Vignette for measuring ploidy using scAbsolute

Shadi Shafighi

2023-05-23

Contents

1 Introduction

To apply scAbsolute on a dataset, you have two options:

1. The first option is to use the snakemake workflow provided in the GitHub repository. This workflow automates the process and ensures reproducibility.
2. The second option is to directly use the scAbsolute function and manually set the parameters. This approach is described in section 2 of this vignette, providing flexibility for customization.

To effectively utilize this vignette, it is crucial to ensure that your environment is properly configured to support its functionality. Due to the diverse dependencies involved, we highly recommend executing scAbsolute within Singularity. Singularity provides a consistent and reproducible environment for running the analysis.

However, for tasks such as quality control and result visualization (covered in Section 3 and 4), you may utilize the provided conda environments. These environments offer a convenient way to manage the necessary dependencies for these specific tasks.

2 Running scAbsolute

2.1 Setting up the environment

To set up the environment, follow these steps:

1. Install the Singularity container by following the quick steps provided in the Singularity documentation.
2. After installing Singularity, pull the Singularity image and run it using the following commands:

```
singularity pull IMAGE
singularity shell IMAGE
```

3. Once you're inside the Singularity shell, activate the Conda environment:

```
set +eu
. /opt/conda/etc/profile.d/conda.sh
conda activate conda_runtime
```

By following these steps, you will be in an environment with the necessary dependencies to proceed with the vignette.

2.2 Loading libraries and files

To utilize scAbsolute, it is necessary to load the required libraries as well as the essential functions. To achieve this, please set BASEDIR to the root folder of scAbsolute, which can be downloaded from the GitHub using "git clone https://github.com/markowetzlab/scAbsolute.git".

```
BASEDIR="~/scAbsolute/"
source(file.path(BASEDIR, "R/load_dependencies.R"))
load_dependencies(BASEDIR)
```

2.3 Naming convention

To ensure proper organization and easy identification of samples, we have implemented a consistent naming convention for our genomics data files. Our convention includes using the sample name, library technology, cell id and cell tag, all seperated by "_" in the .bam file names to indicate the specific cell type and experiment. We highly recommend that this naming convention is used for any data being analyzed in this context. Here is an example:

```
[unique identifier]_[library technology]_[cell id]_[cell tag].bam
UID-10X-Fibroblast-cell_SLX-00000_000001_AAACCTGAGCAGCCTC-1.bam
```

The "separate()" function from "tidyr" can be utilized to separate the names into four distinct parts.

```
tidyr::separate(name, sep="_", into=c("UID", "SLX", "cell_id", "cell_tag"), remove=FALSE)
```

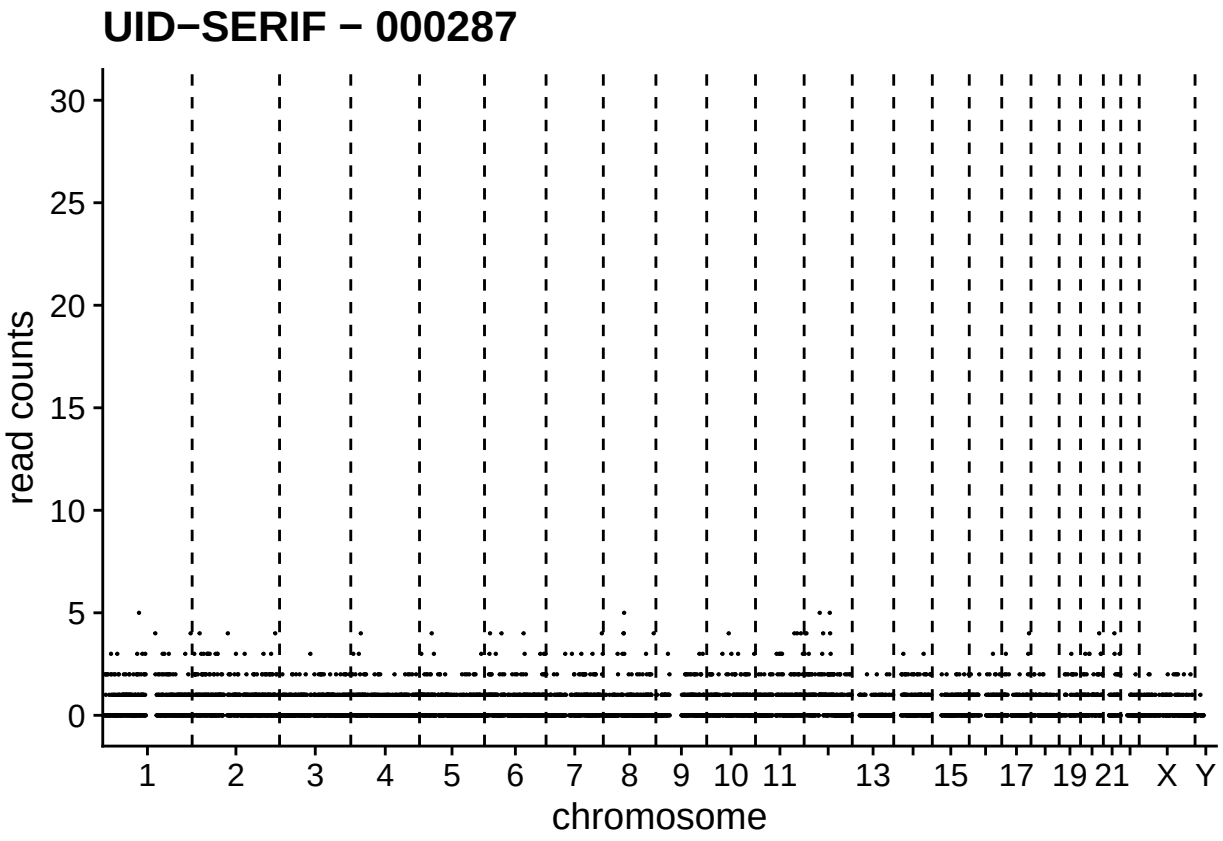
2.4 Pre-run quality control

It is important to remove cells with very low read numbers. This can be accomplished using samtools to obtain deduplicated and absolute read counts. Cells with read counts below 300,000 reads are not suitable and should be discarded.

```
samtools view -c -F 1804 file.bam
```

Another method for assessing the read numbers is by utilizing the "readData" function. In addition to that, the "plotCounts" function serves as a tool for analyzing the raw count data of individual cells. This function enables us to examine a specific cells read distribution in detail. To demonstrate the functionality of "plotCounts", we showcase an example of a cell that exhibits a low number of reads.

```
library(QDNAseq, quietly = TRUE)
library(ggplot2, quietly = TRUE)
BASEDIR="~/scAbsolute"
BAMFILES=c("~/data/align/UID-SERIF_SLX-00000_000285_R40-C69.bam",
           "~/data/align/UID-SERIF_SLX-00000_000287_R41-C03.bam")
source(file.path(BASEDIR, "R/core.R"))
source(file.path(BASEDIR, "R/visualization.R"))

counts <- readData(BAMFILES, binSize=500)
reads = Biobase::assayDataElement(counts[,2], "calls")
reads[is.na(reads)] <- 0
print(paste("Number of reads:",sum(reads)))
#> [1] "Number of reads: 2453"

plotCounts(counts[,2],copynumber=FALSE,ylab="read counts",ylim=c(0,30))
```



2.5 Running the scAbsolute algorithm

Here, we will explain some of the important parameters in detail to help users effectively use it for their CNV analysis. Firstly, we need to give the paths to the BAM files as the input. For this, we need a list of the paths to the BAM files in the filePaths variable which will be the first input of scAbsolute.

```
filePaths = ["data/UID-10X-Fibroblast-cell_SLX-00000_000001_AAACCTGAGCAGCCTC-1.bam",
             "data/UID-10X-Fibroblast-cell_SLX-00000_000001_AAACCTGAGCAGCCTC-1.bam"]
```

Next, we specify the genomic bin size in kbp. The supported values for this parameter are: 1, 5, 10, 15, 30, 50, 100, 500, 1000.

```
binSize = 500
```

Then, we need to indicate the species that we are working with and the reference genome, which we decided to use. For parameter "species", we support "Human" and "Mouse" and for parameter "genome", we support hg19 and hg38 regarding to the human species. We recommend using hg19 since the bin annotation has been optimized for it, using empirical data.

```
species = "Human"
genome = "hg19"
```

Next parameter is change_prob, which is the total probability of changing state in the segmentation. The segmentation algorithm divide the genome into segments, where each segment has a same underlying copy number. A high value will result in over segmentation, while a low value will produce more conservative segmentation. This value should be between 0 and 1 and change_prob=1e-3 is a conservative default.

```
change_prob=1e-3
```

For restricting the results using prior knowledge, scAbsolute accept "minPloidy" and "maxPloidy" as a hard limit on possible solution. Here are the default values:

```
minPloidy = 1.1
maxPloidy = 8.0
```

The 'splitPerChromosome' parameter is a flag that affects the speed and performance of the segmentation algorithm. By setting it to TRUE, the algorithm is parallelized across chromosomes with slightly reduced quality. Conversely, setting the flag to FALSE would increase run time specifically for small segments but with better quality. If the bin size is less than 500 kb, we recommend to set it to TRUE.

```
splitPerChromosome = FALSE
```

For setting the maximum number of copy number states, we use parameter "max_states".

```
max_states=10
```

When performing genomic analysis, it may be useful to restrict our analysis to specific chromosomes. This is particularly important when analyzing the sex chromosomes X and Y, as they tend to have less uniform coverage. In this case, we can use the "ploidyRegion" parameter, which accepts a character vector of chromosome names to be used for ploidy calculation. Also, we can use selectRegion parameter, which accept a character vector of chromosome names to select for processing.

```
ploidyRegion=paste0("chr", as.character(seq(1:22)))
selectRegion=paste0("chr", c(as.character(seq(1:22)), "X", "Y"))
```

Lastly, we need to specify the "outputPath" parameter, which should be set to the path where the final RDS output file will be saved. This RDS file will contain the result object for all the cells and serves as the output of the analysis.

After setting the parameters, we can use the scAbsolute wrapper function to execute the scAbsolute package for absolute copy number calling in single-cell DNA sequencing data. However, as mentioned before, this part of the vignette is not intended to be executed by the user and is included for illustrative purposes only.

```
scaledCN = scAbsolute(filePaths, binSize=500,
genome="hg19", minPloidy=1.1, maxPloidy=8.0,
ploidyRegion=paste0("chr", as.character(seq(1:22))),
selectRegion=paste0("chr", c(as.character(seq(1:22)), "X", "Y")),
splitPerChromosome=FALSE, change_prob=1e-3,
max_states=10, randomSeed=2023,
outputPath="results/500/out.rds")
```

3 Quality control

3.1 Loading packages

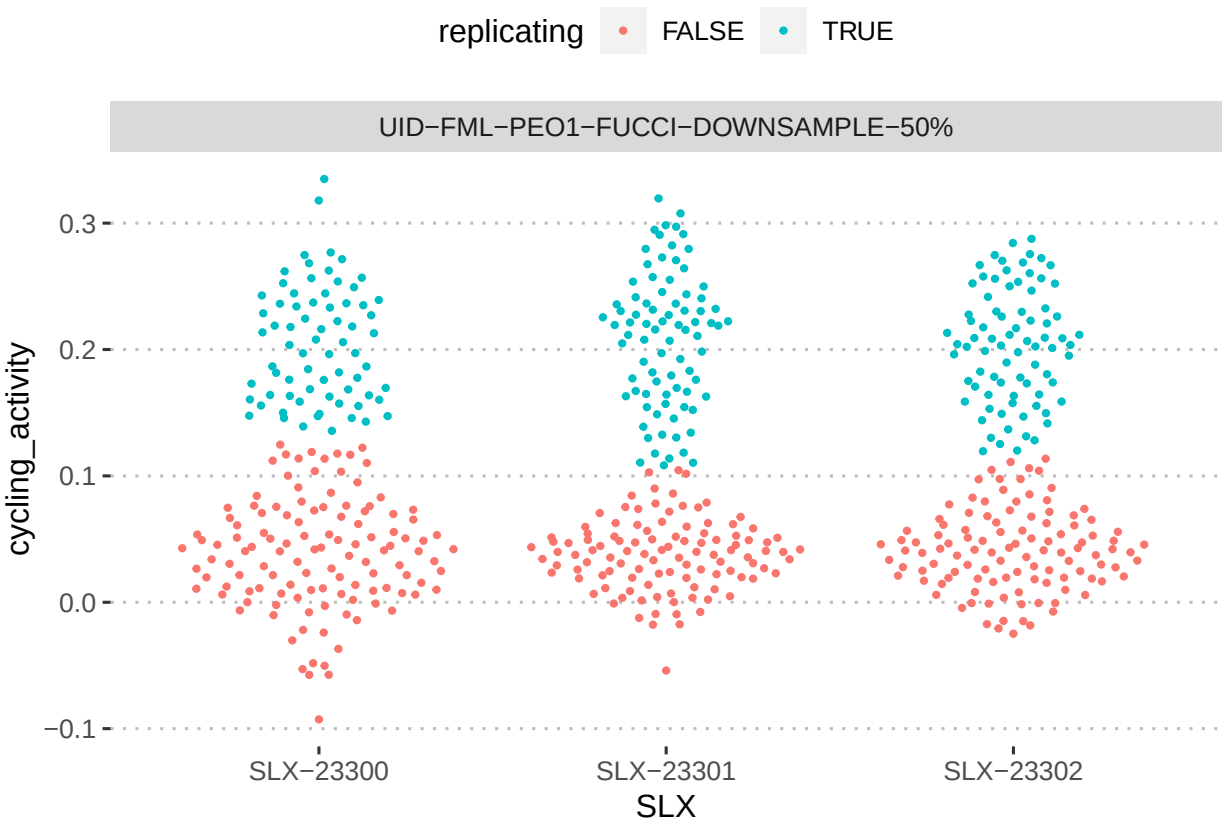
We recommend using conda environemt provided in the GitHub. But instead you can install the following packages and load the following r files from scAbsolute package. Besides, for running this part of the vignette, you'll need to set BASEDIR to the scAbsolute root folder, which you can download from GitHub.

```
library(QDNAseq, quietly = TRUE)
library(magrittr, quietly = TRUE)
library(dplyr, quietly = TRUE)
library(tidyr, quietly = TRUE)
library(ggplot2, quietly = TRUE)
library(ggbeeswarm, quietly = TRUE)
library("ggpubr", quietly = TRUE)

BASEDIR="~/scAbsolute"
ALLSAMPLES="~/scAbsolute/results/500/UID-FML-PEO1-FUCCI-DOWNSAMPLED-EQUAL-FORDLP_500.rds"
source(file.path(BASEDIR, "R/core.R"))
source(file.path(BASEDIR, "R/visualization.R"))
```

3.2 Predicting replicating cells

Here, using "predict_replicating" function, we predict the cycling status and update the metadata of the output object with the predicted values. For this prediction, we need to set the cutoff_value for outlier exclusion. Here, we set it to "cutoff_replicating = 1.5".

```
object = readRDS(ALLSAMPLES)

df = predict_replicating(Biobase::pData(object) %>%
  tidyr::separate(name, sep="_", into=c("UID", "SLX", "cellid", "celltag"), remove=FALSE)
  ,cutoff_value=1,iqr_value = 1.5)
df = df[df['UID']== "UID-FML-PEO1-FUCCI-DOWNSAMPLE-50%",]
fig_cellcycle = ggplot(data = df) + geom_quasirandom(aes(x=SLX, y=cycling_activity,
  color=replicating), size=0.8, alpha=1.0) +
  facet_wrap(~UID, scales = "free_x") + theme_pubclean()
fig_cellcycle
```



In the figure, the blue dots represent cells with a replicating status. These cells need to be removed from our further analysis. By decreasing the cutoff_value, we can adopt a more stringent criteria for exclusion, resulting in the removal of a greater number of cells from our analysis.

4 Visualization

4.1 Required files for running the vignette

To run this part of the vignette, you'll need to set some initial values. First, set BASEDIR to the scAbsolute root folder, which you can download from GitHub. Second, set ALLSAMPLES to the RDS results from scAbsolute containing the results of all the samples. Finally, set OUTPUT to the existing directory where you want to store the generated PDF file.

If you don't have your own single-cell data to work with, you can use toy single-cell samples. These samples, along with the results of running scAbsolute on them, are available in Google Drive.

```
BASEDIR="~/scAbsolute"
ALLSAMPLES="~/scAbsolute/results/500/UID-FML-PEO1-FUCCI-DOWNSAMPLED-EQUAL-FORDLP_500.rds"
OUTPUT= "~/scAbsolute/figures/test"
```

4.2 Loading packages

We recommend using conda environemt provided in the GitHub. We firstly need to load the QDNAseq package and essential files from scAbsolute package.

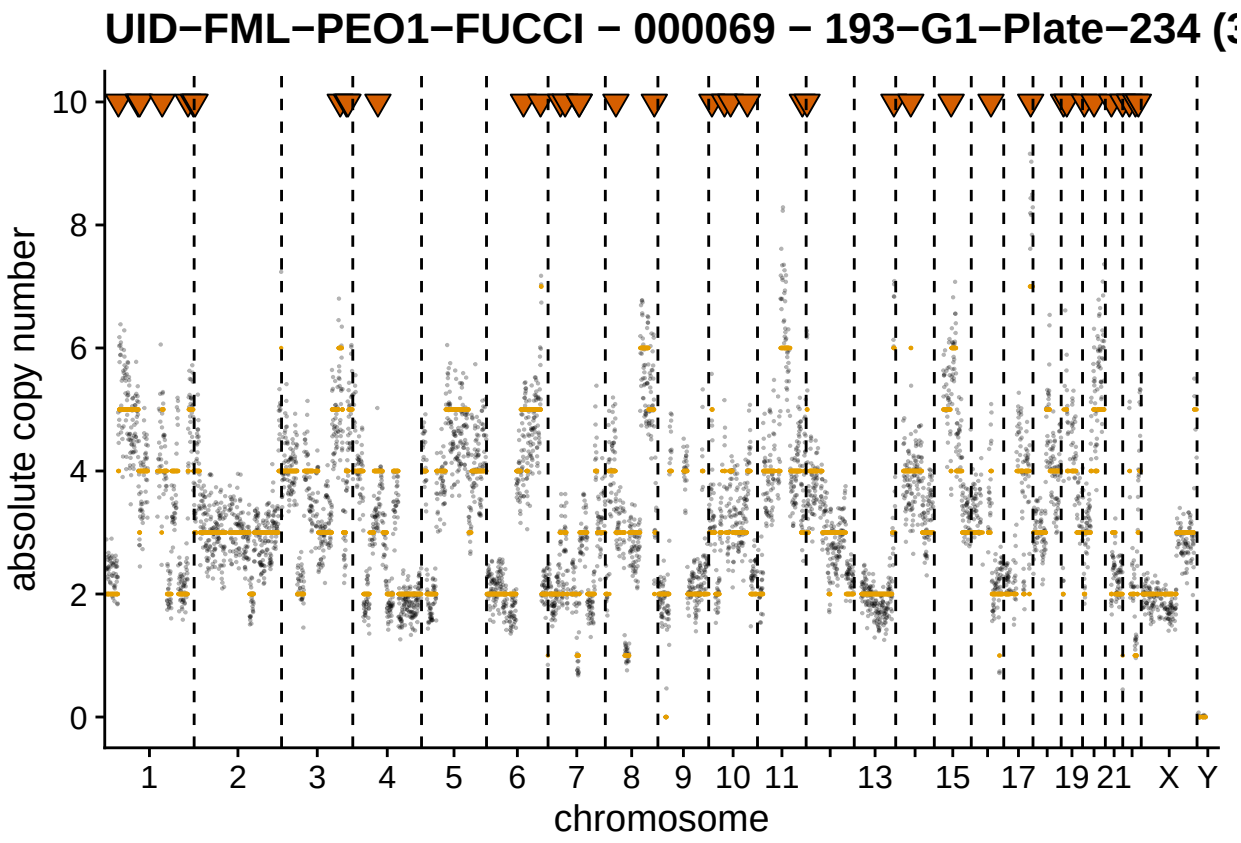
```
library(QDNAseq, quietly = TRUE)
library(magrittr, quietly = TRUE)
library(dplyr, quietly = TRUE)
```

```
source(file.path(BASEDIR, "R/core.R"))
source(file.path(BASEDIR, "R/visualization.R"))
```

4.3 Plotting single cell copy number profile

The only required input for plotting the copy number is a QDNAseq object for at least a single sample. This can be generated by running the scAbsolute package on the single cell BAM files. We load one of the sample's rds file in the results folder to CN. Then, we can select the cell object by CN[, cell_name] and simply plot it. Instead of cell name we can use index as well.

```
CN = readRDS(ALLSAMPLES)
plotCopynumber(CN[,163],verbose = FALSE)
```



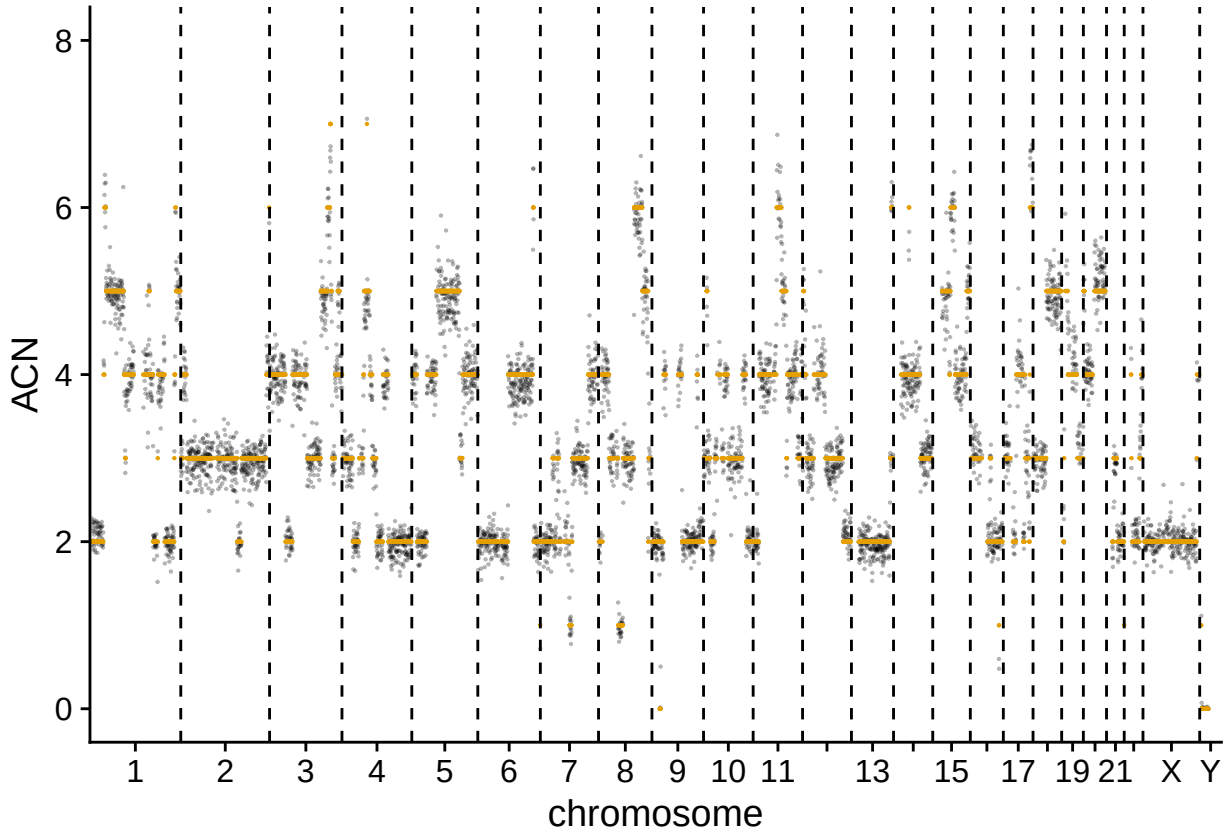
4.4 Parameters

The function plotCopynumber allows for customization of several parameters to adjust the appearance of the plot and highlight specific regions of interest. Setting the parameter "correction" to TRUE will correct for technical biases in the read depth signal that are correlated with GC content and mappability. The y-axis limit can also be adjusted to a desired value using the "ylim" parameter.

By default, the plot includes read information alongside the title. To remove this information, "readInfo" can be set to FALSE. The default title is the name of the sample and the y label is "absolute copy number", but these can be changed using the "main" and "ylab" parameters. Furthermore, unique events are indicated by a triangle symbol by default, but this symbol can be removed by setting "showMarker" to FALSE.

To plot the copy number with the altered parameters, simply call the plotCopynumber function with the desired parameter values.

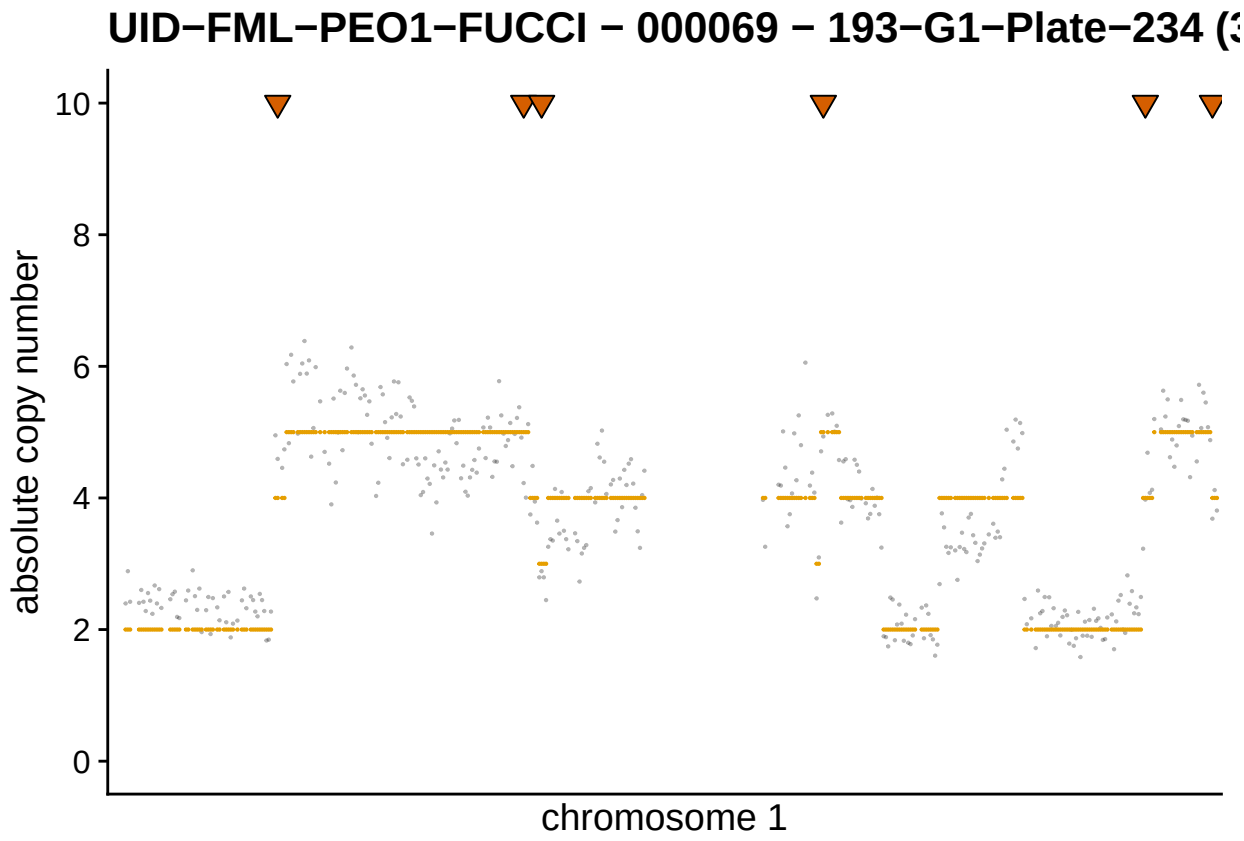
```
plotCopynumber(CN[, 3], correction=TRUE, ylim=c(0, 8),
               readinfo=FALSE, main="", ylab="ACN", showMarker=FALSE,verbose = FALSE)
```



4.5 Plotting a chromosome

Here, we want to select the rows from a data frame where the row names start with "1:", which are corresponding to chromosome 1. The resulting subset is then plotted using plotCopynumber.

```
chr1 = startsWith(x = rownames(CN), prefix="1:")
plotCopynumber(CN[chr1, 163],verbose = FALSE)
```



4.6 Plotting the entire set of cells

For plotting the entire set of cells together, we can use "plotDataset" function. For this, we need to load the scAbsolute result file (out.rds), which contain the results of all the input samples. We also need to define the name of the output in the desired existing folder using "file" parameter.

```
plotDataset(object=CN, file=paste0(OUTPUT,"/out.pdf"),
            f=plotCopynumber,verbose=FALSE)
```

This will generate a pdf files "out-1.pdf" in the direction of OUTPUT.