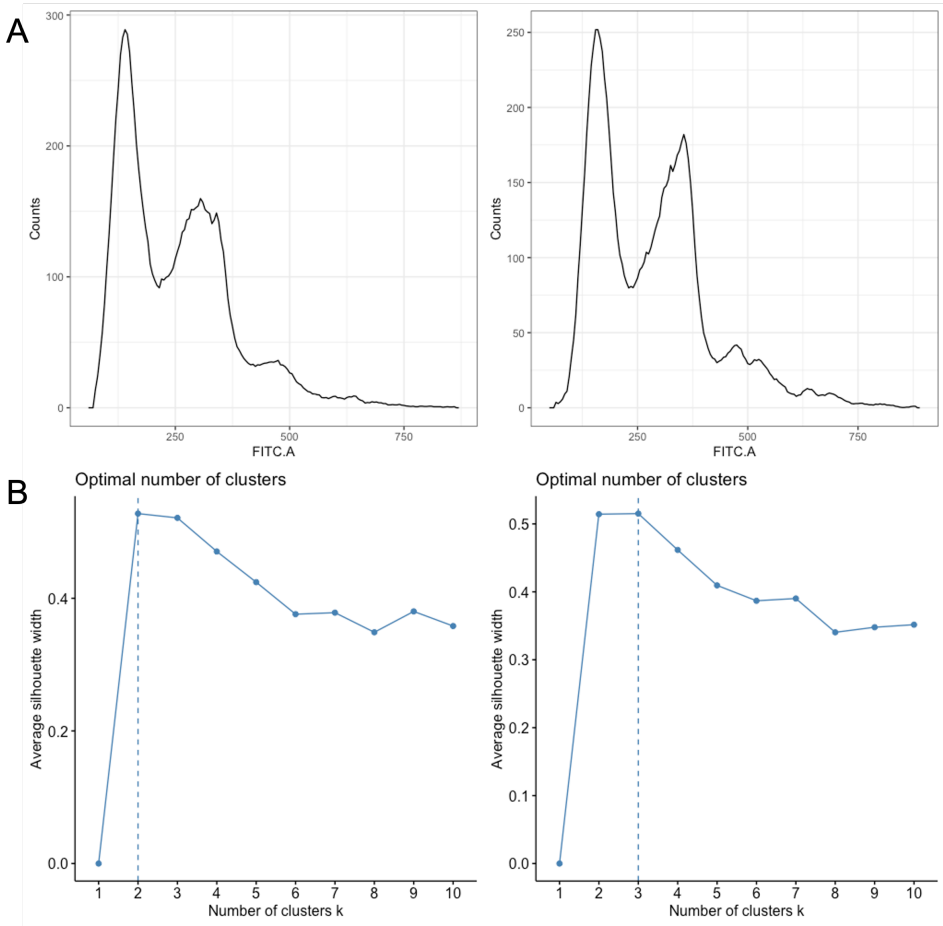

Figure 2.3: Finding the optimal number of clusters using the silhouette method. A) We have two samples that are similar in appearance. They both have two distinct peaks that correspond to the G1 and G2 peaks of a single population and a small G1+G2 doublet peak. B) The silhouette method found two different values of optimal number of clusters for the two similar samples.

the entire samples (i.e., the entire set of 10 000 cells) within a sample as one big cluster and noise (Figure 2.4).

Since these machine learning methods did not work as we needed them to,

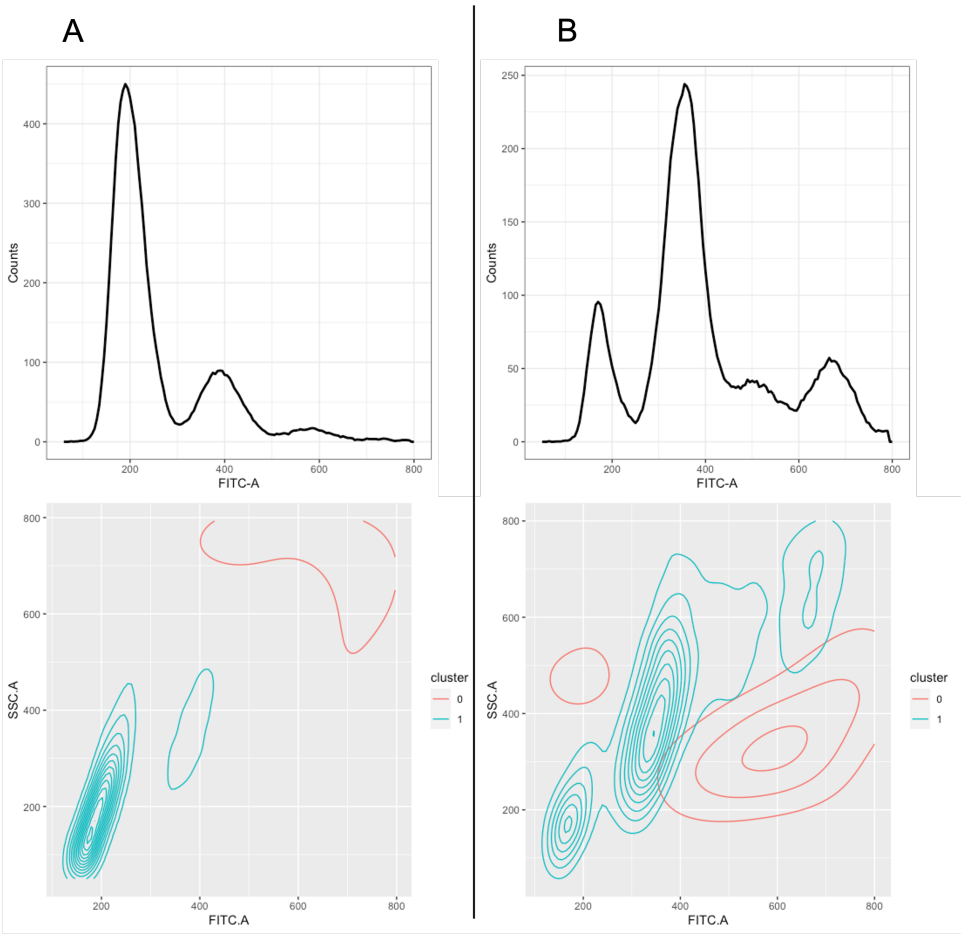

Figure 2.4: DBSCAN example for two samples that contain a different number of peaks. A) From a sample with two distinct clusters (G1 and G2 peaks from a single population) DBSCAN identified only a single cluster (cluster 1), with the remainder of cells classified as noise (denoted as cluster 0). B) A sample with three distinct clusters (G1 and G2 peak for two subpopulations, where the G2 peak for the first population is at the same place as the G1 peak of the second) DBSCAN identified only a single cluster (cluster 1), with the remainder of cells classified as noise (denoted as cluster 0). In both examples, DBSCAN identified the full sample as a single cluster and was not able to distinguish between peaks.

we switched tactics entirely to explore peak detection algorithms. We used peak detection algorithms to identify local maximums in the univariate fluorescence intensity data. The local maxima is defined as the tallest point in a given interval:

> f is said to have a local maximum point at the point a, if there exist some $\varepsilon > 0$ such that $f(a) \geq f(x)$ for all x in the interval $[a - \varepsilon, a + \varepsilon]$

We used the package *scorepeak* that was developed to identify true peaks from false peaks created by noise in times series data (Ochi (2019)). The function we used is called `detect_localmaxima()` which detects local maxima in a given width for univariate data (Ochi (2019)). The peaks identified in `detect_localmaxima()` depend on a chosen boundary condition. There are three boundary conditions:

The reflecting boundary condition considers the neighboring values to the right and to the left of each value of x_i in a given width (size = n) to decide if there is a peak.

$$T = x_{i-n}, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_n$$

The periodic boundary condition considers the neighboring value to the right of x_i in a given width (size = n) to decide if there is a peak.

$$T = x_i, x_{i+1}, \cdots, x_{n-1}, x_n$$

The discarding boundary condition which discards the data points at the beginning and end of the numeric vector in a given width.

There are three mandatory parameters in the function: *data*, which is the numeric vector being analyzed; w, the width of data considered in the peak (referred to as window size), which must be an odd number greater than two; and *boundary*, which is the boundary condition used to identify peaks, we used the "reflecting" boundary. We started by using the smallest possible value for a sliding window ($w = 3$). Samples have peaks in all shapes and sizes, and by picking the smallest window this ensures that each peak will be identified. We quickly realized that the flexibility built in to this method provided a promising avenue to pursue. This function was able to differentiate between the G1 and G2 peaks which was crucial since samples with an odd number of peaks share peaks between subpoulations. We used the function `detect_localmaxima()` to identify all G1, G2 and doublet populations (i.e., peaks) in *ploidyPeaks*.

2.2.4 Smoothing

The DNA fluorescence intensity data is smoothed to eliminate small peak spikes caused by stochasticity in the cell population analyzed and/or machine use abnormalities such as cell doublets(often passing through the laser simultaneously). Two methods were considered for data smoothing: a density curve and the moving mean. The numerical vector of densities was obtained from the histogram of the numerical vector of the fluorescent dye value measured for each cell in each sample (`hist()` function, `hist()$density`). The density curve smooths the data using kernel density estimates to remove any unwanted

spikes. We did not use this method, as we found the moving mean method was easier to adjust the granularity of the data.

The moving mean, also called a rolling mean, calculates a series of means over a certain interval. The moving mean eliminates stochastic small spikes in the dataset, since the spike will be averaged out by the neighboring values. Different smoothing levels can be applied within the function to specify the desired granularity. We used the `rollmean()` function from the *zoo* package(Zeileis et al. (2022)). This function has four mandatory parameters (Table 2.1). *ploidyPeaks* starts with a smoothing level of $k = 11$ to analyze samples that clearly contain single populations. Once the 'easy' samples are analyzed, a more granular smoothing level of $k = 5$ was set to capture peaks that are close to each other.

Table 2.1: Parameter description and option used in *rollmean()* function from the *zoo* package.

Parameter	Parameter used
x	Numerical vector of peak heights
k	Using different k's throughout the package: 11, 5, 4, 3
Align	Determines how to apply the sliding window: • align = "center" will average the values from (x-k, x+k) with x being the current data point
Fill	Fill = 0

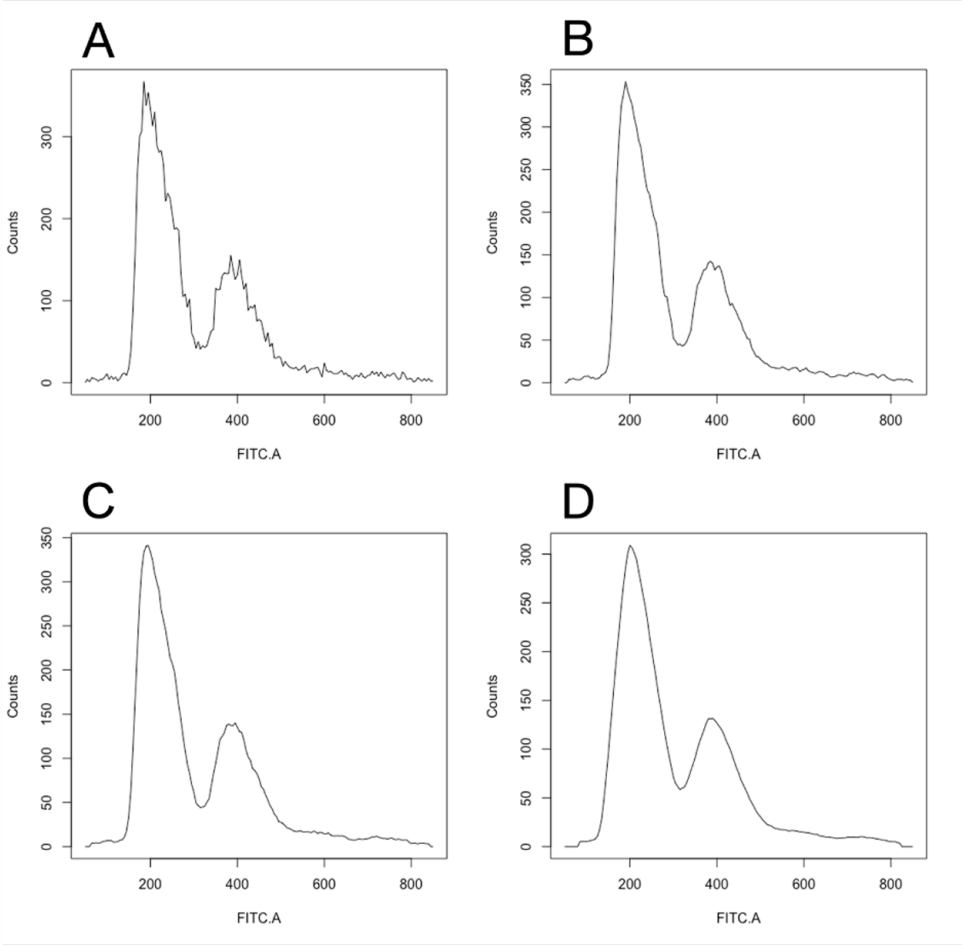

Figure 2.5: An example of the outcome of the *SmoothData()* function. A) The raw data is read into the function, to eliminate small spikes from being identified as peaks. In the examples here, the smoothing factor applied was B) 3, C) 5, and D) 11.

2.2.5 True Peaks

The algorithm identifies peaks that represent both peaks of interest (i.e., peaks that correspond to true G1 and G2 populations) as well as peaks that arise in

the data due to spikes or abnormalities caused by external factors that and result in a local maximum in the histogram. The function `findTruePeaks()` was written to identify the true peaks, and to remove spurious peaks caused by stochasticity. It is initially assumed that each local maximum is a true peak. Testing datasets were used to find the average distance between peaks identified in samples with known subpopulation peaks that were close together, and determined the distance that would still allow both of these peaks to be classified as true. If two clusters are too close together (I.e., below that distance), they are combined and only one peak is identified per cluster (Figure 2.6).

2.2.6 Single vs. Multiple Populations

Samples are divided into two categories by the package: those that consist of a single population (i.e., one G1 and one G2 peak) and those that contain multiple subpopulations. A single wrapper function called `flowPeakDetection()` is used initially to discriminate between suspected single vs. multiple population samples, and then to iterate through a series of functions that progressively use increasingly granular data to identify smaller peaks in multiple population samples.

peakAlgorithm1

The first peak detection algorithm, `peakAlgorithm1()` identifies samples that very clearly contain only two peaks. The algorithm smooths the data by a

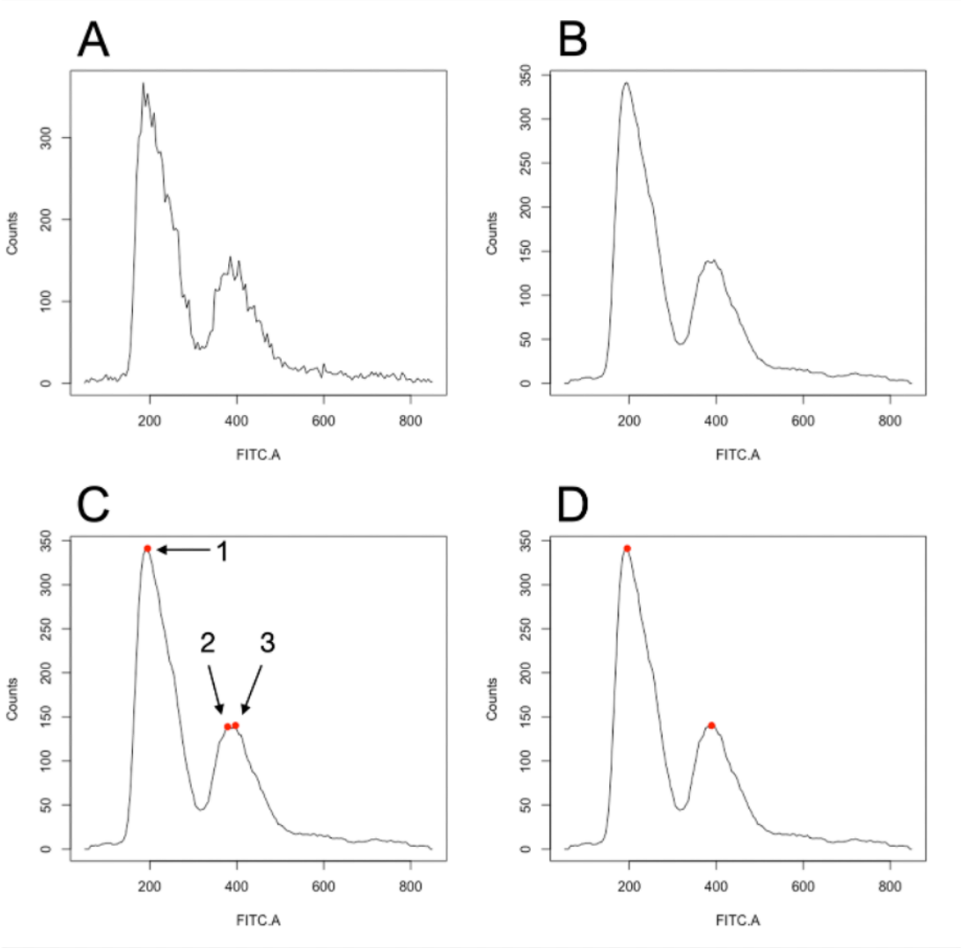

Figure 2.6: The raw data that gets read into the algorithm. B) The data is smoothed by a smoothing factor of five .C) The peak detection algorithm identified three peaks. D) The *findTruePeaks()* function will group together peaks that are in close proximity (peaks 2 & 3).

factor of 11 and identifies a single G1 and G2 pairing. If there are more than two peaks identified, the algorithm will assume that the population is composed of the G1 and G2 pairing that is furthermost left. The function calculates the distance between the G1 and G2 peaks and marks a boundary at a G2 peak plus half this distance (*popBoundary*, Figure 2.7). The percent of cells outside the boundary is calculated. If it is within the specified threshold, the population is marked as a single population, otherwise it is passed to the next function. The default threshold was set at 8.5% through testing on multiple samples with a known number of peaks, though the user can select their own threshold.

If the percent of cells outside the boundary exceeds the threshold, the population is flagged as likely containing multiple populations and is passed to the next algorithm, `peakAlgorithm2()` (Figure 6, C and B), which assumes there are multiple subpopulations and seeks to identify the cells associated with each G1 and G2 pairs using `findTruePeaks()` and `findPairs()`. The algorithm repeats the peak detection step with a smoothing factor of 5, which maintains more granular data (i.e., more peaks).

Peak matching

Once peaks have been identified, the matching G1 and G2 peaks for each subpopulations are sought. If the total number of identified peaks is even, this indicates that each subpopulation will have unique G1 and G2 peaks, and the number of subpopulations is the total number of peaks divided by two.

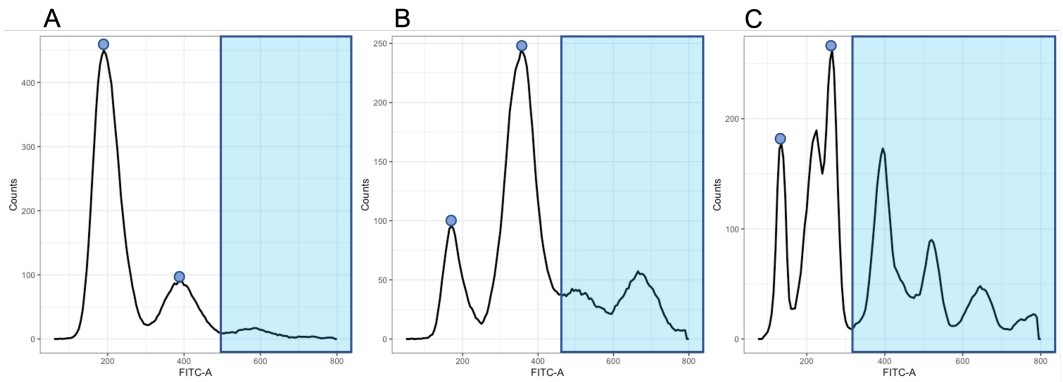

Figure 2.7: *PeakAlgorithm1()* calculates the percentage of cells to the right of the G2 peak (i.e., beyond *popBoundary*). The blue dots represent the G1 and G2 peaks used to find *popBoundary* and the blue shaded area represents the cells found in *popBoundary* A) An example of data from a single population sample, with only 5.5% of cells beyond *popBoundary*. B) An example of a population that would be passed on to *PeakAlgorithm2()*, where the percent of cells (32.4%) exceeds the set 8.5% threshold. C) An example of a population with multiple populations. The algorithm assigns a G1 and G2 pairing from the peaks that are farthest to the left as the focal population. This sample would also be passed on to *PeakAlgorithm2()*, as the percent of cells in *popBoundary* (49.10%) exceeds the set threshold.

Conversely, if the total number of peaks is odd, this indicates that the G1 of one subpopulation is the same as the G2 of another subpopulation. For example, in the case of a population that contains diploid, 2n, and tetraploid, 4n, subpopulations, the diploid G2 peak is the same as the tetraploid G1, and the number of subpopulations is the total number of peaks plus one divided by two (Figure 2.4 B).

The subpopulation pairing method we use builds off the Watson Pragmatic (WP) and the Dean-Jett-Fox (DJF) cell cycle algorithms. They both identify

the tallest fluorescence intensity peak and assign that to be the G1 mean. Watson et al. (1987) identifies location of the highest point near 1.75×G1 mean to find the G2 mean, while Fox (1980) identifies the location of the highest point near the 2×G1 mean to find the G2 mean. We created the function `findPairs()` to match peaks when there are subpopulations. `findPairs()` assumes that each peak remaining after `findTruePeaks()` is a possible G1 and will try to find their subpopulation pairing. The range for a matching G2 peak from each possible G1 peak is calculated by creating lower and upper bounds based of WP and DJF.

We set the initial lower limit to 1.75×G1 peak and an upper limit of $2.2 \times G1$ peak. For each possible G1 peak, `findPairs()` looks at all peaks identified within the specified range; if there is more than one peak identified, it will assign the tallest of those identified peaks to be the associated G2 peak. If there is no possible G2 peak within the range, the algorithm will assume that this possible G1 peak does not have a subpopulation pairing, and will be removed from the list of potential subpopulations at this time.

Throughout the package, different lower and upper bounds are applied. We start with (1.75×G1, 2.2×G1) in the first and second peak detection algorithm, then as we deal with the less obvious peaks in the samples, we widen the range to (1.7×G1, 2.3×G1) in 3rd and 4th peak detection function and eventually (1.5×G1, 2.5×G1) for the last peak detection function.

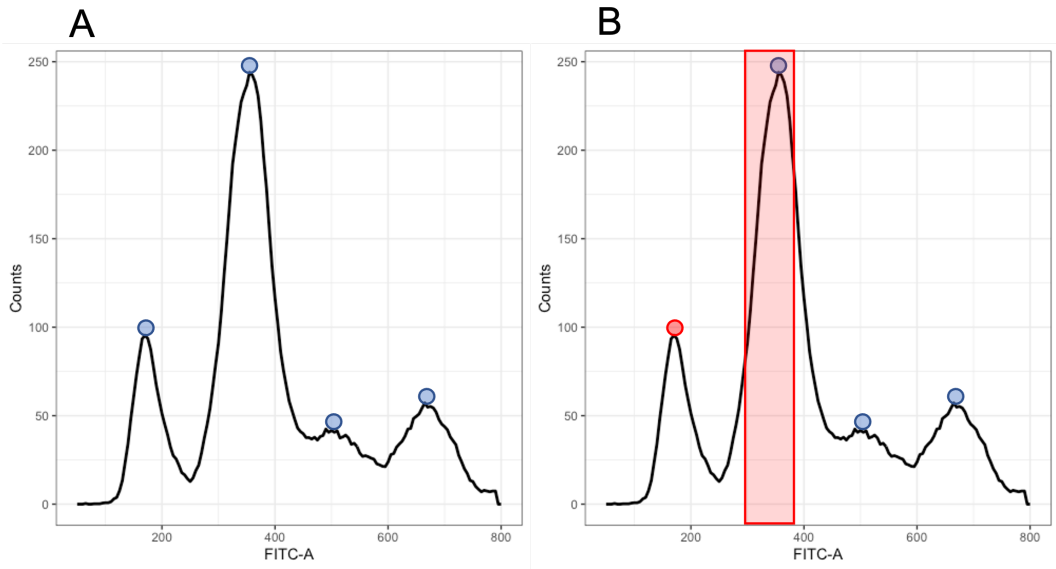

Figure 2.8: Subpopulation pairings example from *findPairs()*. In A) The peak detection algorithm and *findTruePeaks()* identified four peaks located at x=170, 355, 490 and 675, which are marked by blue dots. B) Considering the first peak at x = 170 (red dot), *findPairs()* will calculate lower and upper bounds and look to see if a peak falls within those limits. For *peakAlgorithm2()* the lower bound is set at 1.75×170 = 297.5, and the upper bound is set at 2.2×170 = 374, which is represented as the red shaded area. The peak located at x = 355 falls within those limits, so *findPairs()* assigns these two peaks as a subpopulation pair.

Doublets

Previous workflows and programs required the user to gate out doublets prior to the analysis. Since our goal was to analyze data free of bias and with minimal user input, we decided not to require that the doublets be manually removed. Instead, we created a function to identify them, and to provide the user with information on the height of the doublet peak(s) and mean. To our knowledge, the existence of doublets within samples has previously been ignored, yet they

may reflect something of biological interest since they are likely related to cell surface properties such as 'sticky' proteins that cause cells to stay together even through mechanical disruption.

After the subpopulations have been identified by our algorithms, the number of cells associated with each subpopulation is determined. If the proportion of unused cells is large, the algorithm will assume some subpopulation peaks were not identified, and the sample will get passed to the next peak detection algorithm. Since the doublet peak does not belong to any subpopulation, doublets increase the number of unused cells and when there are large doublet populations the peak algorithm will fail (Figure 2.9). The function `doubletCheck()` identifies doublet peaks in each sample. It looks for peaks around the G1+G2 range as well as the G2+G2 range. If there is a peak identified in that range, the sample will save the information on the location and height of the doublet.

There is no way to discriminate between the a G1+G1 doublet and a true G2 peak. The G2+G2 doublet can be mistaken as the G2 of a subpopulation and discriminating between these two types of peak is difficult. We have made some criteria for a subpopulation G2 based on samples with known ploidy. If the sample identifies a subpopulation G2 but that peak is smaller than the G1+G2 doublet, then it is a G2+G2 doublet(Figure 2.9 B). Additional rules include the G2 must be within a certain the height of their G1 peak (called *maxDoubletHeight*), the height changes from higher to lower going through the peak algorithm function. *maxDoubletHeight* is a parameter in `flowPeakDetection()`

that can be adjusted by the user. If the user notices their data has higher doublets/lower G2 peaks, then they can adjust *maxDoubletHeight* to better analyse their data.

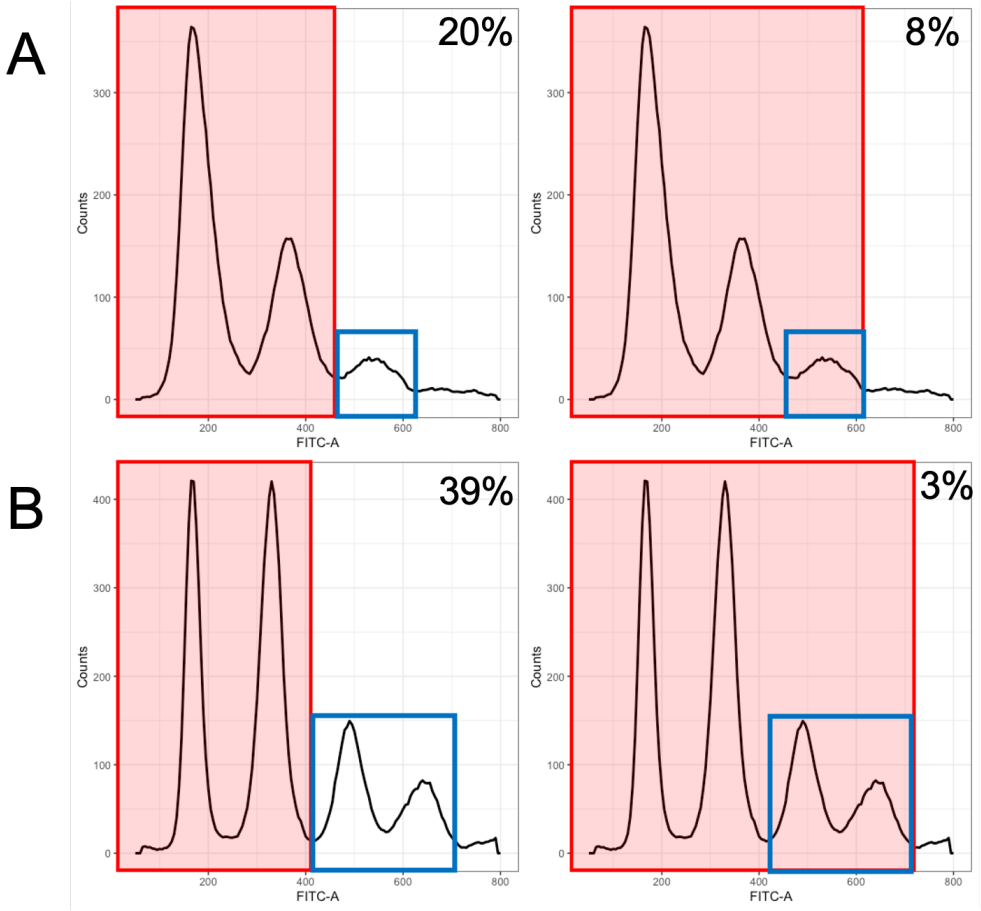

Figure 2.9: Identification of doublets. A) Before specifying that there is a G1+G2 doublet peak, the sample would fail all peak detection branching points since the proportion of cells being used is 80%. After considering doublets, 92% of cells are considered accounted for. B) Before identifying the G1+G2 and the G2+G2 doublet, this sample had only 61% of cells used; once we included the doublet peaks, 97% of cells were accounted for.

peakAlgorithm 2, 3, 4 and 5

Once all peaks are identified, the proportion of cells that were used is calculated ("*cellPerc*", Figure 2.10). If *cellPerc* is under 86%, that sample will be passed on to the next algorithm, `peakAlgorithm3()`. The *cellPerc* threshold was determined by examining control populations composed of known *Candida albicans* single populations; it can be adjusted by the user.

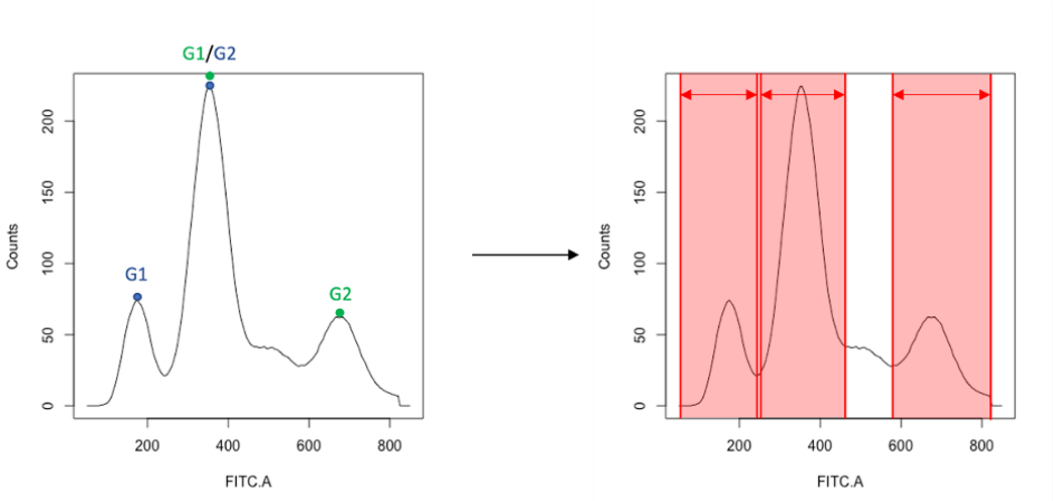

Figure 2.10: The left most panel is sample with two subpopulations. The blue dots represent the G1 and G2 mean of the first subpopulation, and the green dot represents the mean of the second subpopulation. The percent of cells being used in the subpopulation pairings is represented in the right panel in the shaded red area, while the area in white ate the cells not being used. *cellPerc* is calculated by the number of cells in the red area divided by the total number of cells $\times$ 100.

Populations that do not fall within the threshold from `peakAlgorithm2()` are passed to `peakAlgorithm3()`. The interval where a G2 peaks is looked for is widened to (1.7, 2.3), and `findTruePeaks()` and `findPairs()` are applied

as before. If the sample's *cellPerc* still does not pass the threshold of 86% in `peakAlgorithm3()`,then it gets passed on to `peakAlgorithm4()`, where smoothing level is reduced to four, and `findTruePeaks()` and `findPairs()` are applied as before. Samples that are flagged at the end of `peakAlgorithm4()` are annotated in the final data table to investigate by the user for visual inspection, as there is less confidence in the identified peaks. They are passed on to the final function, `peakAlgorithm5()`, which has a smoothing level of three and the G1 and G2 peak interval is widened to (1.4, 2.5) in a final attempt to identify possible pairs.

2.2.7 Confidence term using residual standard error (RSE)

Following peak detection analysis, the Dean-Jett-Fox cell cycle algorithm is applied to each sample using the identified peak means and heights (from each subpopulation, where appropriate) to find a residual standard error (RSE) using nonlinear least squares models.Three possible DJF models are run for each sample; a single population model, a doublet model and a subpopulation model.

The DJF cell cycle algorithm assumes the peaks are Gaussian distributed with

$$F_i(x) = \frac{N_i}{\sqrt{2\pi\sigma_i^2}} exp[-\frac{(x - x_i)^2}{2\sigma_i^2}] \tag{2.2}$$

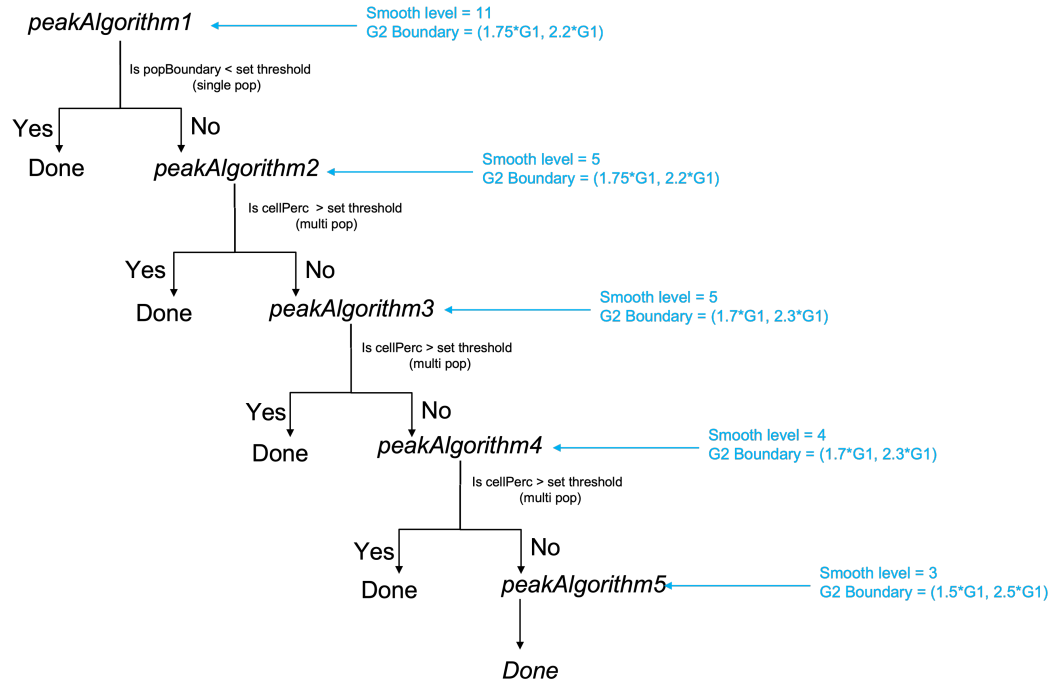

Figure 2.11: Workflow of wrapper function *flowPeakDetection()*. All samples go through to *peakAlgorithm1()*, which uses a smoothing level of 11 and a G2 bound of $(1.75 \times G1, 2.2 \times G1)$, if the *popBoundary* falls within the set threshold the sample is done being analyzed, if it does not then it gets passed on to *peakAlgorithm2()*. The process in peakAlgorithm2() is similar, but uses more granular data (smoothing level of 5) and calculates *cellPerc*. If *cellPerc* exceeds the threshold it gets passed on to the *peakAlgorithm3()*. *peakAlgorithm3()* widens the bounds for G2 to $(1.75 \times G1, 2.2 \times G1)$, and *peakAlgorithm4()* uses a smoothing level of 4. *peakAlgorithm5()* uses the most granular data (smoothing level of 3) and the widest bounds for G2 $(1.5 \times G1, 2.5 \times G1)$.

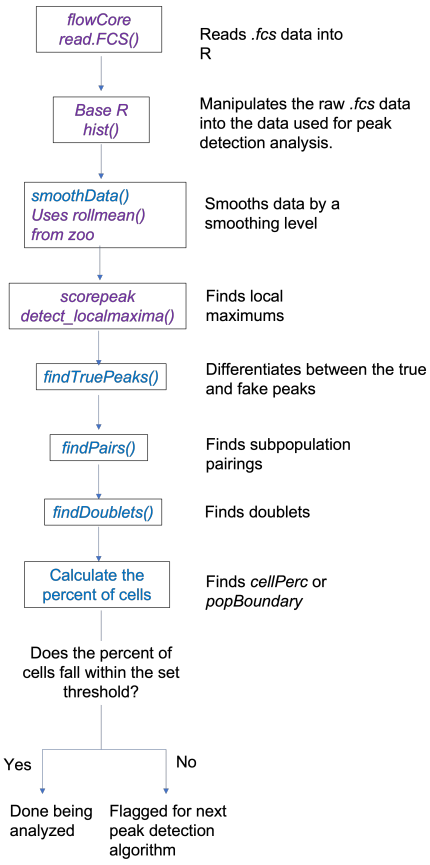

Figure 2.12: The workflow of the *peakAlgorithm* functions. The existing functions are highlighted in purple, while we created the ones highlighted in blue. The data gets read into R using the existing *read.FCS()* function from *flowCore*. The data gets passed through the hist function from base R to manipulate the raw data into the data that will be used for the peak detection algorithms. The data then gets smoothed using the *rollmean()* function from the *zoo* package in a function called *smoothData()*. We find the local maximums with *scorepeak*'s *detect_localmaxima()* and remove the 'fake' peaks with *findTruePeaks()*. For the samples' that are identified as a possible multi population will use *findPairs()* to identify possible subpopulations. Then the algorithm will find doublets (if any). Lastly, we find the percent of cells within a specific area, i.e., *popBoundary* and *cellPerc*, and if that area falls within the set threshold, the sample is done being analyzed or else if it gets passed to the next algorithm.

Where the x_i's are represented by the means identified in our peak algorithm. The σ_i's are calculated as the width of the distribution at 60%(Watson et al. (1987)) of each mean, i.e., if the G1 mean is 170, the standard deviation will be calculated where the height is $0.6 \times 170 = 102$ on either side of the peak. The N_i's are calculated by summing the cells within $(-\sigma_i,\sigma_i)$.

Three DJF models are run for each sample. The first model assumes each sample is a single population.

G1:

$$F_{1_1}(x) = \frac{N_1}{\sqrt{2\pi\sigma_1^2}} exp[-\frac{(x-x_1)^2}{2\sigma_1^2}] \tag{2.3}$$

G2:

$$F_{2_1}(x) = \frac{N_2}{\sqrt{2\pi\sigma_2^2}} exp[-\frac{(x-x_2)^2}{2\sigma_2^2}] \tag{2.4}$$

S-phase between G1 and G2:

$$F_{s1}(x) = \sum_{j=x_1}^{x_2} f(x_j) \frac{1}{\sqrt{2\pi\sigma_1^2 \frac{x_j}{x_1}}} exp[-\frac{(x-x_j)^2}{2(\sigma_1 \frac{x_j}{x_1})^2}] \tag{2.5}$$

And the model 1 is $F_{M1}(x) = F_{1_1}(x) + F_{2_1}(x) + F_{s1}(x)$

The second model will adds doublet peaks to the single population, if any were identified by `doubletCheck()`.

G1+G2 doublet:

$$F_{d1}(x) = \frac{N_{d1}}{\sqrt{2\pi\sigma_{d1}^2}} exp[-\frac{(x - x_{d1})^2}{2\sigma_{d1}^2}] \tag{2.6}$$

S phase between G2 and G1+G2 doublet:

$$F_{s_d1}(x) = \sum_{j=x_2}^{x_{d1}} f(x_j) \frac{1}{\sqrt{2\pi\sigma_2^2 \frac{x_j}{x_2}}} exp[-\frac{(x - x_j)^2}{2(\sigma_2 \frac{x_j}{x_2})^2}] \tag{2.7}$$

G2+G2 doublet:

$$F_{d2}(x) = \frac{N_{d2}}{\sqrt{2\pi\sigma_{d2}^2}} exp[-\frac{(x - x_{d2})^2}{2\sigma_{d2}^2}] \tag{2.8}$$

S phase between G1+G2 doublet and G2+G2 doublet:

$$F_{s_d2}(x) = \sum_{j=x_{d1}}^{x_{d2}} f(x_j) \frac{1}{\sqrt{2\pi\sigma_{d1}^2 \frac{x_j}{x_{d1}}}} exp[-\frac{(x - x_j)^2}{2(\sigma_{d1} \frac{x_j}{x_{d1}})^2}] \tag{2.9}$$

And our model 2 is

$$F_{M2}(x) = F_{1_1}(x) + F_{2_1}(x) + F_s(x) + F_{d1}(x) + F_{s_d1}(x) + F_{d2}(x) + F_{s_d2}(x)$$

If appropriate (i.e., if `flowPeakDectection()` identified any subpopulations), the third model fits each subpopulation identified by the algorithm.

$$F_{multi}(x) = \sum_{i=1}^{n} F_{1_i}(x) + F_{2_i}(x) + F_{s_i}(x) \tag{2.10}$$

Where

$$f(x_j) = A + Bx_j + Cx_j^2 \tag{2.11}$$

A, B and C are estimated using Nonlinear least squares

The final RSE for the sample is the lowest of the fitted models. Users can use the RSE as a confidence term to see how well the algorithm performed relative to their visual observation or among other samples.

Example

Consider the sample in Figure 2.14 A), where the algorithm found four peaks, which were assigned to two subpopulations without any doublets (Table 2.2).

Table 2.2: Subpopulations identified in Example 1

	G1 mean	G2 mean	G1 height	G2 height
Subpopulation 1	160	310	138.4	146.8
Subpopulation 2	235	455	476.2	123.4

Model 1:

The first model assumes we have a single population, therefore even when more than two peaks are identified, we only use the peaks associated with the subpopulation which contains the lowest DNA fluorescence intensity, $G1_1$ and $G2_1$. The standard deviations are estimated to be located where the x-values correspond to 60% of the height of the peak. For the $G1_1$ peak, the standard deviation is the x-value that corresponds to where the height is

$60\% \times 138.4 = 83.04$. We find the x-value in the interval $(0,G1_1)$ at $y = 83.04$, which we will call x_L and find x_R in the interval $(G1_1, G2_1)$ (Figure 2.13). Once we find the nearest x-value for $y = 83.04$ on either side of the peak, we average the distance between x_L and x_R, this will become the standard deviation. The standard deviation for the $G1_1$ peak in the example is 27.5 (σ_1). Similarly, for the $G2_1$ peak, the standard deviation will be located where the y-axis is $60\% \times 146.8 = 88.08$, and the standard deviation is 22.5 (σ_2). N_i is estimated by summing the cells within ($-\sigma_i$,σ_i) of the peak. Our $N1_1$ is be the number of cells from $x = (130 - 27.5, 130 + 27.5)$, which in this case is $N1_1 = 1145$. The number of cells within our $G2_1$ peak is estimated to be $N2_1 = 1233$.

The first model curves are:

G1: $F_{1_1}(x) = \frac{1145}{\sqrt{2\pi \times 27.5}} exp[-\frac{(x-160)^2}{2\times 27.5}]$

G2: $F_{2_1}(x) = \frac{1233}{\sqrt{2\pi \times 22.5}} exp[-\frac{(x-310)^2}{2\times 22.5}]$

S-phase between G1 and G2:

$F_{s_1}(x) = \sum_{j=160}^{310} f(x_j) \frac{1}{\sqrt{2\pi\sigma_1^2 \frac{x_j}{160}}} exp[-\frac{(x-x_j)^2}{2(27.5\frac{x_j}{160})^2}]$

$F_{M1}(x) = F_{1_1}(x) + F_{2_1}(x) + F_{s_1}(x)$

Model 2:

Since no doublets were identified, there will not be a model 2 output for this sample.

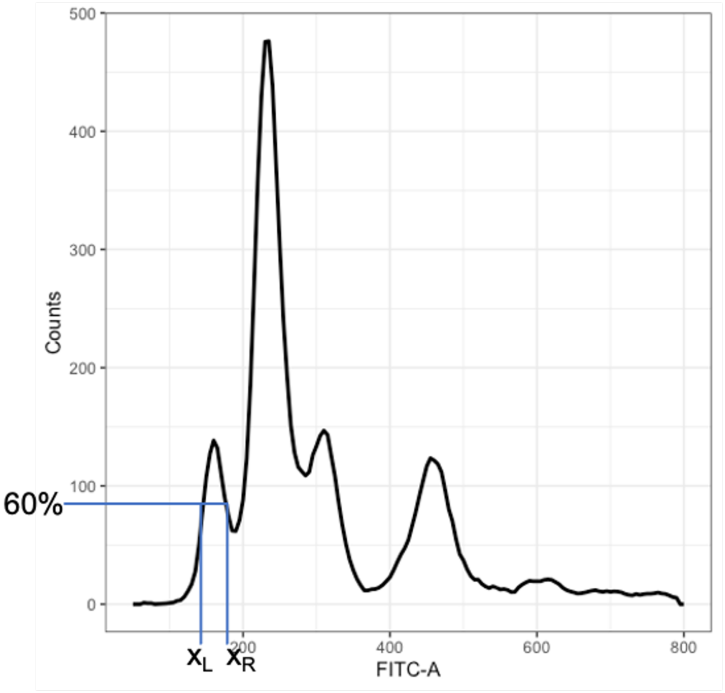

Figure 2.13: The standard deviations are estimated to be located at a height of 60% of the height of the peaks on either side of the peak. For the $G1_1$ peak ($x = 160$), the standard deviation will be located where the y-axis is $60\% \times 138.4 = 83.04$. We find the x-value value in the interval $(0,G1_1)$ at $y = 83.04$, which we will call x_L and find x_R in the interval $(G1_1, G2_1)$.

Model 3:

The third model considers every peak identified: $G1_1$, $G1_2$, $G2_1$ and $G2_2$. Our $F_{1_1}(x)$, $F_{2_1}(x)$ and $F_{s_1}(x)$ will be the same as in the first model. We need to find the standard deviation and the number of cells within each peak for the second population. This is done with the same process as for the first population. The standard deviation for the $G1_2$ peak will be located where the y-axis is $60\% \times 476.2 = 285.72$, and the standard deviation is 17.5 (σ_1). The standard

deviation for $G2_2$ will be located where the y-axis is $60\% \times 123.4 = 74.04$, with a standard deviation of 25 (σ_2). The estimated number of cells within each peak is 3123 ($N1_2$) and 1113 ($N2_2$) for $G1_2$ and $G2_2$ respectively.

$G1_1$: $F_{1_1}(x) = \frac{1145}{\sqrt{2\pi \times 27.5}} exp[-\frac{(x-160)^2}{2\times 27.5}]$

$G2_1$: $F_{2_1}(x) = \frac{1233}{\sqrt{2\pi \times 22.5}} exp[-\frac{(x-310)^2}{2\times 22.5}]$

S-phase between $G1_1$ and $G2_1$:

$F_{s_1}(x) = \sum_{j=160}^{310} f(x_j) \frac{1}{\sqrt{2\pi \times 27.5 \frac{x_j}{160}}} exp[-\frac{(x-x_j)^2}{2(27.5\frac{x_j}{160})^2}]$

$G1_2$: $F_{1_2}(x) = \frac{3123}{\sqrt{2\pi \times 17.5}} exp[-\frac{(x-235)^2}{2\times 17.5}]$

$G2_2$: $F_{2_2}(x) = \frac{1113}{\sqrt{2\pi \times 25}} exp[-\frac{(x-455)^2}{2\times 25}]$

S-phase between $G1_1$ and $G2_1$:

$F_{s_2}(x) = \sum_{j=235}^{455} f(x_j) \frac{1}{\sqrt{2\pi \times 17.5 \frac{x_j}{235}}} exp[-\frac{(x-x_j)^2}{2(17.5\frac{x_j}{235})^2}]$

$F_{M3}(x) = F_{1_1}(x) + F_{2_1}(x) + F_{s_1}(x) + F_{1_2}(x) + F_{2_2}(x) + F_{s_2}(x)$

Model 1 had an RSE of 59.35 (Figure 2.14 B), while model 3 had an RSE of 11.85 (Figure 2.14 C). The final RSE for this sample is 11.85, which is model 3, giving us confidence that *ploidyPeaks* correctly identified the peaks and subpopulations for this sample.

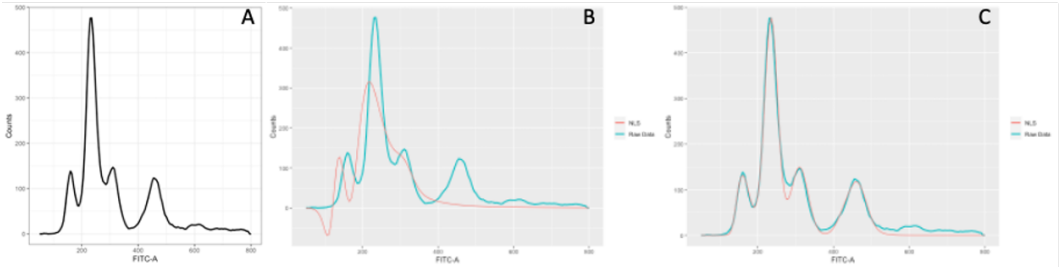

Figure 2.14: Model Comparison. A) The sample we are fitting a model to, *flowPeakDetection()* identified two subpopulations with 4 peaks located at x=160, 235, 310, 455. B) The model 1 output. The green line is The trace of the sample of interest and the pink line is our model 1 attempting to fit the first population, resulting in a RSE of 59.35. C) The model 3 output. We can see the two lines almost overlap, resulting in a RSE of 11.85

2.2.8 Data Table Outputs

Throughout the analysis different data outputs are provided. These outputs will be saved in the parent directory where the folder with the *.fcs* files are, this is also where the *.fcs* of the gated data will be saved is saved (Figure 2.15). When the user gates the data using `rectGateFlowFrame()` or `rectGateFlowSet()`, a *.csv* file, named *percentOfCellsGatedOut*, will be saved with information on the percent of cells that have been gated out for each sample. *percentOfCellsGatedOut* can be used as an indication of proper gating. I.e., if there were no cells gated out this is an indication that the user needs to adjust his gating bounds.

The main data output from the package is a *.csv* file called *ploidyPeaksOutput*, this is saved in a folder called 'analysis', in the same location as the *percentOfCellsGatedOut*. *ploidyPeaksOutput.csv* contains all of the information

from the peak detection analysis. Each row is a different sample. Column information is peak means (G1 and G2 peaks), peak heights, and includes the number of identified subpopulations. The user can also specify if they want similar information on doublets, whether to include an indicator of whether the sample should be investigated further by the user, which peak algorithm was used to analyze each sample, and the residual standard error value from the models.

2.2.9 Visualization

Visualizing the data is an important part of the workflow that adds another layer of confidence to the analysis as the user will be able to compare the results from a sample from the *ploidyPeaksOutput.csv* to the graphs. Throughout the package there is an opportunity to save graphs from each major analysis step. During the gating process, the user has the option to save graphs that compare pre-gated and gated data. These are bivariate density plots with the DNA fluorescence intensity on the x-axis and SSC-A on the y-axis. The user also has the option to save the histograms with the DJF cell cycle algorithm applied to the sample. These graphs have the DNA fluorescence intensity on the x-axis and the cell height on the y-axis. These graphs will have two lines, one for the trace of the data and one for the model. In addition, the function `flowLineGraph()` was created for the purpose of visualization. This function permits the users to plot as many samples as they would like on top of each

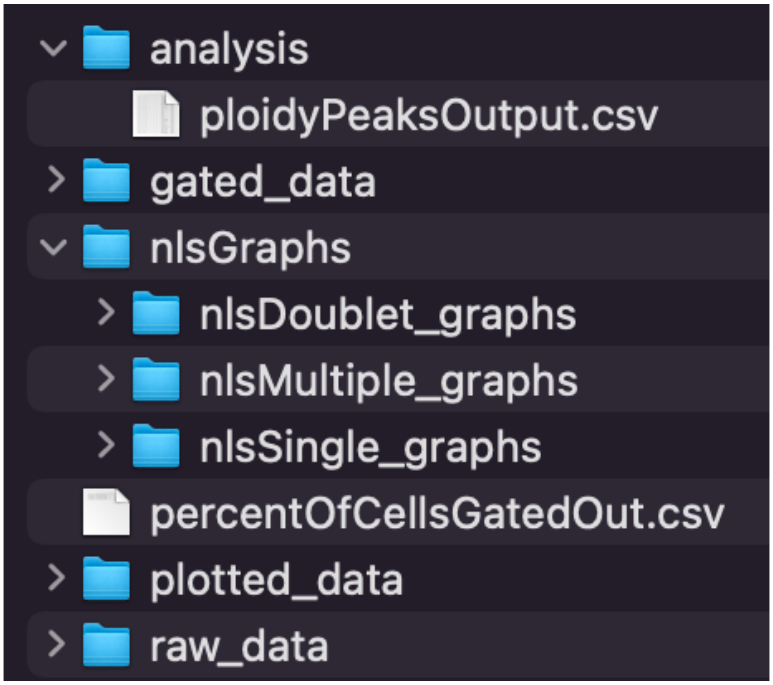

Figure 2.15: The folder structure created by *ploidyPeaks*. The user creates the initial folder that contains the data they want to analyze (we suggest raw_data in the package vignette), this location of this folder will be the main directory for the package. When using the gating functions, the package creates a folder for the gated data *.fcs* files (gated_data), a folder for the plots comparing the gated and the raw data (plotted_data), and a file called *percentOfCellsGatedOut.csv* with information on the percent of cells gated out for each sample. When analyzing the data with *flowPeakDetection()*, a file called *ploidyPeaksOutput.csv* will be created in a folder called analysis. Additionally, a folder will be created to store the graphs for the models (nlsGraphs) with sub folders for each model (nlsSingle_graphs, nlsDoublet_graphs, nlsMultiple_graphs).

other on the same panel. Each trace indicates a different sample. If desired, a control population (specified by the user) can be plotted with a black line for comparison. The other samples are represented by colourful lines.

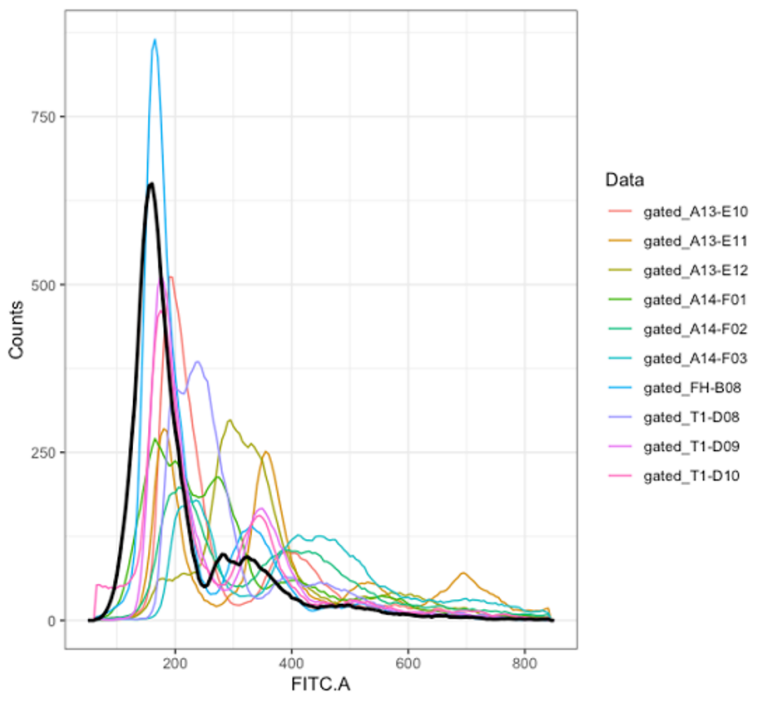

Figure 2.16: A sample of a plot with ten samples (colourful lines) and one control sample (black line) produced by *flowLineGraph()*.

2.3 Testing Data

Samples that contain a known number of populations were used to assist with building the package. We used 480 ancestral strain which are known to be single populations and we created 250 mock data with known subpopulations. We created three types of mock data: a mixture of a haploids (populations containing a single set of chromosomes) and diploids (populations containing two homologous sets of chromosomes; the haploid G2 and diploid G1 are overlapping), a mixture of diploids and tetraploids (populations containing four

complete sets of chromosomes; diploid G2 and tetraploid G1 are overlapping), and a mixture of diploids and any ploidy greater then a diploid (no peaks overlapping). We generated mock data from different samples that contained ∼10000 cells, by randomly sampling cells from each population at different specified percentages. For example, for the haploid and diploid mixture, we sampled from 10% haploid:90% diploid to 90% haploid:10% diploid, changing in increments of 10%. We did this for each type of mock data.

The first threshold required was *singleThreshold* in `peakAlgorithm1()`, which finds a single G1 and G2 pairing and calculates the proportion of cells to the right of the G2 peak, *popBoundary*. We sought a threshold for *popBoundary* where we would be confident that there is likely only a single population. We wanted to ensure the value was conservative, since once a sample is identified as having a single population, it does not undergo further analysis. The default threshold is currently set at 8.5%, meaning if there are more than 8.5% of cells above *popBoundary*, we suspect there are multiple populations or doublets and further analysis is required. *singleThreshold* was chosen by looking at the percent of cells beyond *popBoundary* of the 480 ancestral samples (Figure 2.17). We tested different threshold values between 6%-11% by increments of 0.5 on our mock data that had subpopulations. We wanted to find the lowest *singleThreshold* value before sample with known subpopulations would be 'identified' as single. A mock data sample with a 20%diploid:80%tetraploid (had subpopulations) mixture had a *singleThreshold* greater than 8.5% but

lower than 9%. For this reason we created the *singleThreshold* at 8.5%.

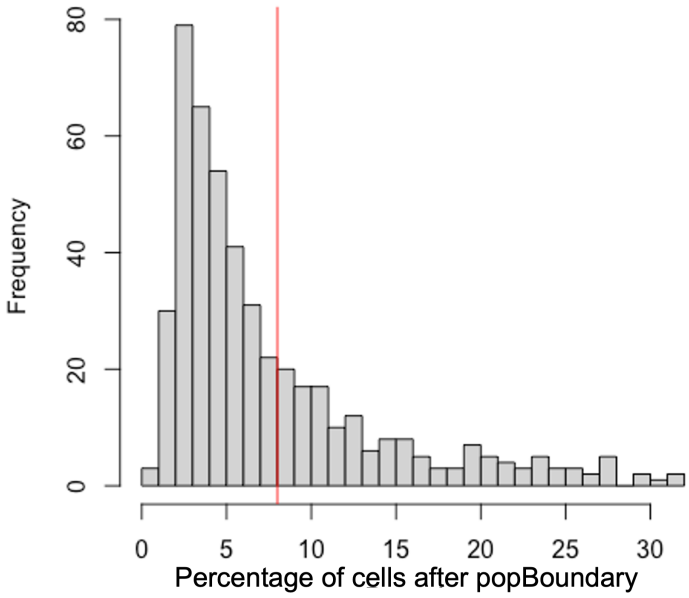

Figure 2.17: The distribution of *singleThreshold* in single populations. The first threshold required was *singleThreshold* in *peakAlgorithm1()*, which finds a single G1 and G2 pairing and calculates the proportion of cells to the right of the G2 peak, *popBoundary*. We looked at the percent of cells after the *popBoundary* for the 480 ancestral samples. The current threshold is set at 8.5%.

The second threshold (*usedCellsThreshold*) required was for the percentage of cells used by the subpopulations, *cellPerc*, in `peakAlgorithm2()`, `peakAlgorithm3()`, `peakAlgorithm4()` and `peakAlgorithm5()`. If the percentage of cells being used is below a certain threshold, we are not confident

that all the peaks have been identified and the sample will be passed on to the next peak algorithm for further analysis. We used mock data containing a mix of single and multiple populations that range in location, size and shape of the peaks. The first threshold is looking for unused cells (a low number) while the second is looking for the used cells (a high number). To find a value for this threshold we considered the samples that were passed by `peakAlgorithm1()` to be further analyzed. Those samples were passed through `peakAlgorithm2()` and we manually went through each sample and visually we concluded if samples were correctly analyzed (have their proper G1 and G2 pairings) or not. If they were properly analyzed, we looked at the *cellPerc* of each sample and obtained a distribution (Figure 2.18). We then looked at the *cellPerc* of samples that likely did not have all of their peaks identified in `peakAlgorithm2()`. For example, if a sample has a G2 located at 2.3×G1 peak, then `findPairs()` will not be able to find a G2 peak in `peakAlgorithm2()` since the G2 bound is set to (1.75×G1, 2.2×G1). That peak will count towards unused cells and their *cellPerc* will be lower. There were a few samples that had small peaks that were missed in `peakAlgorithm2()` and the highest value of *cellPerc* from those samples was 85%. Because of this we set our threshold to 86%.

In addition to the thresholds, the testing samples were used in every decision of the package. The smoothing of the data relied on the known populations, to make sure we do not over smooth and miss any peaks and knowing when to make the data more granular. The `findtruePeaks()` function used the testing

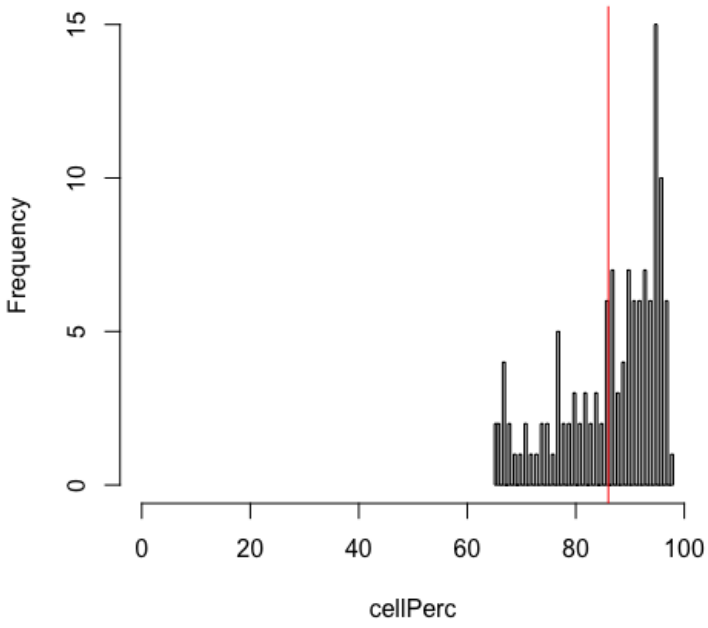

Figure 2.18: The distribution of proportion of cells (*cellPerc*) of samples that have been correctly analyzed after the second peak detection algorithm. We created 250 mock data samples with known ploidy, *peakAlgorithm1()* identified 62 samples as single populations, the remaining 188 samples was analyzed by *peakAlgorithm2()* and 119 samples were correctly analyzed.The current threshold is set to 86%

datasets to find the average distance between peaks to be able to distinguish between true and false peaks. `findPairs()` used the datasets to establish G2 bounds. We used WP and DJF's definition for the first bound, but quickly found that not all G2 peaks fell within the $(1.75 \times G1, 2.2 \times G1)$ bound. We relied on the different shapes of the peaks, different ploidy of the samples to model our package *ploidyPeaks* after to try and identify all the possible

variation of subpopulations.

Chapter 3

Use case: Genome Size Variation During Evolution

Here we present a use case analysis. We will be using the package to analyze two different DNA fluorescence intensity datasets from two different experiments that were performed on different machines with cells stained with different fluorescent dyes. These datasets were evolved in the presence of a drug to see what would happen over time to their genome size. The goal is to demonstrate a walk through on the package capabilities and demonstrate how to interpret the output. We have two different experiments that were performed on different machines with cells stained with different fluorescent dyes.

3.1 Project 1

DNA fluorescence intensity data was acquired from *Candida albicans* populations that were evolved in the lab for 50 generations through batch culture

transfers (termed "experimental evolution") in the presence of the azole antifungal drug posaconazole (Kukurudz et al. (2022)). Azole drugs are known to cause genome instability (Gerstein and Berman (2020);Kukurudz et al. (2022)). The dataset we examine here includes 40 samples from the beginning of the evolution experiment (referred to as ancestral, or t0 strains, which are known to be single populations) and 65 samples from the end of the experiment (evolved, t5 strains) that contain an unknown number of subpopulations. The experiment was initiated from five different strain backgrounds (A4, A10, A12, A17, A18).

Prior to starting the analysis, we put the *.fcs* files in a folder, raw_data. We apply a gate to the whole dataset with `rectGateFlowSet()` which creates a folder called gated_data which includes the *.fcs* of the gated data, a folder called plotted_data which includes a comparison plot for each sample, and a file with the pecentage of cells gated out for each sample called *percentOfCellsGatedOut.csv*. `rectGateFlowSet()` has a few parameters we need to assign: *flowDir* is the directory file path to the data we want to gate, *xVariable* is the slot used to store the fluorescence intensity data used for this experiment, which will be the variable plotted on the x-axis, *yVariable* is the parameter on the y-axis, *xMinValue* is the x value to create the left side bound to the gate, *yMinValue* is the y value to create the lower bound to the gate, *xMaxValue* is the x value to create the right side bound to the gate, and *yMaxValue* is the y value to create the upper bound to the gate. These four bounds create a

rectangle on the graph. Lastly we have the *savePlot* parameter which will save the comparison plots if it is set to true. Different machines save intensity data in different scales/magnitudes so although there are defaults in some cases users will need to alter based on their machine settings.

```
rectGateFlowSet(
  flowDir = "flow_data/project1/raw_data",
  xVariable = "FL1-A",
  yVariable = "SSC-A",
  xMinValue = 100000,
  yMinValue = 100000,
  xMaxValue = 900000,
  yMaxValue = 900000,
  savePlot =  TRUE
)
```

Once our data is gated we run `flowPeakDetection()` to start our core analysis: *xVariable* is the slot used to store the fluorescence intensity data, *flowDir* is the directory file path to our gated data, doublets are always looked for *doublet* = TRUE records the information in *ploidyPeaksOutput*, *saveGraph* = TRUE will save the model graphs. *singleThreshold* is our threshold in *peakAlgorithm1* to separate single populations is set to our default value of 8.5% and *usedCellsThreshold* our threshold in the other peak detection algortihms that looks at the percentage of cells used, which is also set to our default

value of 86%. By setting *maxDoubletHeight* = NA, `flowPeakDetection()` will calculates the maximum doublet height based on the height of the G1 peak in the sample, the user has the option to specify their own height (value on the y-axis). *subsetDs* is only used if we are running a subset of the samples. For example, if a user wants to rerun the analysis for three specific samples, they can do so by feeding a vector of the sample names to subsetDs.

```
flowPeakDetection(
  xVariable = "FL1-A",
  flowDir = "flow_data/project1/gated_data",
  doublet = TRUE,
  saveGraph = TRUE,
  singleThreshold = 8.5,
  usedCellsThreshold = 86,
  maxDoubletHeight = NA,
  subsetDs = NA
)
```

Looking at the results from *ploidyPeaksOutput* in our analysis folder, we see that the majority of samples (100) were classified as single populations, while `flowPeakDetection()` identified five samples with subpopulations and flagged four additional samples for the user to investigate.

After looking at the data table, the next step is visual inspection. Plotting the five subpopulation samples (Figure 3.1) in some cases it is not obvious

whether there is truly a second population, or simply a tall doublet (i.e., Figure Figure 3.1, sample A3_12). This sample was flagged for review because their single model RSE (5.93) was lower than their multiple model RSE (8.47). This provides an indicator that this sample might not have subpopulations and the G2 of the second population is likely a doublet, but ultimately it is up to the user to decide based on their expertise and familiarity with typical traces from their samples. The other samples have distinct G2 peaks for second populations.

Table 3.1: *ploidyPeaksOutput* for the samples with subpopulations in project 1. Sample A3_12 has a lower single model RSE (5.93) than their multiple model RSE (8.47) which indicates this should be a single population and the G2_2 peak is likely a doublet. The other four samples have lower multiple model RSE and all have an overlapping G1/G2 peak for their second population.

Data	G1_1	G1_2	G2_1	G2_2	G1Count_1	G1Count_2	G2Count_1	G2Count_2	singleRSE	multipleRSE
A17_6	200000	385000	385000	830000	95.75	100.75	100.75	35	11.77	9.68
A3_12	355000	400000	810000	845000	37.25	76	23.25	24.5	5.93	8.47
A8_10	205000	435000	435000	825000	32.4	57.4	57.4	86	24.03	11.17
A2_6	235000	435000	435000	870000	84.8	141.2	141.2	63	20.12	4.63
A4_5	220000	470000	470000	870000	27.75	34	34	87.5	25.56	7.85

Three additional samples were flagged for investigation. Visual inspection (Figure 3.2) suggested that two samples have subpopulations and were missed because they have very small G1 peaks in the first population and a very tall G2 in the second population (Figure 3.2 C & D). One samples was likely flagged due to the algorithm failing to account for a G2+G2 doublet peak (Figure 3.2 B).

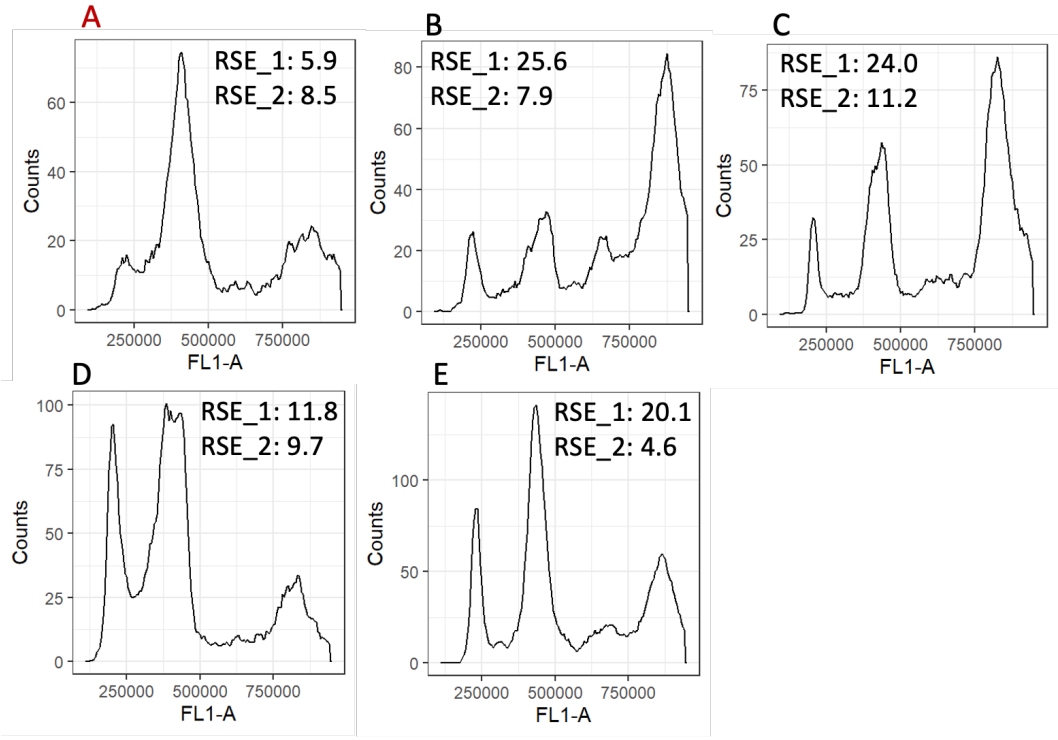

Figure 3.1: Visualization of project 1 samples with subpopulations. Samples with red letters are flagged for review. A) The subpopulations for A3_12 are located at G1 = 355000 for the first population, an overlapping G1/G2 = 355000 and a G2 = 830000 for the second population. B) A4_5 has a subpopulation located at G1 = 220000 for the first population, an overlapping G1/G2 = 470000 and a G2 = 870000 for the second population. C) A8_10's subpopulations are located at G1 = 205000, an overlapping G1/G2 = 435000 and at G1 = 435000 and G2 = 825000. D)A17_6 subpopulations are located at G1 = 200000, an overlapping G1/G2 = 385000 and G2 = 830000 for the second population. E) The sample A2_6 has their first subpopulation at G1 = 235000 for the first population, overlapping G1/G2 peak = 435000, and G2 = 870000.

At the conclusion of the automated and manual (visualization) analysis, six samples out of 105 were found to contain two populations. Four of those

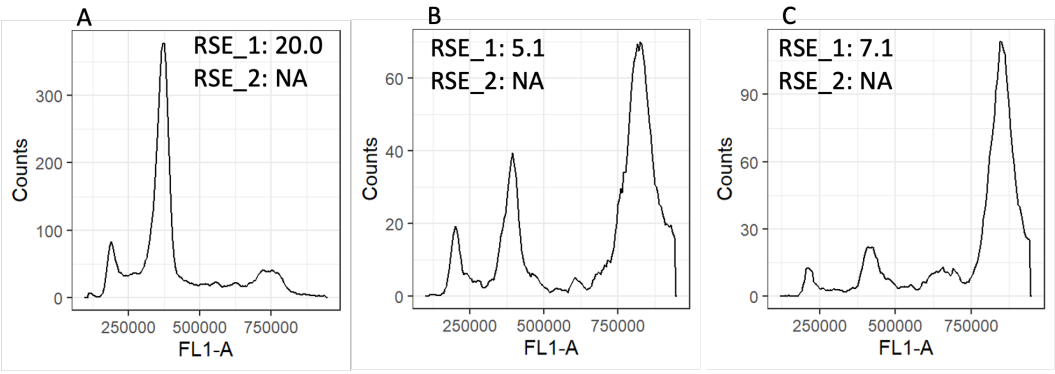

Figure 3.2: Three samples flagged for review in project 1. Upon visual investigating we found two more samples with subpopulations (B,C).

samples were identified by the package. The other two were flagged to be investigated due to their *cellPerc* being below the threshold (62.5% and 83%) at the last peak detection algorithm. From visual inspection we found that they likely also contain multiple subpopulations. By contrast, we found that one sample that was initially categorized as having subpopulations more likely is a single population with a large G2+G2 doublet peak. The remaining 99 samples are consistent with single populations, as the algorithm identified.

Having gone through and visually inspected subpopulation and flagged samples, we can compare the G1 mean from the ancestral strains (t0) to the evolved strains (t5) to see how much variation in genome size arose following azole drug evolution (Figure 3.3). Evolved replicates from strain backgrounds A10 and A12 had the least variation and all but one evolved replicate in each case retained the ancestral (diploid) genome size. Two evolved A18 replicates had an increased G1 mean, which is clearly evident in both the numerical results

and the raw traces. Evolved replicates from A4 and A17 exhibited considerable variation, and samples from both included subpopulations, pointing to increased genome instability in these genetic backgrounds.

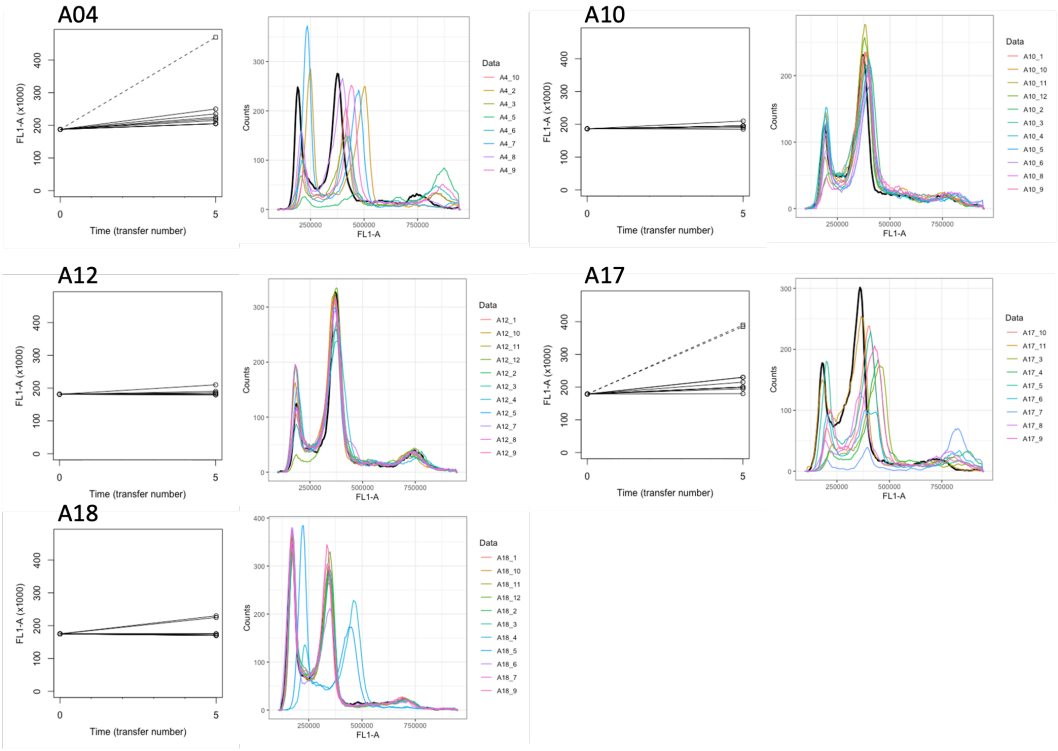

Figure 3.3: Genome size variation in replicates evolved from five strain backgrounds evolved to the drug posaconazole for 50 generations. Panels on the left for each plot demonstrates the variation in G1 mean from the ancestral strains (t0) to the evolved strains (t5). The dashed line and square indicates the samples with subpopulations. Panels on the right are display traces of the evolved replicates in each strain. The black line in each panel is an ancestral replicate for that given strain.

3.2 Project 2

The second project was very similar conceptually to the first: replicate populations of *C. albicans* were evolved to the azole drug fluconazole for 100 generations of experimental evolution in the lab. The data from this project was larger: 12 replicates were evolved from 20 different strain backgrounds for a total of 240 ancestral and evolved samples. The ancestral strains (t0) were measured twice ("t0_1" and "t0_2"), and in addition to the final evolved strains (t10), data was also collected at t8.

We start off the same way as in project 1, by gating our data place in a folder called raw_data using `rectGateFlowSet()`. The data were collected on a different machine than Project 1, so the only difference is the name of the *xVariable*. The code remains the same for each evolution timepoint with the exception of changing *flowDir* to reflect the location of the data (the .fcs files of samples from different timepoints were in different folders).

```
rectGateFlowSet(
  flowDir = "flow_data/project2/t0_1/raw_data",
  xVariable = "FITC-A",
  yVariable = "SSC-A",
  xMinValue = 100,
  yMinValue = 100,
  xMaxValue = 800,
  yMaxValue = 800,
  savePlot =  TRUE
)
```

Then we ran the analysis on the gated datasets using the function `flowPeakDetection()`.

```
flowPeakDetection(
  xVariable = "FITC-A",
  flowDir = "flow_data/project2/t0_1/gated_data",
  doublet = TRUE,
  saveGraph = TRUE,
  singleThreshold = 8.5,
  usedCellsThreshold = 86,
  maxDoubletHeight = NA,
  subsetDs = NA
)
```

As in project 1, we expected all t0 samples to be single populations. From the 240 t0_1 samples, the algorithm correctly classified 235 as single populations. While five samples were analyzed as having two subpopulations. Upon visual examination, it was clear that samples A08-04 and A08-09 did indeed have clear peaks consistent with two subpopulations (Figure 3.4 A & B). Two additional samples, A11-07 and A16-05, have a peak to the right of the G2 peak (Figure 3.4 C & D), and the data in these samples are much messier than typical for single populations, likely due to a poor stain/sample preparation. The algorithm flagged for review the sample A06-03 (Figure 3.4 E), in the final output we see that it has a single population RSE of 3.62 and a multiple population RSE of 26.67, meaning the sample should not be classified as having sub populations. The function also flagged this sample as the RSE 26.67 was considered an outlier. After looking at the graphs, we noticed that some of the doublets peaks were higher than typical (likely due to something related to the cell preps) which is likely how it got assessed as having multiple populations. One additional sample, A02-10, was flagged for further investigation for having a high RSE (Figure 3.4 F), the algorithm missed a small G2+G2 doublet. When we reran the function with the parameter *maxDoubletHeight* increased to 65, which increases the maximum height that a doublet can have to 65. If a peak is lower than 65, it will not be considered a doublet. Only two samples, A08-04 and A08-09, retained identified subpopulations.

Table 3.2: *ploidyPeaksOutput* for the samples in t0_1 with subpopulations in t0. Sample A06-03 has a much lower single model RSE (3.62) than their multiple model RSE (26.67) which gives the user an indication this should be a single population and the G2_2 peak is actually a doublet. A08-04 (Figure 3.4 A) was flagged for investigation because the RSE from both models are very similar, but upon visual inspection we concluded it has subpopulations. The other three samples have lower multiple model RSE but only A08-09 (Figure 3.4 B) had true subpopulations, with the others having a doublet mistaken as a G2.

Data	G1_1	G1_2	G2_1	G2_2	G1Count_1	G1Count_2	G2Count_1	G2Count_2	singleRSE	multipleRSE
A08-04	125	160	270	340	311.25	361.25	115	123.5	13.88	13.18
A08-09	140	165	295	335	287.4	302.2	127.4	159.8	15.30	12.38
A06-03	136	196	284	346	145.25	73.25	43.5	38	3.62	26.67
A11-07	175	240	360	515	194.2	64.4	195	64.8	22.09	11.97
A16-05	175	295	365	520	133.8	89.4	167.2	64.4	24.48	9.72

The t0_2 dataset contains the same 240 samples that were prepped and run at a separate time. Only one sample was identified as including subpopulations, no other samples were flagged for investigation. The sample A14-6 was analyzed as having two populations, with an RSE of 18.25 (Figure 3.5). Similar to samples A11-07 and A16-05 from the previous analysis, the algorithm classified the excess clump of cells as a G2 of the second population. I would consider this peak to be classified as doublets instead of G2 subpopulation peak.

Looking at both t0 datasets together, two samples from the first dataset (A08-04 and A08-09) had data were consistent with subpopulations, but these subpopulations were not present in the second dataset. This suggests that either a variant subpopulation arose during the prep that affected genome size, or (more likely) there was cross-contamination when the samples were grown up or prepped between different samples.

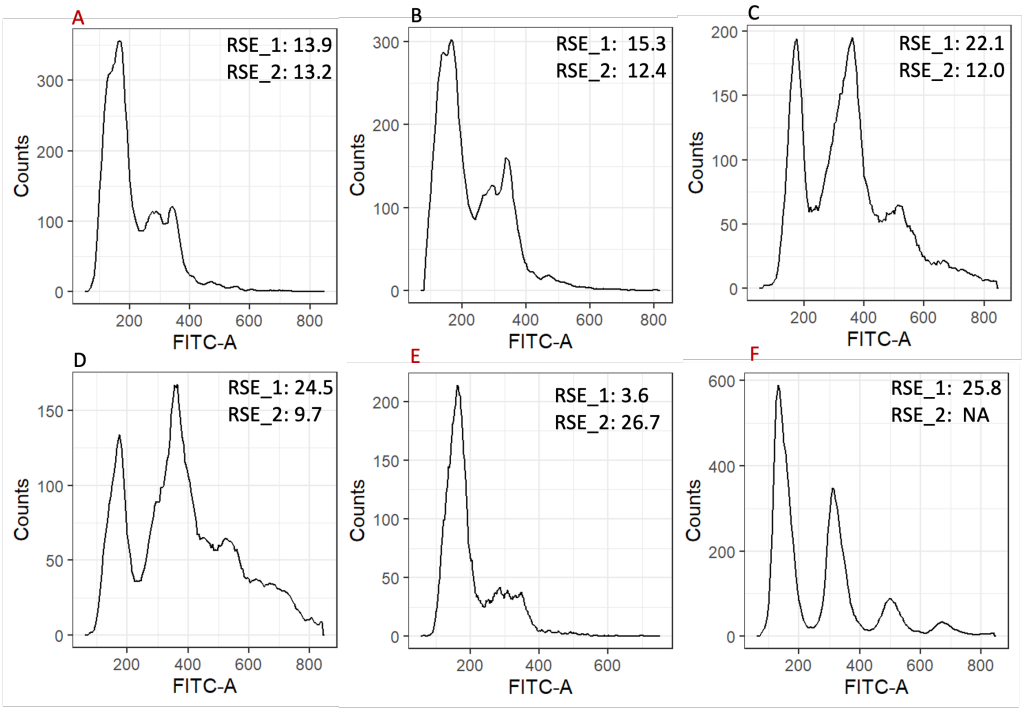

Figure 3.4: Visualization of samples in t0_1, those in red were flagged for review. Five samples were initially analyzed as having two subpopulations(A,B,C,D), upon visual inspection we concluded only two samples have subpopulations (A & B).

Samples that were evolved through eight transfers (80 generations) were much more variable: 22 samples were identified with having subpopulations (most consistent with two, two samples with three), while 15 were flagged for review (including seven of the 22 with subpopulations) (Figure 3.6). Of the seven samples with subpopulation that were flagged for investigation, five have tall doublets (Figure 3.6 A, B, C, D & E) that were likely mis-classified as second populations. The sample A8-6 and A10-5 had five identified peaks and

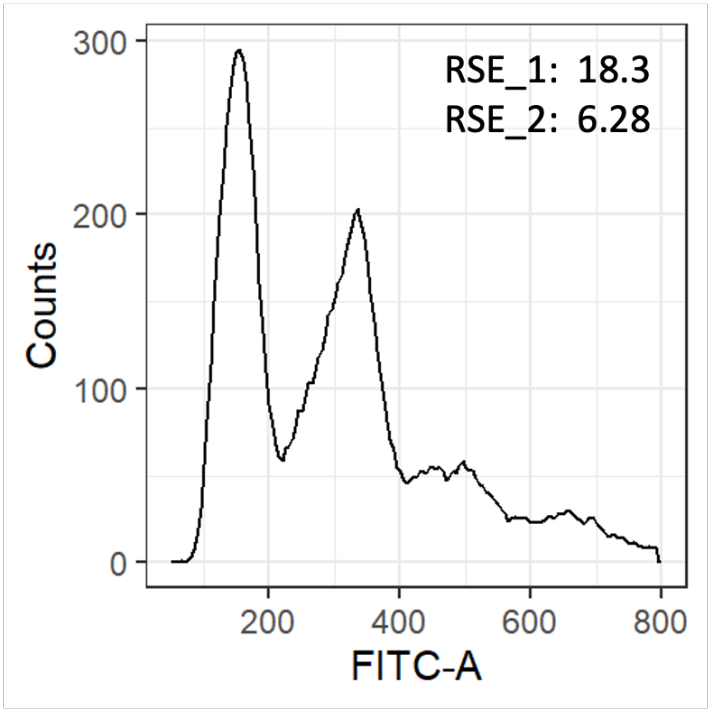

Figure 3.5: Visualization of t0_2 samples with subpopulations. The sample A14-6 was identified as including subpopulations, with the peaks located at G1 = 150/G2 = 330 and G1 = 330/G2 = 490. The second G2 should be classified as a doublet.

two subpopulations were identified (Figure 3.6 F & G), yet looking at the traces the data are also consistent with three populations. A10-5 has their tallest peak at the G1+G2 location. The algorithm identified two subpopulations, but the data is also consistent with three subpopulations, with the first population located at G1 = 145 and G2 = 285, the second population located at G1 = 285 and G2 = 550, and the last population located at G1 = 440 and G2 = 710. Similarly, the sample A8-6 has the possibility to have three subpopulations with the first population located at G1 = 135 and G2 = 270, the second population

located at G1 = 270 and G2 = 530, and the last population located at G1 = 400 and G2 = 655. This would be a case where the user would have to use their knowledge of the data to make an educated decision. All other subpopulations samples look good after manual inspection.

Table 3.3: *ploidyPeaksOutput* for the 22 samples with subpopulations in t8. Two samples identified three subpopulations while the other samples identified two subpopulations. The G1 means of the first subpopulation was located around 135-170, 265-315 for the second, and 265-270 for those with a third subpopulation. The G1Count columns are the heights of the G1 peaks, G2Count are the heights are the G2 peaks and G3Count are the heights of the G3 peaks. All samples have a lower RSE for the multiple model than for the single model indicating that they are likely, as the function indicated, samples with subpopulations.

Data	G1_1	G1_2	G1_3	G2_1	G2_2	G2_3	G1Count_1	G1Count_2	G1Count_3	G2Count_1	G2Count_2	G2Count_3	singleRSE	multipleRSE
A2_10	145	290	NA	290	555	NA	265	317.75	NA	317.75	80	NA	34.34	31.30
A3_4	160	245	NA	310	480	NA	178.2	267	NA	113.6	98.4	NA	29.51	13.72
A7_4	150	280	NA	280	525	NA	298.8	213.2	NA	213.2	86.4	NA	27.15	15.20
A7_12	150	290	NA	290	530	NA	246.4	265.2	NA	265.2	81.2	NA	24.69	9.24
A8_6	135	270	NA	270	530	NA	232.25	410.5	NA	410.5	148	NA	46.61	36.88
A10_4	170	245	NA	290	470	NA	102.75	201	NA	145	85.25	NA	34.60	13.34
A10_5	145	285	NA	285	550	NA	104.75	140.75	NA	140.75	66.5	NA	52.19	40.72
A10_6	150	185	NA	265	350	NA	127.75	98.5	NA	169.75	61.5	NA	12.02	6.30
A11_12	145	175	NA	270	340	NA	163.2	129	NA	111.2	79.6	NA	9.73	9.73
A13_1	160	245	NA	315	480	NA	427.75	306.75	NA	80.5	86.25	NA	26.26	9.89
A13_3	160	235	NA	310	455	NA	138.4	476.2	NA	146.8	123.4	NA	59.35	11.85
A13_6	160	265	NA	310	525	NA	233	193.8	NA	265.8	96.4	NA	37.61	11.81
A13_9	140	285	NA	285	545	NA	126.5	286.25	NA	286.25	87.5	NA	39.35	31.60
A13_11	145	285	NA	285	565	NA	395.75	345.5	NA	345.5	67	NA	30.11	22.47
A14_5	150	285	NA	285	560	NA	163.75	245.75	NA	245.75	86.5	NA	35.44	26.97
A14_7	170	195	NA	300	360	NA	215.8	213.4	NA	87.2	65.8	NA	18.04	10.50
A14_8	155	280	NA	280	555	NA	112.8	276	NA	276	118	NA	38.26	24.06
A16_11	160	245	290	290	435	560	647	397.2	541.2	541.2	166	158.2	75.21	16.04
A17_9	135	225	265	265	395	520	177.4	189.4	262	262	173	90	36.85	27.65
A17_11	145	270	NA	270	520	NA	387	275.75	NA	275.75	76	NA	27.04	26.83
A19_3	145	185	NA	270	420	NA	249.75	108.75	NA	216.75	72	NA	21.94	21.94
A19_6	150	235	NA	305	455	NA	262.4	315	NA	123.6	105.6	NA	34.20	10.16

Eight additional samples that were classified as single population were also flagged for review (Figure 3.7). It looks as though four have subpopulations (Figure 3.7 A, B, C & D): the algorithm missed the small G1 peak for the first subpopulation in A09-A5 and for the second subpopulation in A10-B9, and a small G2 peak in A14-F6 for the second population. Two samples that were

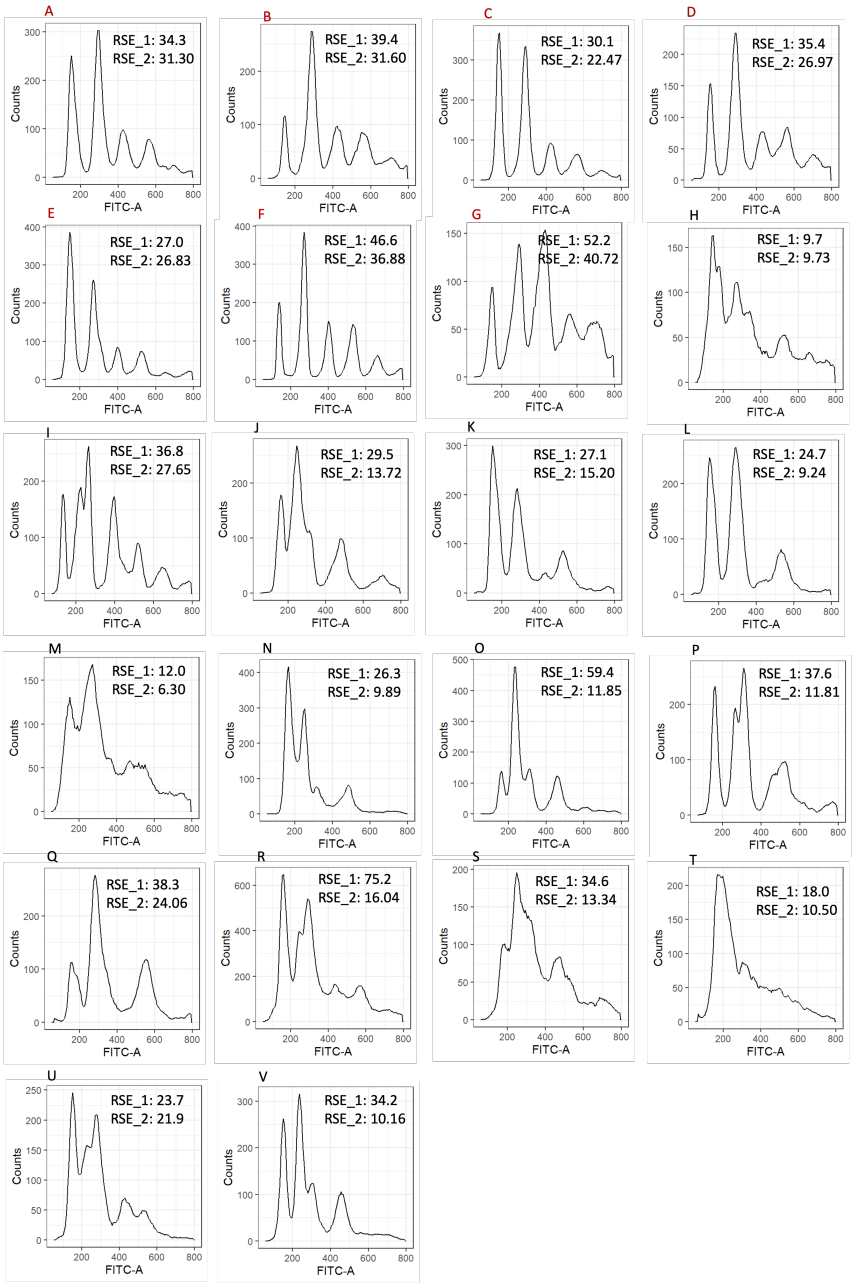

Figure 3.6: Samples from t8 flagged as having subpopulations. Two samples, H & I, have three samples, and the rest have two identified subpopulations. The samples with a red letter are the subpopulation samples flagged for review.

flagged seem like a doublet population was not identified (Figure 3.7 E & F). The last two samples both have really tall doublets mis-classified as G2 peaks which means they are likely single populations(Figure 3.7 G & H).

After quantitative and visual inspection, we identified 21 samples with subpopulations. Seventeen were classified correctly while we found an additional four samples that were initially classified as single populations, but were flagged for review. We removed five samples that were flagged for review, we found they either had tall doublets or an odd shape and were in fact single populations.

The final time point in the series was t10. From the 240 samples, the algorithm identified 25 samples with subpopulations (Figure 3.8) and flagged 21 for review (including seven with subpopulations; Figure 3.9). The algorithm found that one of the 25 samples have 3 populations, while the others have two. There was good visual support for this in subpopulations in all samples except those that were flagged. Seven subpopulation samples were flagged for review. A11-7, which is flagged for review, should be classified as a single population. It has an odd shape and the algorithm found the two G1 peaks located at 205 and 240, and their G2 at peaks 380 and 500, but the final RSE of 8.16 is for the single model (Figure 3.8 A). Two samples have odd shapes (Figure 3.8 B & C) but they do seem to have clear subpopulations. Three samples have tall G2+G2 doublets that were likely incorrectly identified as a secondary population (Figure 3.8 D, E & F). In sample A09-11, a small mass of cells to the right of the G2 peak around x = 350 was classified as a subpopulation

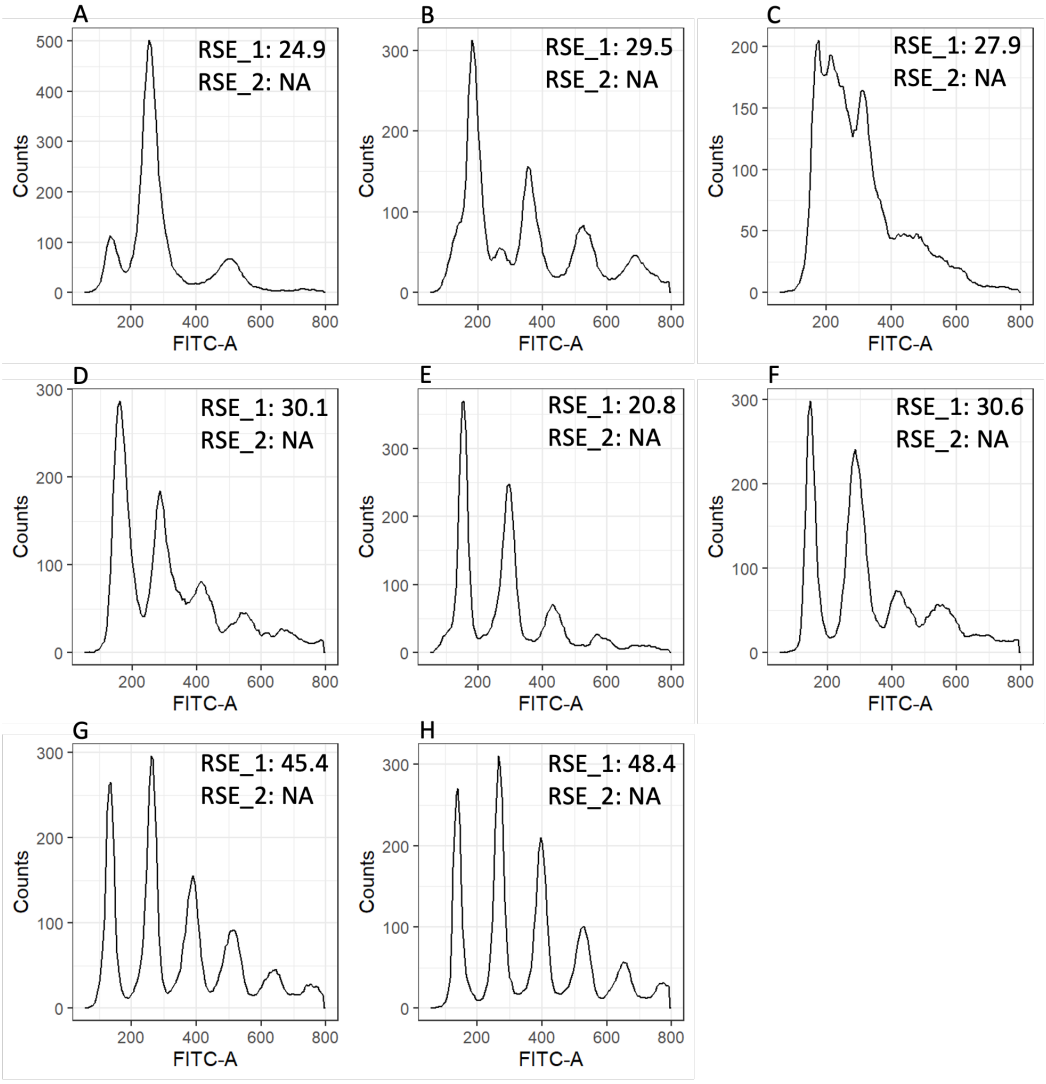

Figure 3.7: Sample flagged in t8 for users to investigate. Eight additional samples were flagged for review. Upon visualization we found four samples (A, B, C & D) do have subpopulations and the other samples are single populations.

component but I would likely classify this sample as a single population (Figure 3.8 G).

Table 3.4: emphploidyPeaksOutput for the 25 samples with subpopulations in t10. Sample A11-C7 was flagged since it has a lower single RSE than their multiple RSE, all other samples had a lower multiple model RSE. The G1 for the first subpopulations ranged from 145-205 and 300-435 for the second subpopulation.

Data	G1_1	G1_2	G1_3	G2_1	G2_2	G2_3	G1Count_1	G1Count_2	G1Count_3	G2Count_1	G2Count_2	G2Count_3	singleRSE	multipleRSE
A02-10	170	355	NA	355	665	NA	95.4	244	NA	244	57.2	NA	23.8611527	6.58094075
A04-12	165	215	NA	335	435	NA	358.6	427	NA	137.4	107.4	NA	25.5904003	12.7042632
A07-12	165	325	NA	325	630	NA	302.75	258.25	NA	258.25	60	NA	23.7236941	10.2637573
A09-11	165	245	NA	335	475	NA	1019.25	364.25	NA	170.5	61.25	NA	33.2297188	15.0434163
A09-2	145	185	NA	300	340	NA	529	278	NA	215.5	203.5	NA	22.3716239	15.8585165
A10-1	175	240	NA	395	410	NA	62.5	93	NA	133.5	135.25	NA	18.3291315	17.829535
A11-5	160	250	NA	360	430	NA	231.6	239.2	NA	60	62.6	NA	24.5903127	5.57934285
A11-7	205	240	NA	380	415	NA	185.25	160.25	NA	93	78	NA	8.15559144	14.33472
A13-1	175	305	NA	305	600	NA	144	304.75	NA	304.75	87.75	NA	34.7259737	32.4891382
A13-3	195	290	NA	370	570	NA	129.4	206	NA	154.6	66	NA	23.5947051	14.4617723
A13-6	195	310	375	375	630	645	92.4	136.6	146.6	146.6	93.2	91	34.5578949	6.1312766
A13-7	205	330	NA	435	625	NA	164.75	71.5	NA	148.5	76.25	NA	28.2603106	9.42240692
A13-8	175	360	NA	360	710	NA	155	211.25	NA	211.25	70	NA	37.0315795	17.8431791
A13-9	185	355	NA	355	700	NA	70.5	173.75	NA	173.75	77.25	NA	38.5827151	15.7696429
A14-1	165	200	NA	365	395	NA	274.5	238.75	NA	53.25	59.75	NA	45.7015232	17.8951051
A14-10	170	340	NA	340	660	NA	212	170.75	NA	170.75	61	NA	29.997735	22.9797008
A14-4	200	290	NA	350	525	NA	164	87.4	NA	142.2	61.2	NA	15.359488	6.93968408
A14-5	190	355	NA	355	645	NA	158	163	NA	163	73.75	NA	32.345303	6.55030339
A18-10	155	205	NA	315	395	NA	326	190	NA	204.4	86.8	NA	13.7731669	12.9249075
A18-12	165	205	NA	320	390	NA	359.8	224	NA	272.4	92.6	NA	22.4311936	12.4426274
A18-6	195	285	NA	395	540	NA	136.25	350.5	NA	112	98.75	NA	26.0132197	8.64616284
A18-7	175	370	NA	370	670	NA	220.4	273.2	NA	273.2	61.8	NA	21.2190626	14.5761998
A19-11	185	325	NA	325	620	NA	575.75	309.25	NA	309.25	90.25	NA	37.0798369	35.2115407
A20-11	205	230	NA	385	460	NA	360.4	370	NA	128.4	76.4	NA	15.7916389	14.2118278
A20-8	195	240	NA	385	450	NA	353	386.5	NA	66	55.25	NA	11.3490477	11.2180128

Fourteen additional samples were flagged for review. Two samples have subpopulations that were missed. The RSE for the single model was for A20-12 was high (28.24), this sample has very small G2 peaks in both subpopulations (Figure 3.9 A). Similarly, we can visually see that A16-11 has subpopulations, but it does have an odd shape, and the algorithm had a hard time finding the second G2 peak (Figure 3.9 B). Some had odd shapes (Figure 3.9 C-J), one sample has an odd shaped G1 peak (Figure 3.9 K), one had tall doublets (Figure 3.9 L), all of the samples from the group A09 (Figure 3.9 G, M & N)

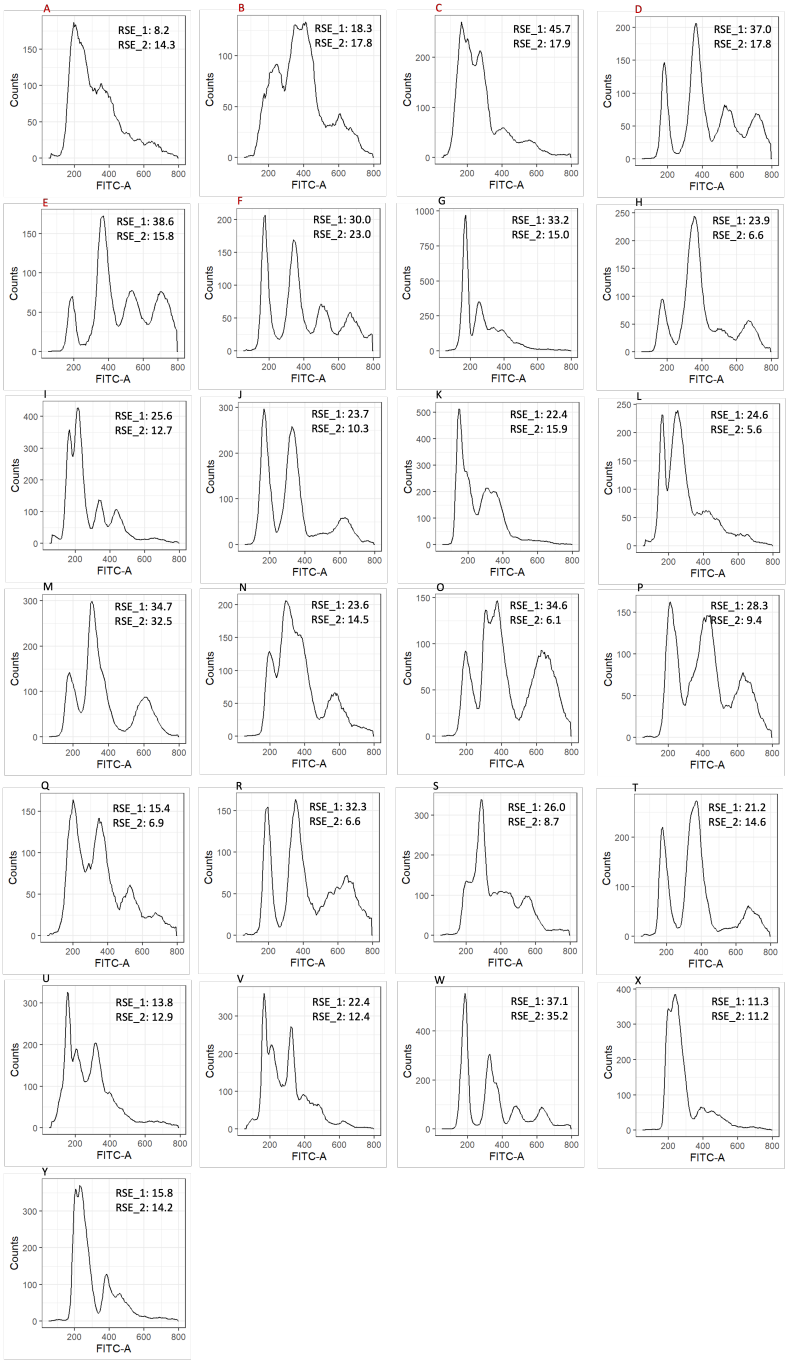

Figure 3.8: Twenty-five samples from t10 with subpopulations in the t10.

Figure 3.8: (cont.) One of the samples, O, was flagged as having three populations, and the rest as having two populations. The samples with a red letter are flagged for review (A-G). After visually reviewing the samples A, D, E, F and G, we conclude that they are in fact single populations and the rest are correctly identified.

had higher RSE because the models did not fit their s-phases well.

At the conclusion of the automated and manual (visualization) analysis, we identified 22 samples with subpopulations. Twenty were correctly classified, and we found an additional two samples while investigating the flagged samples, they were initially classified as single populations. We removed five samples from the samples with subpopulations. All five samples were removed upon reviewing the flagged subpopulations, we found they had an odd shape or tall doublets, but we concluded that are single populations.

We looked at the genome size variation between the t0, t8 and t10 samples. Replicates from some strain backgrounds did not really change over time (A1, A5, A12, A18, A20; Figure 3.10). A single evolved replicate of A15 had an increased genome size at the end. Two evolved replicate, A6 and A11, had variation but did not have any subpopulations. Evolved replicates from the other groups (A2, A3, A4, A7, A8, A9, A10, A13, A14, A16, A17, A19) exhibited considerable variation, and these samples included subpopulations, pointing to increased genome instability in these genetic backgrounds.

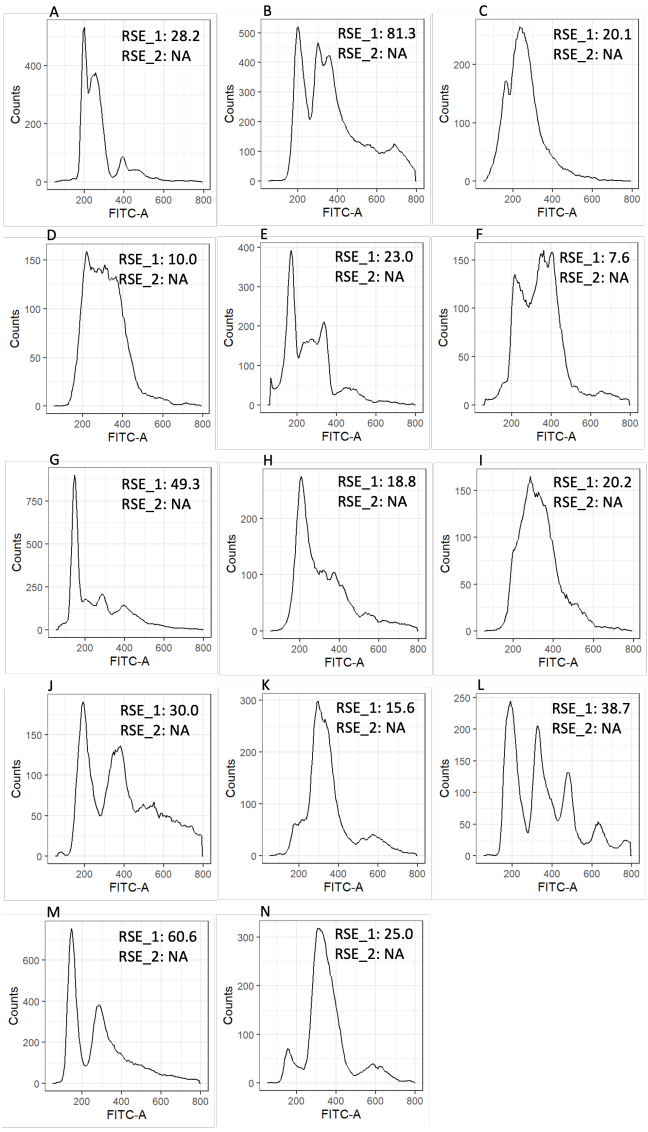

Figure 3.9: Fourteen additional samples were flagged for review in t10. Two samples have subpopulations that were missed. A)The RSE for the single model was for A20-12 was high (28.24), this sample has very small G2 peaks in both subpopulations that were missed. B) Similarly, we can visually see that A16-11 has subpopulations, but it does have an odd shape, and the algorithm had a hard time finding the second G2 peak. All other samples are single populations.

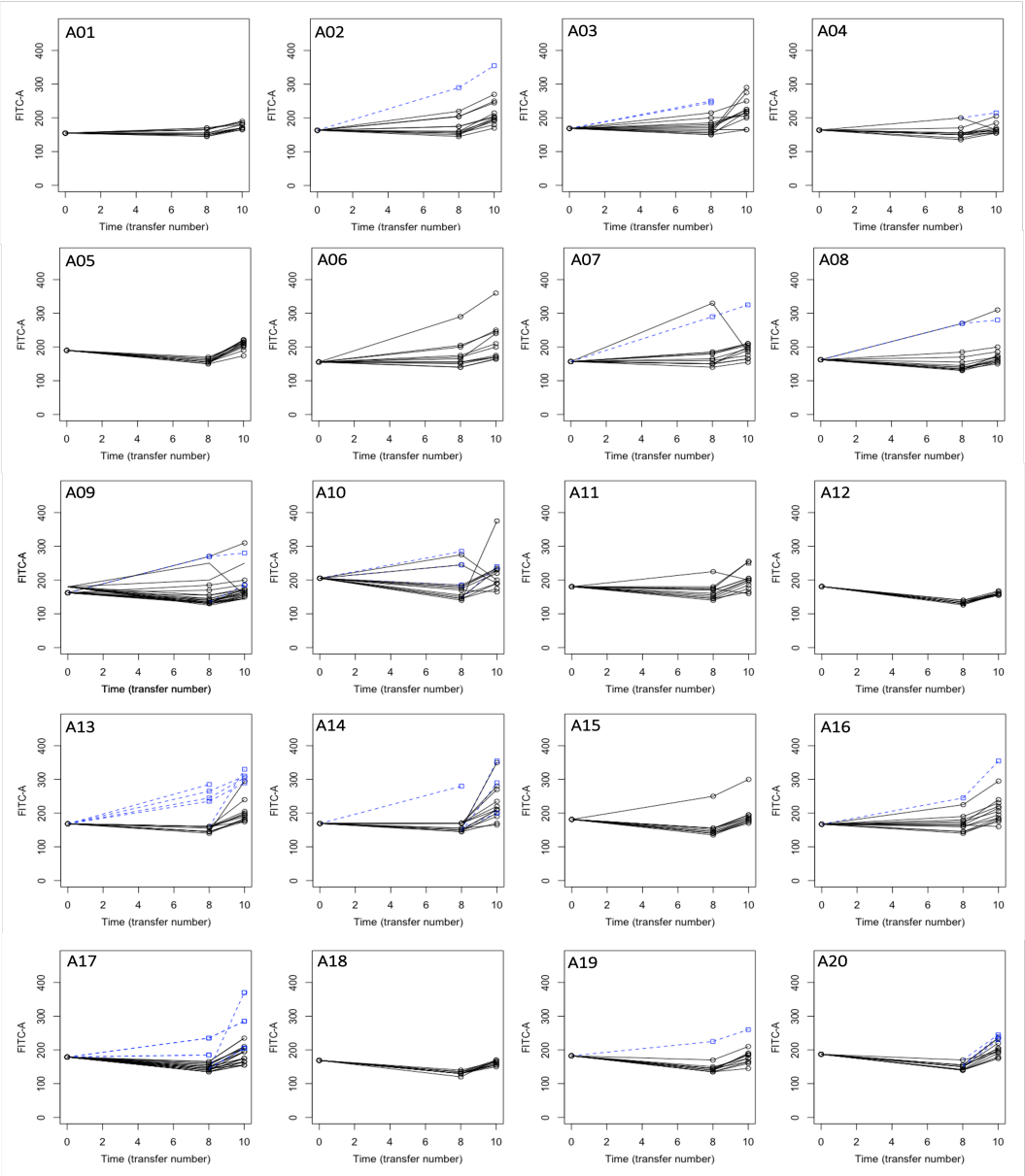

Figure 3.10: Genome size variation in replicates evolved from 20 strain backgrounds evolved to the drug fluconazole for 100 generations. Each Panel demonstrates the variation in G1 mean from the ancestral strains (t0), to the evolved strains (t10). The dashed blue line and square indicates the samples with subpopulations.

3.3 Conclusion

Throughout both experiments, we saw that `flowPeakDetection()` did a generally good job at identifying subpopulations where they obviously existed, as well as flagging for investigation samples where subpopulations were less obvious. From visual review and looking at the RSE values for the subpopulation and flagged samples, we were able to identify additional samples with subpopulations as well as make corrections to the initial classification. Without ploidyPeaks, we would have had to go through each sample individually. Thus, by using the package, we saved considerable time by only looking at samples that were flagged. On average, the `flowPeakDetection()` run time can take as little as 30 seconds or up to 3 minutes depending on the numbers of samples being analyzed.

Chapter 4

Discussion

We have developed an approach to analyze DNA fluorescence intensity data in an unbiased manner, in an open source software. This will allow anyone to use this application without requiring to be a coding expert. The goal of this package was to eliminate the user bias as well as minimize the amount of work required to analyze DNA fluorescence intensity data. Our package allows users to gate the data without, visualize as many samples as needed, determine the number of subpopulations within a sample, find the genome size, determine if the sample has doublets, and get an indication of the model fit.

Previous applications and workflows were time consuming; the user had to manually gate each sample and apply the DJF or WP cell cycle algorithm. If the algorithm did not perform well, the user can backtrack and try to re-gate the data and see if the cell cycle algorithm will fit better. This would be done for each sample. We want to lighten the workload by eliminating manual gating, and analyzing the data with user input. We created a confidence term (RSE) so

that the user will not need to review every sample. They might need to review just samples that have a high RSE or have been flagged to get investigated.

A secondary goal is to better understand the dynamics of karyotype subpopulations and their roles in adapting populations. By being able to analyze time series data unbiasedly, we can get a better idea as to the causes and consequences of mutations that evolve through generations.

4.1 Limitations and adaptations

As we saw in chapter 3, the algorithm did not always correctly categorize the samples with subpopulations, but when looking at the RSE values provided for the sample we found our own conclusion. The biggest issue we have encountered is differentiation between the G2+G2 doublet and the G2 in the second subpopulation. This is left to the user to have knowledge on their data. We have added a parameter called *maxDoubletHeight* that will allow the user to give a maximum height for the G2+G2 doublet. If a peak is identified higher than the *maxDoubletHeight* provided, it will be considered a G2 for the second population. If left blank, we have a default height in our algorithm based on testing datasets. If the user would like to rerun some samples with new thresholds, we created a parameter called *subsetDs* that will take a vector of sample names, and `flowPeakDetection()` will analyze those samples and provide similar outputs.

Working with biological data can be tricky. We make due with what is given

to us. The preparation step in DNA fluorescence intensity data is important. Many factors can affect the results of the flow cytometry prep, such as flow cytometer calibration, cell staining protocol. Due to this the algorithm will perform to its best abilities but if the data is not in its best format, we may not get the best results. Through observation it seems that some strains are more prone to seeing doublets, which could be due to poor cell separation and possibly correlated with biological factors of interest and *ploidyPeaks* will permit researchers to study/test hypotheses about doublets in a way that hasn't been done before.

4.2 Future works

We are in the process of adding more functions. The first function will serve its purpose after the user has reviewed the results. Consider a situation where you visualize a sample and you want to see the results of this sample with four peaks, creating two subpopulations. The function will allow the user to input the number of peaks and the number of subpopulations they want to analyze. The function will work in a similar manner by using peak detection algorithms as well as using the same helper functions. The outputs will be the same, but just for the sample they are analyzing. We will even take it a step further, we will provide an option for the user to input peak locations. Consider the same example, the user wants the sample to have four peaks with two subpopulations, and they want the second peak to be located at $x = 250$.

The algorithm will take that into consideration while finding the other three peaks.

Bibliography

Bagwell, C. B., M. Inokuma, B. Hunsberger, D. Herbert, C. Bray, B. Hill, G. Stelzer, S. Li, A. Kollipara, O. Ornatsky, and V. Baranov (2020, 2). Automated data cleanup for mass cytometry. *Cytometry Part A 97*, 184–198. (Cited on page 5.)

Ellis, B., P. Haaland, F. Hahne, N. L. Meur, N. Gopalakrishnan, J. Spidlen, M. Jiang, and G. Finak (2022). Package 'flowcore' title flowcore: Basic structures for flow cytometry data. (Cited on page 10.)

Ester, M., H.-P. Kriegel, J. Sander, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pp. 226–231. AAAI Press. (Cited on page 18.)

Ezov, T. K., E. Boger-Nadjar, Z. Frenkel, I. Katsperovski, S. Kemeny, E. Nevo, A. Korol, and Y. Kashi (2006). Molecular-genetic biodiversity in a natural population of the yeast saccharomyces cerevisiae from "evolution canyon": Microsatellite polymorphism, ploidy and controversial sexual status. *Genetics 174*, 1455–1468. (Cited on page 2.)

Fox, M. H. (1980). A model for the computer analysis of synchronous dna distributions obtained by flow cytometry. *Cytometry 1*, 71–77. (Cited on pages 4 and 29.)

Gentleman, R. C., V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang (2004). Open access bioconductor: open software development for computational biology and bioinformatics. (Cited on page 10.)

Gerstein, A. C. and J. Berman (2020, 6). Candida albicans genetic background influences mean and heterogeneity of drug responses and genome stability during evolution in fluconazole. *mSphere 5*. (Cited on page 54.)

Gerstein, A. C. and N. P. Sharp (2021, 01). The population genetics of ploidy change in unicellular fungi. *FEMS Microbiology Reviews 45*(5). fuab006. (Cited on page 2.)

Harari, Y., Y. Ram, N. Rappoport, L. Hadany, and M. Kupiec (2018). Spontaneous changes in ploidy are common in yeast. *Current Biology 28*, 825–835.e4. (Cited on page 2.)

Kukurudz, R. J., M. Chapel, Q. Wonitowy, A.-R. A. Bukari, B. Sidney, R. Sierhuis, A. C. Gerstein, and A. Gerstein (2022). Acquisition of cross-azole

tolerance and aneuploidy in candida albicans strains evolved to posaconazole running title: Posaconazole evolution in candida albicans. *Biorxiv*. (Cited on page 54.)

Lang, G. I., D. P. Rice, M. J. Hickman, E. Sodergren, G. M. Weinstock, D. Botstein, and M. M. Desai (2013). Pervasive genetic hitchhiking and clonal interference in forty evolving yeast populations. *Nature 500*, 571–574. (Cited on page 1.)

Lo, K., F. Hahne, R. R. Brinkman, and R. Gottardo (2009, 5). flowclust: A bioconductor package for automated gating of flow cytometry data. *BMC Bioinformatics 10*, 145. (Cited on page 11.)

McKinnon, K. M. (2018, 2). Flow cytometry: An overview. *Current Protocols in Immunology 120*, 5.1.1–5.1.11. (Cited on page 4.)

Ochi, S. (2019). Package 'scorepeak' type package title peak functions for peak detection in univariate time series. (Cited on page 21.)

Peter, J., M. D. Chiara, A. Friedrich, J.-X. Yue, D. Pflieger, A. Bergström, A. Sigwalt, B. Barre, K. Freel, A. Llored, C. Cruaud, K. Labadie, J.-M. Aury, B. Istace, K. Lebrigand, P. Barbry, S. Engelen, A. Lemainque, P. Wincker, G. Liti, and J. Schacherer (2018). Genome evolution across 1,011 saccharomyces cerevisiae isolates. *Nature 556*, 339–344. (Cited on page 2.)

Phu, V., W. Jiang, R. Gottardo, and G. Finak (2018, 11). Ggcyto: Next gen-

eration open-source visualization software for cytometry. *Bioinformatics 34*, 3951–3953. (Cited on page 11.)

Rice, D. P., B. H. Good, and M. M. Desai (2015). The evolutionarily stable distribution of fitness effects. *Genetics 200*, 321–329. (Cited on page 1.)

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. (Cited on page 16.)

Smith, T. (2018). flowploidy: Getting started. (Cited on page 12.)

Smith, T. (2022). flowploidy: Analyze flow cytometer data to determine sample ploidy. (Cited on page 11.)

Tibshirani, R., G. Walther, and T. Hastie (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society 63*, 411–423. (Cited on page 18.)

Watson, J. V., S. H. Chambers, and P. J. Smith (1987). A pragmatic approach to the analysis of dna histograms with a definable g1 peak. *Cytometry 8*, 1–8. (Cited on pages 4, 29 and 37.)

Zeileis, A., G. Grothendieck, and J. A. Ryan (2022). Package 'zoo' title s3 infrastructure for regular and irregular time series (z's ordered observations). (Cited on page 23.)

Zhu, Y. O., G. Sherlock, and D. A. Petrov (2016). Whole genome analysis of 132 clinical saccharomyces cerevisiae strains reveals extensive ploidy variation. *G3: Genes, Genomes, Genetics 6*, 2421–2434. (Cited on page 2.)

Appendix A

Vignette

ploidyPeaks: Peak detection and statistical analysis of karyotypic variation from flow cytometry data

Introduction to ploidyPeaks

Karyotypic variation is observed in fungal microbial populations isolated ecologically, clinically, and industrially, and is also the hallmark of many populations of cancer cells. Flow cytometry is the standard method to measure genome size in cell populations. A dye that stains DNA is applied to a population that contains cells at all phases of the cell cycle, i.e., G_0/G_1 cells prior to DNA replication, S phase cells undergoing replication, and G_2/M cells that have doubled their DNA but not yet divided. The flow cytometer passes each cell one-by-one across a laser and captures the fluorescence of each cell, corresponding to genome size. Typically 10,000 cells are measured. Two univariate models are widely used and incorporated into standard software. The Watson Pragmatic

and Dean-Jett-Fox cell (DJF) cycle models both assume that G_0/G_1 and G_2/M cell clusters are Gaussian distributed, but vary when finding the distribution for S phase cells. In practice with fungal microbial populations, the two models typically give nearly identical results. When all cells in the population are a single ploidy population, the the mean of the G_0/G_1 peak is interpreted as the genome size, corresponding to ploidy. When there are multiple subpopulations of mixed ploidy (which is commonly observed during evolution *in vitro* and *in vivo*), the researcher must manually gate the population into subpopulations prior to analysis based on their best guess of the subpopulation locations and boundaries. We developed the R Bioconductor package *ploidyPeaks* to quantify subpopulations genome sizes without requiring any user input. The package uses theory from previous work and peak detection algorithms to identify possible G_0/G_1 and G_2/M peaks, and then identify and quantify sub-populations. In addition, *ploidyPeaks* extends existing algorithms by providing a statistical confidence term (residual standard errors). The workflow is separated into four parts, with built-in visualizations provided throughout: 1) gating, 2) peak detection, 3) confidence term, and 4) dataset output. *ploidyPeaks* will allow users to analyze their flow cytometry data unbiasedly with confidence and with minimal prior coding knowledge required.

Availability

ploidyPeaks is currently on GitHub, you must install the package prior to the analysis. If this is your first time downloading the package you can do it using: `devtools::install\_github("margothenry/PloidyPeaks")`. The vignette, package dependencies and system requirements can be found (eventually link to bioconductor page with this info). The package has been tested on macOS (versions 12.3.1 and later) and Windows, using R version 4.1 and later.

Implementation

The main functions of *ploidyPeaks* are:

- `rectGateFlowFrame()`: A function to gate out debris from one sample (termed a 'flow frame' by *flowCore*).
- `rectGateFlowSet()`: A function to gate out debris for a whole data set (termed a 'flow set' by *flowCore*).
- `flowLineGraph()`: A function to visualize multiple samples at once.
- `flowPeakDetection()`: The main package function used to identify G_0/G_1 and G_2/M peaks from all subpopulations using peak detection, to apply DJF to get a RSE, and to create outputs such as graphs and a *.csv*.

Preparation of the data: Gating

The flow cytometer scatters light though all suspended particles, and those light scatters are read into a detector that can visualize the light, this information will be passed on to an analyzer that will convert them to electrical impulses. Multiple parameters are measured for each cell. Forward scatter (FCS) measures the light scatter in the same direction as the laser, providing information on cell size, while side scatter (SSC) measures scatter on a 90 degree angle, providing information on cell granularity, which can be used to identify to identify the granularity of the cells. Fluorescence is measured in specific fluorescence channels that are named based on machine settings and applications. When the cells get passed through the laser it pulses, and that pulse is identified by height (-H), area (-A), and width (-W) for each parameter. Unlike cell size/shape parameters (i.e., SSC, FSC), the fluorescence intensity/genome size parameter can be called by arbitrary fluorescence/laser numbers (FL1, FL2 etc.) or the name of the dye used to stain the DNA (FITC, PI, SYBR, SYTOX etc.)

Measured particles often include the cell population of interest as well as debris, which includes inert particles such as dust that are introduced during the cell preparation process as well as large clumps of cells that stick together. These cause an excess of cells along the plot margins. It is common to remove the debris by first gating on a bivariate density plot. The parameters can be FCS, SSC or any fluorescence that was applied in the flow set. The population

of interest is visualized as a dense area in the plot.

Two gating functions were created: `rectGateFlowFrame()` applies the gate on a single flow frame (.fcs), while `rectGateFlowSet()` applies the gate to a flow set, a series of multiple flow frames. Combined they apply a rectangular gate to the particles and save the cells within the rectangular boundary as a .fcs file in a folder that is created called 'gated_data' for downstream analysis. The gate is set to default values of $xMinValue = 10000$, $xMaxValue = 900000$, $yMinValue = 10000$ and $yMaxValue = 900000$, gating on variables "SSC-A" and "FL1-A", but the user has the option to input their own rectangular boundaries. There is an option to save graphs to display the comparison between the raw data (pre-gating), and the gated data (savePlot = TRUE). `rectGateFlowFrame()` is used for testing, the user can see how the data looks on a single sample, adapt the gating boundaries if needed, and once satisfied, apply the gate to the whole flow set with `rectGateFlowSet()`.

Examples of different minimum values for the same sample:

```
rectGateFlowFrame(
  rawDir = "flow_data\raw_data\",
  flowName = "Sample1",
  xVariable = "FL1-A",
  yVariable = "SSC-A",
  xMinValue = 20000,
  xMaxValue = 950000,
  yMinValue = 10000,
  yMaxValue = 950000,
  savePlot = TRUE
)
```

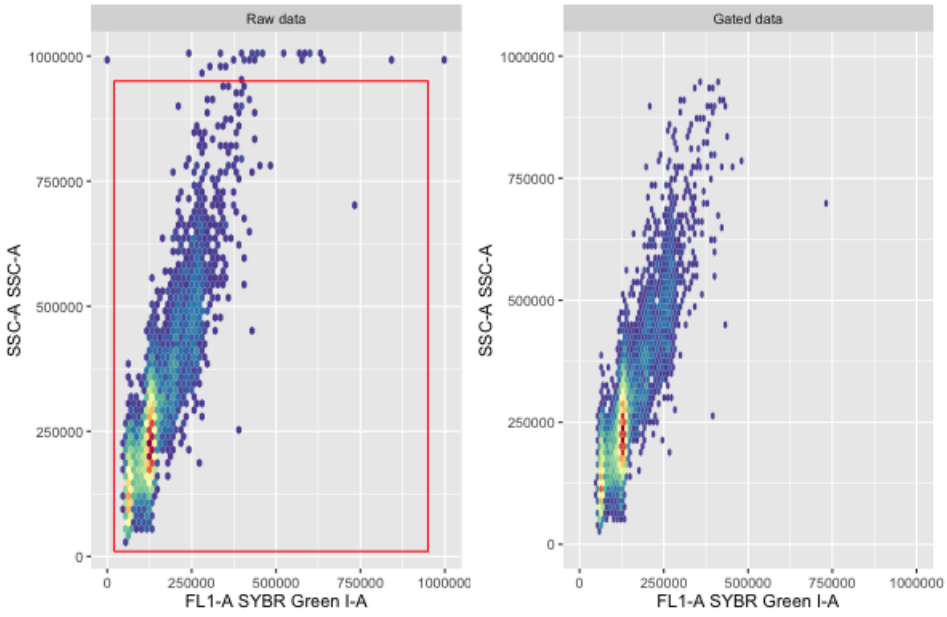

Figure A.1: Example 1 of *rectGateFlowFrame()*

```
rectGateFlowFrame(
  rawDir = "flow_data\raw_data\,
  flowName = "Sample1",
  xVariable = "FL1-A",
  yVariable = "SSC-A",
  xMinValue = 30000,
  xMaxValue = 700000,
  yMinValue = 30000,
  yMaxValue = 750000,
  savePlot = TRUE
)
```

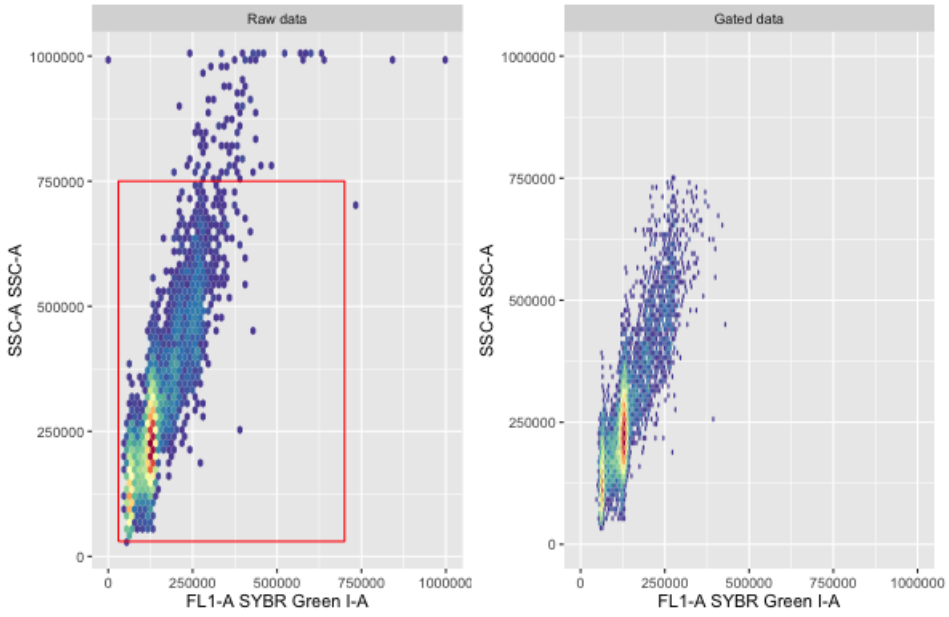

Figure A.2: Example 2 of *rectGateFlowFrame()*

Visualization

The data from a single parameter (typically the one that corresponds to fluorescence intensity) can be visualized as a histogram using `flowLineGraph` This function permits the users to plot as many samples as they would like on top of each other on the same panel. Each trace indicates a different sample. If desired, a black line will indicate the control population that is specified by the user for comparison.

Examples:

```
flowLineGraph(
  flowDir = here("vignettes/data/gated_data"),
  flowControl = "A4_12.fcs",
  flowSamples = c(
      "A3_9.fcs",
      "A2_4.fcs",
      "A4_4.fcs",
      "A1_12.fcs",
      "A1_6.fcs"
      ),
  xVariable = "FITC-A"
)
```

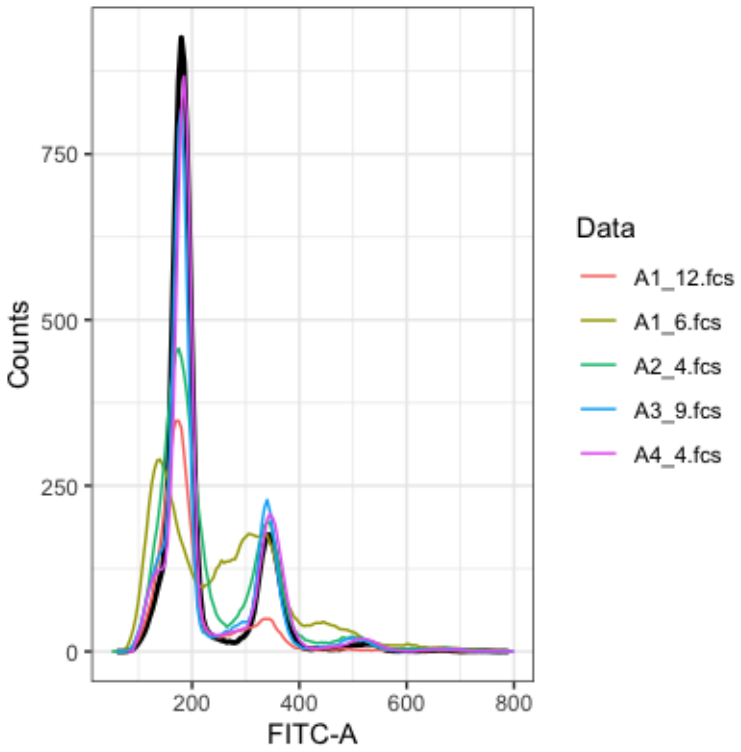

Figure A.3: *flowLineGraph()* example with a control sample

By leaving the *flowControl* parameter blank, no black line will appear.

```
flowLineGraph(
  flowDir = here("vignettes/data/gated_data"),
  flowSamples = c(
      "A3_9.fcs",
      "A2_4.fcs",
      "A4_4.fcs",
      "A1_12.fcs",
      "A1_6.fcs"
      ),
  xVariable = "FITC-A"
)
```

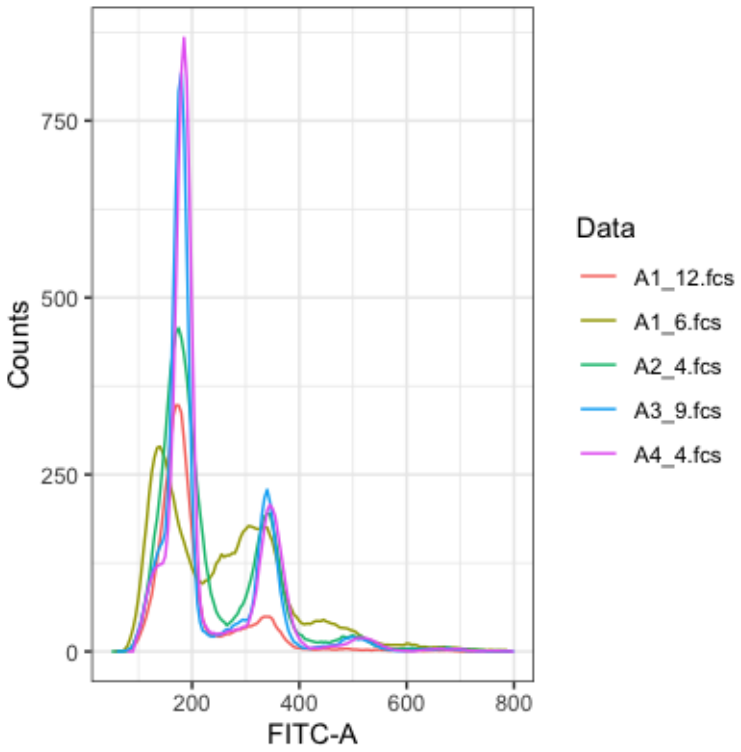

Figure A.4: *flowLineGraph()* using example without a control sample

You can plot a single sample by putting a single sample in either the ‘flowSamples‘ or ‘flowControl‘ parameters.

```
flowLineGraph(
  flowDir = here("vignettes/data/gated_data"),
  flowSamples = c("A3_9.fcs"),
  xVariable = "FITC-A"
)
```

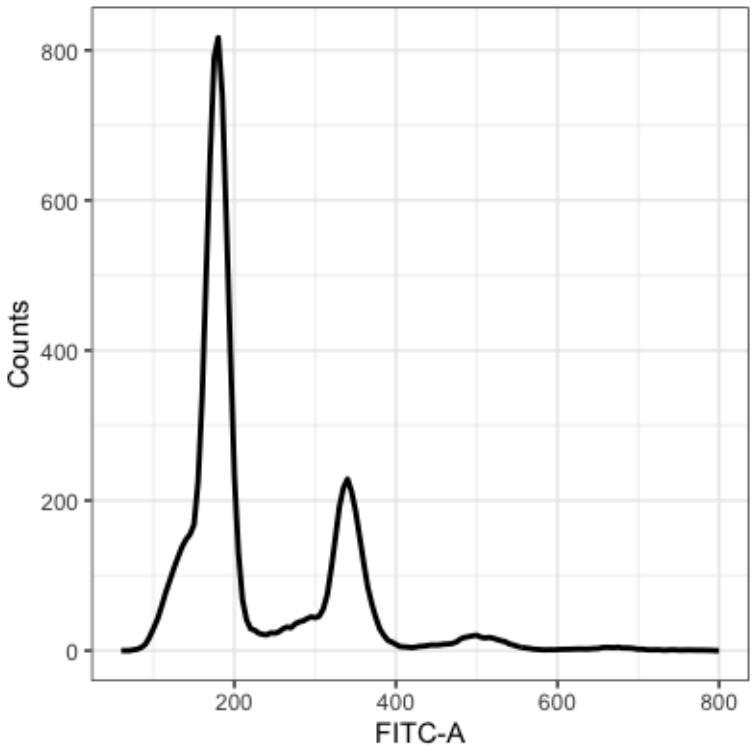

Figure A.5: *flowLineGraph()* using example with one sample

Analysis of the data

The core of the package is the peak detection analysis, with the goal of properly identifying G1 and G2 peaks from all subpopulations. The wrapper function `flowPeakDetection()` is the main function that consists of five main peak algorithm functions, as well as multiple helper functions used within the main functions. The main output of `flowPeakDetection()` is a *.csv* file called *ploidyPeakOutput* with information about the location of each peak and their height, as well as a doublet and a "to investiage" indicator. In addition, the Dean-Jett-Fox cell cycle model is used to obtain a statistical measure of confidence for the number of sub-populations.

flowPeakDetection() **function overview:**

`peakAlgorithm1()`

`peakAlgorithm1()` is the first branching point to identify samples with single populations (which contain only two peaks and can be easily analysed) and to flag everything else. The algorithm will smooth the data with a smoothing level of 11 with the `smoothData()` function. Then it finds a single G1 and G2 pairing. If there are more than two peaks identified, the algorithm will consider the G1/G2 pairing that is furthermost left on the x-axis to be the single population. With that population, a boundary is set to the right of the G2 peak and the ratio of cells in that area are calculated. If this ratio is within a certain threshold, the population is marked as a single population. If

the ratio exceeds the threshold, the population is flagged to pass to the next algorithm, `peakAlgorithm1()`. A default threshold of 8.5% was established by looking at the ratio of cells after the G2 peak from 500 known control populations composed of known *Candida albicans* single populations. But it is straightforward for researchers to infer an appropriate threshold from their data.

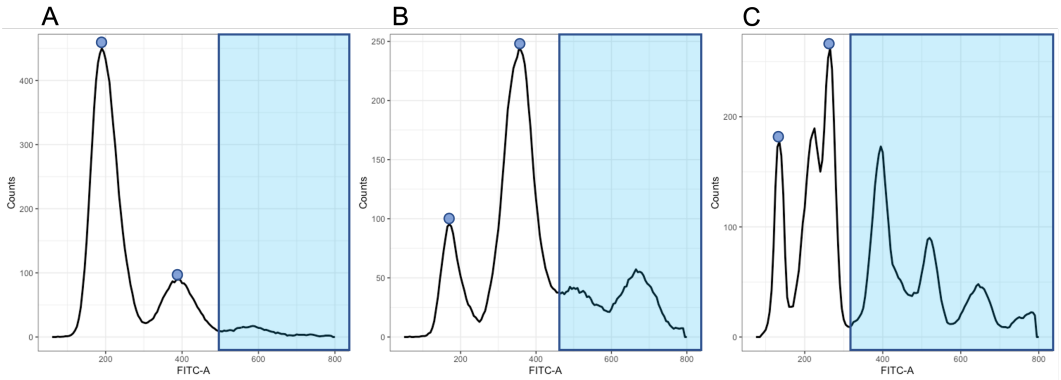

Figure A.6: PeakAlgorithm1() calculates the percentage of cells to the right of the G2 peak (i.e., beyond *popBoundary*). The blue dots represent the G1 and G2 peaks used to find *popBoundary* and the blue shaded area represents the cells found in *popBoundary* A) An example of data from a single population sample, with only 5.5% of cells beyond *popBoundary*. B) An example of a population that would be passed on to peakAlgorithm2(), where the percent of cells (32.4%) exceeds the set threshold. C) An example of a population with multiple populations. The algorithm assigns a G1 and G2 pairing from the peaks that are farthest to the left as the focal population. This sample would also be passed on to peakAlgorithm2(), as the percent of cells in *popBoundary* (49.10%) exceeds the set threshold.

`smoothData()`

`smoothData()` reads in the gated data and applies a smoothing factor to the data. Different smoothing factors are applied throughout the functions.

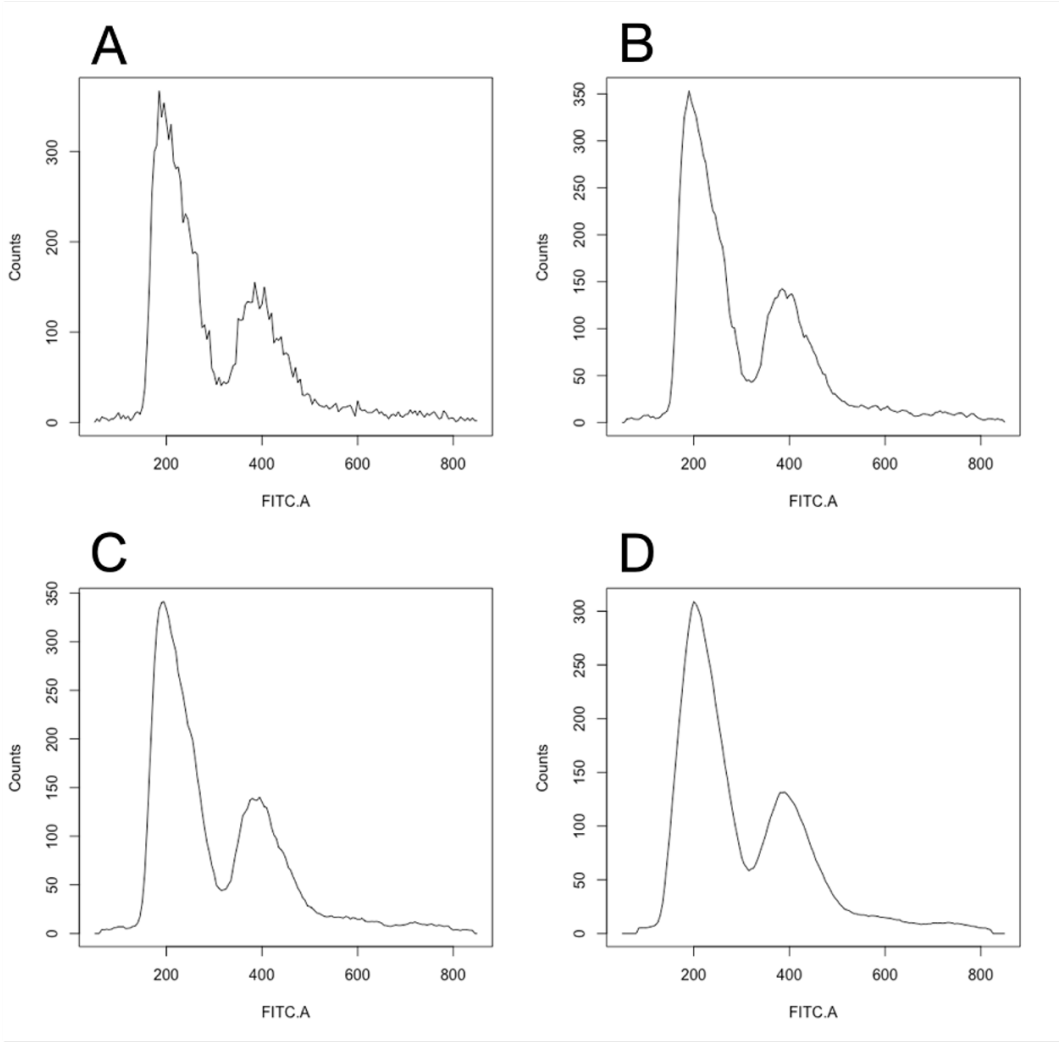

Figure A.7: This is an example of the outcome of the *smoothData()* function. A) The raw data is read into the function and a smoothing factor is applied. This eliminates small spikes from being identified as peaks. In the examples here, the smoothing factor applied was B) 3, C) 5, and D) 11.

`peakAlgorithm2()`

The second branching point is in `peakAlgorithm2()`, which seeks to identify all possible sub populations. The algorithm looks at the proportion of cells used in the subpopulations. In other words, once the populations have been identified for a given flow frame, `peakAlgorithm2()` will find the cells associated with the G1/G2 pairs. The algorithm uses more granular data, a smoothing factor of 5 and repeats the initial peak detection step as in `peakAlgorithm1()`. Two helper functions, `findTruePeaks()` and `findPairs()`, are employed to find possible sub populations.

`findTruePeaks()`

While The algorithm identifies peaks that represent both peaks of interest (i.e., peaks that correspond to true G1 and G2 populations) as well as peaks that arise in the data due to spikes or abnormalities in the histogram. `findTruePeaks()` is used to differentiate between the two types of peaks and to remove the unnecessary peaks. This function will start by assuming each peak is its own cluster, then if two clusters are too close together, they will get combined and only one peak can be identified per cluster. In Figure A.8 C, we see that three peaks are identified, and that peaks 2 and 3 are very close together. This function will group together cluster two and three to become a single cluster (6D).

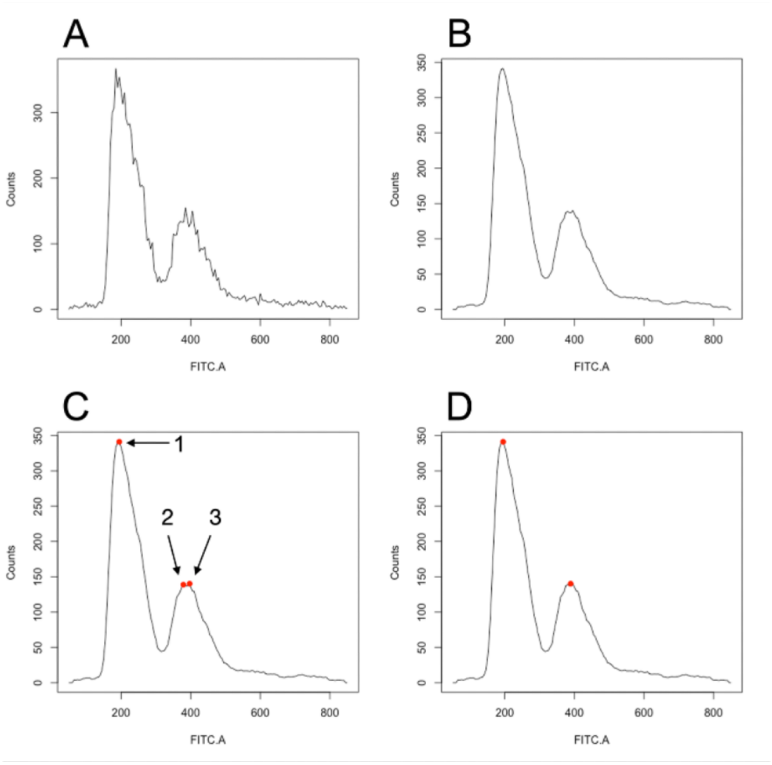

Figure A.8: The raw data that gets read into the algorithm. B) The data is smoothed by a smoothing factor of five .C) The peak detection algorithm identified three peaks. D) The findTruePeaks() function will group together peaks that are in close proximity (peaks 2 & 3).

`findPairs()`

For all the peaks that have currently been identified, `findPairs()`, assumes that each peak is a possible G1 and will try and find their subpopulation pairing. The algorithm finds a lower and upper limit for each possible G1 peak. This is done by taking into account that for every true G1 peak (i.e., genome size prior to replication), there should be a corresponding G2 peak (genome size after DNA replication before division) 2 larger (in practice, it is typically

1.75-2.0x the G1 peak). Therefore, our lower limit is 1.75xG1 peak and our upper limit is 2.2xG1 peak. For each possible G1 peak, `findPairs()` looks at peaks identified in this range, and if there is more than one peak identified, it will look at the tallest of those identified peaks to pair up with the G1 peak. If no peaks have been identified in this range, the algorithm will assume that this peak is not a G1 peak and remove from our dataset. In Figure A.9, four peak have been identified as possible G1 peaks. The algorithm found a G2 pairing for the first two peaks and the remaining two are removed from the dataset.

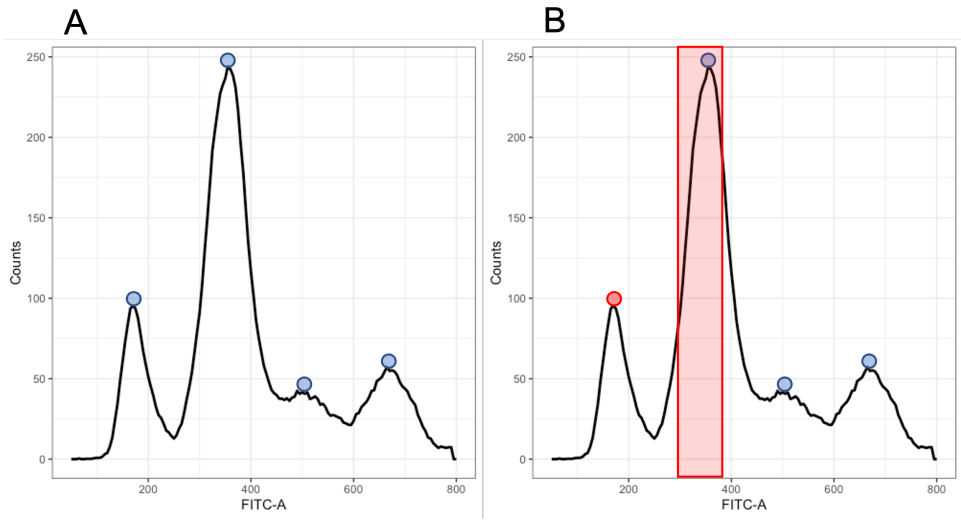

Figure A.9: Subpopulation pairings example from *findPairs()*. In A) The peak detection algorithm and *indtruePeaks()*f identified four peaks located at x=170, 355, 490 and 675, which are marked by blue dots. B) Considering the first peak at x = 170 (red dot), *findPairs()* will calculate lower and upper bounds and look to see if a peak falls within those limits. For *peakAlgorithm2()* the lower bound is set at 1.75×170 = 297.5, and the upper bound is set at 2.2×170 = 374, which is represented as the red shaded area. The peak located at x = 355 falls within those limits, so *findPairs()* assigns these two peaks as a subpopulation pair.

Once the sub population peaks are identified, the proportion of cells that are being used is calculated. If the proportion of cells used (*cellPerc*) is under our threshold of 86%, that sample will be flagged for further analysis. Currently, the threshold has been determined by looking at the *cellPerc* from known control populations composed of known *Candida albicans* single populations. But there is an option for researchers to infer an appropriate threshold from their data.

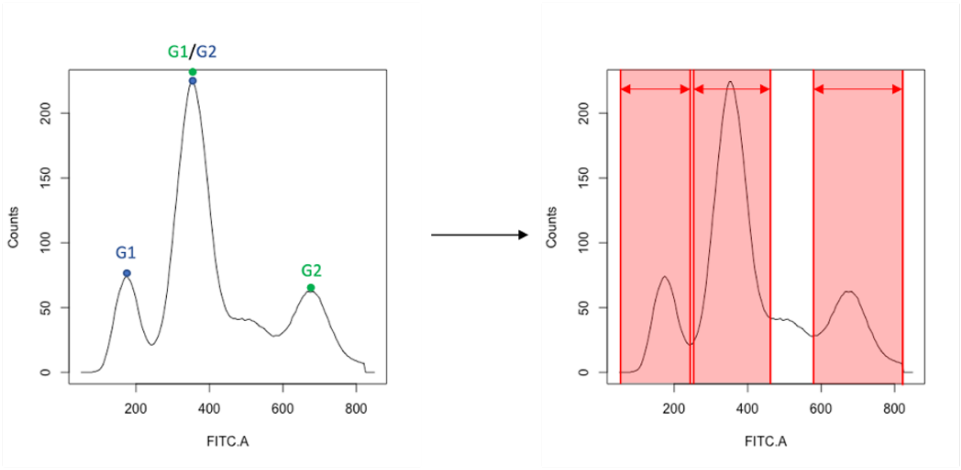

Figure A.10: The left most panel is sample with two subpopulations. The blue dots represent the G1 and G2 mean of the first subpopulation, and the green dot represents the mean of the second subpopulation. The proportions of cells being used in the subpopulation pairings is represented in the right panel in the shaded red area, while the area in white ate the cells not being used. *cellPerc* is calculated by the number of cells in the red area divided by the total number of cells $\times$ 100.

`peakAlgorithm3()`, `peakAlgorithm4()`, `peakAlgorithm5()`

Flagged populations from `peakAlgorithm2()` are passed on to the next algorithm, `peakAlgorithm3()`. The granularity is adjusted, the interval between G1 and G2 peaks is widened to (1.7, 2.3), and then `findTruePeaks()` and `findPairs()` are applied as before. If the sample still does not pass the threshold in `peakAlgorithm3()`, then it gets flagged and passed on to `peakAlgorithm4()`, where the data is more granular, with a smoothing level of four and then `findTruePeaks()` and `findPairs()` are applied as before. Samples that are flagged at the end of `peakAlgorithm4()` are annotated as

'investigate' to indicate lower confidence in the identified peaks and signify that the user should visually inspect these samples. They are passed on to `peakAlgorithm5()` which has a smoothing level of three and the G1 and G2 peak interval is widened to (1.5, 2.5) to identify for possible pairs.

`doubletCheck()`

`doubletCheck()` is used to identify doublets. Doublets are an excess of cells around the G1+G2 range. The doublet peak will not belong to any sub population as they do not fall within the proper range, and so the proportion of cells that are not part of any subpopulation is higher which will cause our peak algorithms to fail. This function is applied to each peak algorithm function prior to their branching point.

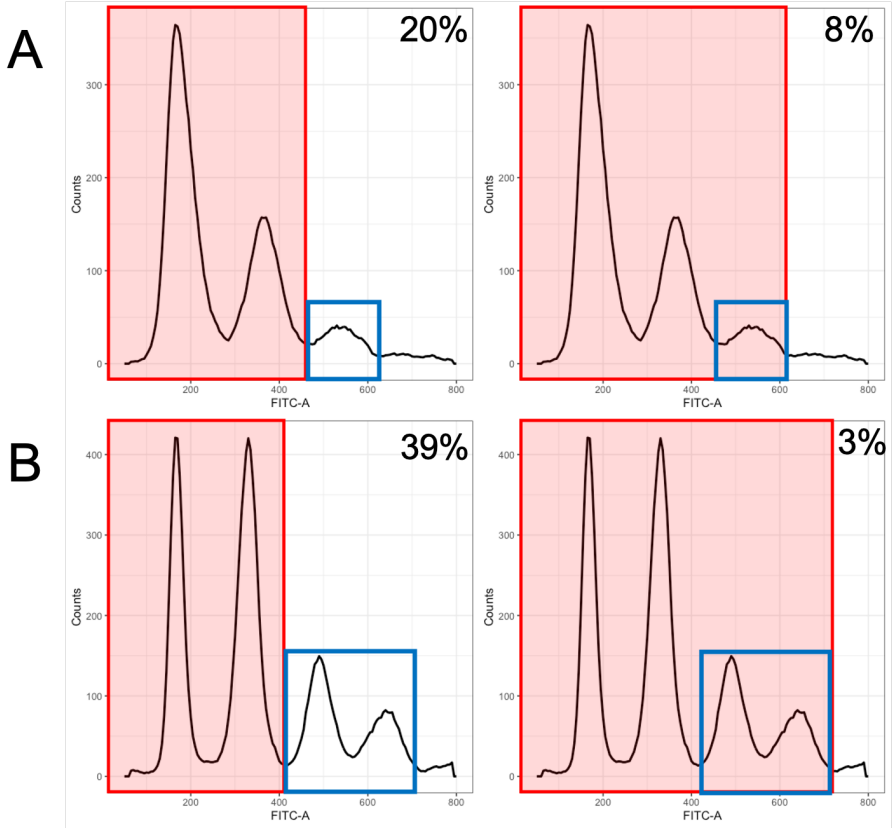

Figure A.11: Identification of doublets. A) Before specifying that there is a G1+G2 doublet peak, the sample would fail all peak detection branching points since the proportion of cells being used is 80%. After considering doublets, 92% of cells are considered accounted for. B) Before identifying the G1+G2 and the G2+G2 doublet, this sample had only 61% of cells used; once we included the doublet peaks, 97% of cells were accounted for.

Confidence term using residual standard error (RSE)

Following peak detection analysis, the Dean-Jett-Fox cell cycle algorithm is applied to each sample using the identified peak means and heights (from each subpopulation, where appropriate) to find a residual standard error (RSE)

using nonlinear least squares models. Three DJF models are run for each sample. The first model assumes each sample is a single population, the second adds doublet peaks (if identified) to the single population, and if appropriate, the third model fits each subpopulation identified by the algorithm in the process. The final RSE is the lowest of the fitted models. Users can use the RSE as a confidence term to see how well the algorithm performed and further investigate the samples with high RSE values.

The DJF cell cycle algorithm assumes the peaks are Gaussian distributed with

$$F_i(x) = \frac{N_i}{\sqrt{2\pi\sigma_i^2}} exp[-\frac{(x-x_i)^2}{2\sigma_i^2}]$$

Where the x_i's are represented by the means identified in our peak algorithm. The σ_i's are calculated as the width of the distribution at 60% of each mean. I.e., if the G1 mean is 170, the standard deviation will be calculated where the height is $0.6 \times 170 = 102$ on either side of the peak. The N_i's are calculated by summing the cells within $(-\sigma_i,\sigma_i)$.

The first model assumes each sample is a single population:

$$F_{1_1}(x) = \frac{N_1}{\sqrt{2\pi\sigma_1^2}} exp[-\frac{(x-x_1)^2}{2\sigma_1^2}]$$

G2:

$F_{2_1}(x) = \frac{N_2}{\sqrt{2\pi\sigma_2^2}} exp[-\frac{(x-x_2)^2}{2\sigma_2^2}]$

S-phase between G1 and G2:

$$F_{s1}(x) = \sum_{j=x_1}^{x_2} f(x_j) \frac{1}{\sqrt{2\pi\sigma_1^2 \frac{x_j}{x_1}}} exp[-\frac{(x-x_j)^2}{2(\sigma_1 \frac{x_j}{x_1})^2}]$$

And the model 1 is $F_{M1}(x) = F_{1_1}(x) + F_{2_1}(x) + F_{s1}(x)$

The second model will adds doublet peaks to the single population, if any were identified by `doubletCheck()`.

G1+G2 doublet:

$$F_{d1}(x) = \frac{N_{d1}}{\sqrt{2\pi\sigma_{d1}^2}} exp[-\frac{(x-x_{d1})^2}{2\sigma_{d1}^2}]$$

S phase between G2 and G1+G2 doublet:

$$F_{s_d1}(x) = \sum_{j=x_2}^{x_{d1}} f(x_j) \frac{1}{\sqrt{2\pi\sigma_2^2 \frac{x_j}{x_2}}} exp[-\frac{(x-x_j)^2}{2(\sigma_2 \frac{x_j}{x_2})^2}]$$

G2+G2 doublet:

$$F_{d2}(x) = \frac{N_{d2}}{\sqrt{2\pi\sigma_{d2}^2}} exp[-\frac{(x-x_{d2})^2}{2\sigma_{d2}^2}]$$

S phase between G1+G2 doublet and G2+G2 doublet:

$$F_{s_d2}(x) = \sum_{j=x_{d1}}^{x_{d2}} f(x_j) \frac{1}{\sqrt{2\pi\sigma_{d1}^2 \frac{x_j}{x_{d1}}}} exp[-\frac{(x-x_j)^2}{2(\sigma_{d1} \frac{x_j}{x_{d1}})^2}]$$

And our model 2 is

$$F_{M2}(x) = F_{1_1} + F_{2_1}) + F_s(x) + F_{d1}(x) + F_{s_d1}(x) + F_{d2}(x) + F_{s_d2}(x)$$

If appropriate (i.e., if `flowPeakDectection()` identified any subpopulations), the third model fits each subpopulation identified by the algorithm.

$$F_{multi}(x) = \sum_{i=1}^{n} F_{1_i}(x) + F_{2_i}(x) + F_{s_i}(x)$$

Where

$$f(x_j) = A + Bx_j + Cx_j^2$$

A, B and C are estimated using Nonlinear least squares

Package Output

Throughout the analysis different data outputs are provided. These outputs will be saved in the parent directory of where the user saved their raw data, this is also where the gated data is saved. When the user gates the data, a *.csv* file, named *percentOfCellsGatedOut*, will be saved with information on the proportion of cells that have been gated out for each sample. The user can use *percentOfCellsGatedOut* as an indication of if a sample is messy (A sample can have a significantly greater proportion of cells gated out than the other samples), if the chosen gating parameters were chosen properly (there were no cells gated out, or the proportion was significantly large for all).

The main data output from the package is a *.csv* file called *ploidyPeaksOutput*, this is saved in a folder called 'analysis', in the same location as the *percentOfCellsGatedOut*. *ploidyPeaksOutput* contains all of the information from the peak detection analysis. Each row is a different sample. Column information is peak means (G1 and G2 peaks), peak heights, and includes the number of identified subpopulations. The user can also specify if they want similar information on doublets, whether to include an indicator of whether the sample should be investigated further by the user, which peak algorithm was used to analyze each sample, and the residual standard error value from the models.

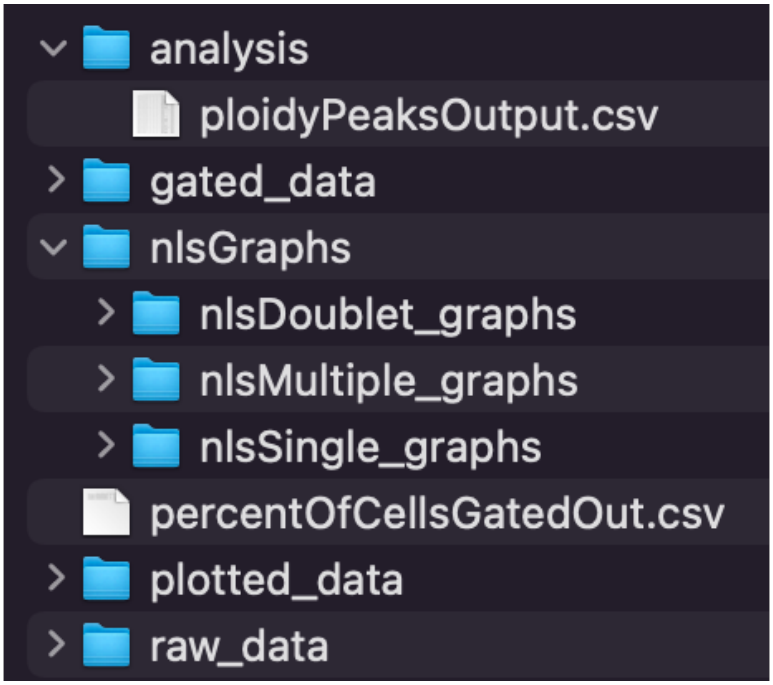

Figure A.12: The folder structure created by *ploidyPeaks*. The user creates the initial folder that contains the data they want to analyze (raw_data), this will be the main directory for the package. When using the gating functions, the package creates a folder for the gated data (gated_data), a folder for the plots comparing the gated and the raw data (plotted_data), and a *.csv* called *percentOfCellsGatedOut* with information on the percent of cells gated out for each sample. When analyzing the data with flowPeakDetection(), a *.csv* called *ploidyPeaksOutput* will be created in a folder called analysis. Additionally, a folder will be created to store the graphs for the models (nlsGraphs) with sub folders for each model (nlsSingle_graphs, nlsDoublet_graphs, nlsMultiple_graphs).

Walk Through

This is a walk through from the starting point of raw flow cytomtery data to the end of the analysis.

Step 1: Gating the data :

Prior to the analysis I put my data I a folder called 'raw_data'. Now I'm going to use `rectGateFlowFrame()` on the data in raw_data to test out which values to use to gate out the debris.

```
rectGateFlowFrame(
  rawDir = here("vignettes/data/raw_data"),
  flowName = "A2_10.fcs",
  xVariable = "FITC-A",
  yVariable = "SSC-A",
  xMinValue = 100,
  xMaxValue = 100,
  yMinValue = 400,
  yMaxValue = 700,
  savePlot = TRUE
)
```

A folder called 'plotted_data' will be created with visual outputs of the gate. From there we can see if we like the gate we applied.

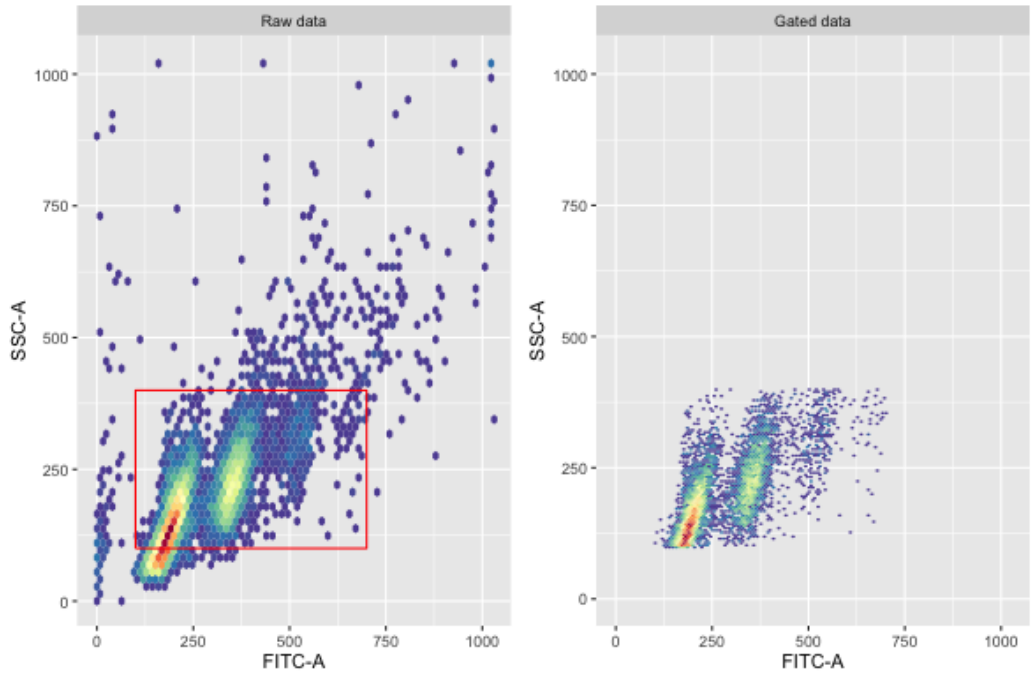

Figure A.13: Example of gating out too many cells using *rectGateFlowFrame()*

This looks like we removed too many cells, cells that belonged to the population, let's try other values:

```
rectGateFlowFrame(
  rawDir = here("vignettes/data/raw_data"),
  flowName = "A2_10.fcs",
  xVariable = "FITC-A",
  yVariable = "SSC-A",
  xMinValue = 50,
  xMaxValue = 50,
```

```
  yMinValue = 800,
  yMaxValue = 800,
  savePlot = TRUE
)
```

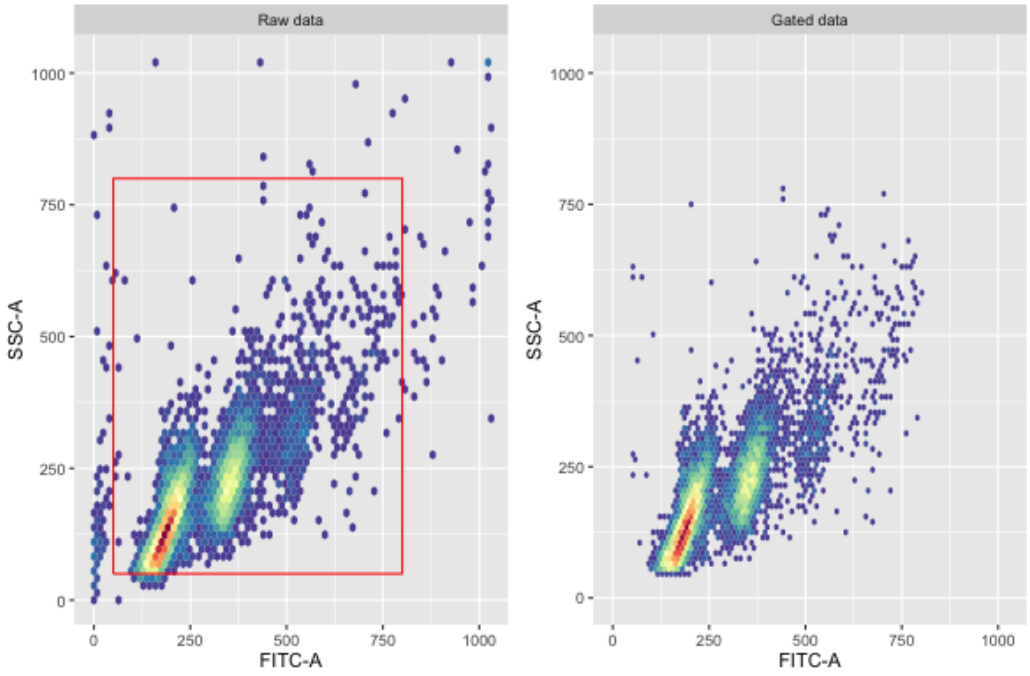

Figure A.14: Example of properly *rectGateFlowFrame()*

This looks pretty good, now let's apply it to our whole flow set.

```
rectGateFlowSet(
  rawDir = here("vignettes/data/raw_data"),
  xVariable = "FITC-A",
```

```
  yVariable = "SSC-A",
  xMinValue = 50,
  xMaxValue = 50,
  yMinValue = 800,
  yMaxValue = 800,
  savePlot = TRUE
)
```

Now that we gated the data, we can proceed to our analysis. The gated data is saved in a folder called 'gated_data', this will be the directory we will be using from now on.

Step 2: Peak detection Analysis

```
flowPeakDetection(
  flowDir = here("vignettes/data/gated_data"),
  xVariable = "FITC-A",
  doublet = FALSE,
  saveGraph = TRUE,
  singleThreshold = 8.5,
  usedCellsThreshold = 86
)
```

The analysis is completed, take a look at the outputs!