Measuring per-cell ploidy and replication status using *scAbsolute*

Shadi Shafighi, Michael Schneider

2024-02-16

Contents

1 Introduction

1.1 Running the first part of the workflow (per-cell analysis)

To apply *scAbsolute* on a data set, you have two options:

1. The first option is to use the snakemake workflow provided here scDNAseq-workflow. This workflow automates the process and ensures reproducibility.
2. The second option is to directly use the docker images provided and mount *scAbsolute* and *scUnique* manually inside the containers. This approach is described in section 2 of this vignette and provides additional flexibility for customization.

1.2 Interactive analysis and post-processing

While we recommend running the steps described in *scAbsolute* and *scUnique* manuscripts via the snakemake pipeline or at least using the docker images, it is convenient to conduct the analysis, visualization and any post-processing in a custom conda environment with less dependencies. We provide one such environment at `envs/copynumber.yml`. Additional dependencies should be loaded at the start of every R session running inside this environment, via

```
BASEDIR="~/"
source envs/dependencies.R
```

1.3 Making changes to the source code (advanced use)

If you wish to customize the source code of the packages, you can directly run the container images with the updated source code mounted inside the containers. Make sure to load the conda environment installed inside the containers using the below command, to have access to the software installed in the containers.

```
. /opt/conda/etc/profile.d/conda.sh
conda activate conda_runtime
```

1.4 Naming convention

To ensure proper organization and consistency with our visualization functions, we identify samples via a unique sample identifier (UID) and a library identifier (SLX). These two variables are required and automatically derived as part of the qc script from the file names.

We suggest to follow the naming convention described below. Otherwise, you will have to add these two variables manually at the start of the qc script for all samples.

In addition to UID and SLX, we also add a running six digit cell number and a cell tag containing any additional information (such as Plate number and well position, or sequencing tags and experimental information). The four fields are separated by an underscore. Here are some example file names following the convention:

```
[unique sample identifier]_[library identifier]_[cell id]_[cell tag].bam
UID-10X-Fibroblast-cell_SLX-00000_000001_AAACCTGAGCAGCCTC-1.bam
UID-CIOV1_SLX-17170_000001_AAAGCAAGTAGAACAT-1
UID-FML-PEO1-2N_SLX-23003_000288_SINCEL-194-Plate-236-P21-CGACAAGGAT-AAGGCCACCT.bam
```

You will notice that we directly use this naming scheme to obtain this information from the cell name at various locations throughout the code.

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

1.5 Initial quality controls

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 200,000 - 300,000 reads are not suitable for being included in the analysis and should be discarded. You can use the below command to obtain read counts per cell.

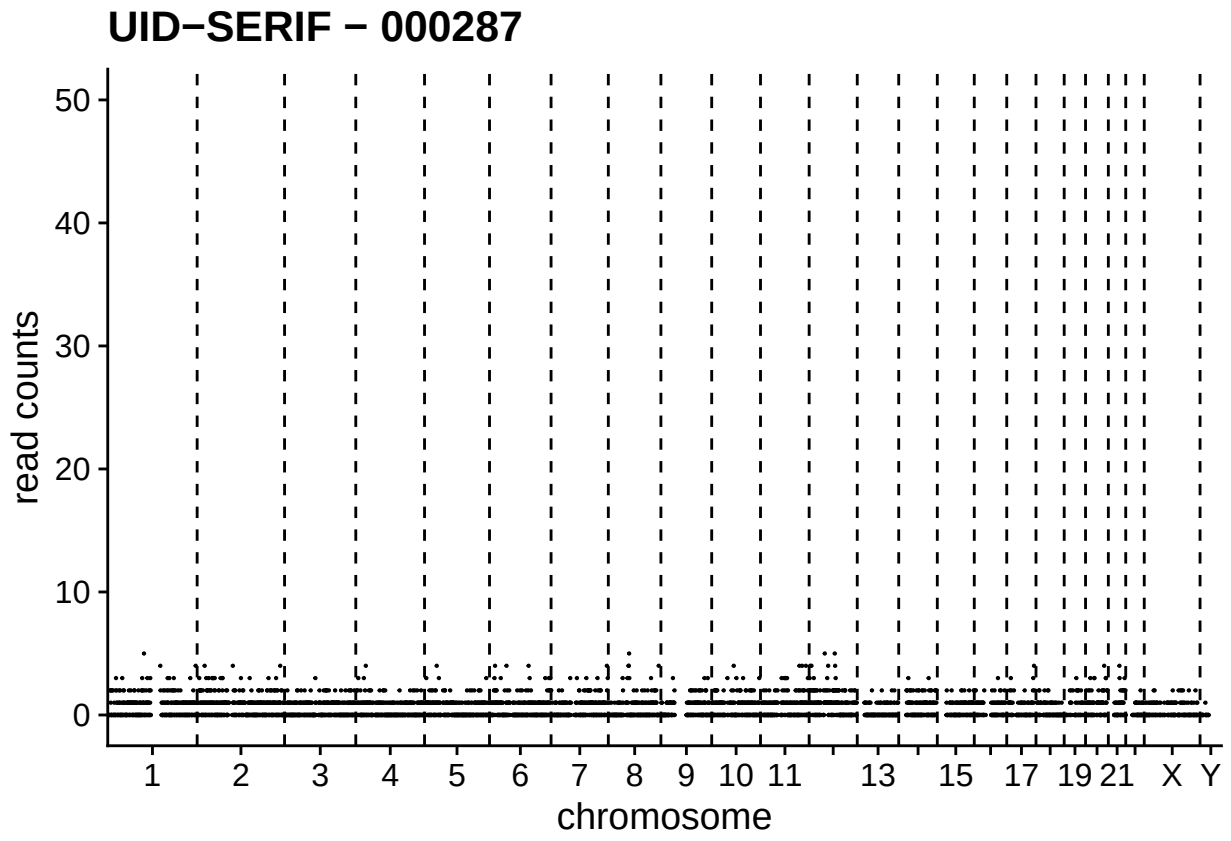
```
samtools view -c -F 1804 file.bam
```

Another method for interactively and visually assessing the quality of individual cells without running the pipeline is by utilizing the "readData" function to load the data from a single-cell bam file without any further processing. You can use the "plotCounts" function on the readData object to inspect individual cells that fail the analysis. Here is an example of a cell that exhibits a very low number of reads.

```
BASEDIR="~/"
WORKFLOW_PATH="~/scDNAseq-workflow/"
source(paste0(WORKFLOW_PATH, "envs/dependencies.R"))
#> [1] "BASEDIR is set to: ~/"
BAMFILES=c(paste0(WORKFLOW_PATH, "vignettes/data/UID-SERIF_SLX-00000_000287_R41-C03.bam"))

BASEDIR="~/scAbsolute/"
counts <- readData(BAMFILES, binSize=500)
valid_reads = binsToUseInternal(counts)
reads = Biobase::assayDataElement(counts[valid_reads, ], "calls")
print(paste("Number of reads:", apply(reads, 2, sum, na.rm=TRUE)))
#> [1] "Number of reads: 2453"
```

```
plotCounts(counts[,1], copynumber=FALSE, ylab="read counts", ylim=c(0,50))
```



1.6 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 = paste0(WORKFLOW_PATH, c("data/aligned/Trial_Human/UID-10X-Fibroblast-cell_SLX-00000_000001_
  "data/aligned/Trial_Human/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, 200(for hg19 only), 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 for human samples, and mm10 for mouse samples. We strongly recommend using hg19 for human samples since the bin annotation has been optimized for it using empirical data.

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

Next parameter is change_prob, which is a proxy for the 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 of the ploidy inference using external knowledge, *scAbsolute* accepts "minPloidy" and "maxPloidy" as a hard limit on possible solutions. Here are the default values, but they can be adapted flexibly either via additional columns in the sample.tsv file (see e.g. config/sample_PEO1.tsv) or directly in the script.

```
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 segemntation 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". Higher values than max_states=8 are currently incompatible with the MEDICC2 implementation.

```
max_states=8
```

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. We recommend for the purpose of determining recent copy number aberrations to exclude the Y chromosome in all cases and the X chromosome by default. Both sex chromosomes are considerably less well mapped and the counts are consequently less reliable.

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

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", c(as.character(seq(1:22)), "X")),
  selectRegion=paste0("chr", c(as.character(seq(1:22)), "X")),
  splitPerChromosome=FALSE, change_prob=1e-3,
  max_states=8, randomSeed=2023,
  outputPath="results/500/exampleCells.rds")
```

1.7 Quality control

1.7.1 Loading packages

We recommend using the conda environment provided in the GitHub. In addition, we load the visualization functionality from the *scAbsolute* package that needs to be locally checked out for this purpose.

```
BASEDIR="~/"
source(paste0(WORKFLOW_PATH, "envs/dependencies.R"))
```

```
#> [1] "BASEDIR is set to: ~/"
object=readRDS(paste0(WORKFLOW_PATH, "vignettes/data/UID-FML-PEO1-FUCCI-DOWNSAMPLED-EQUAL-FORDLP_500.rds
```

1.7.2 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".

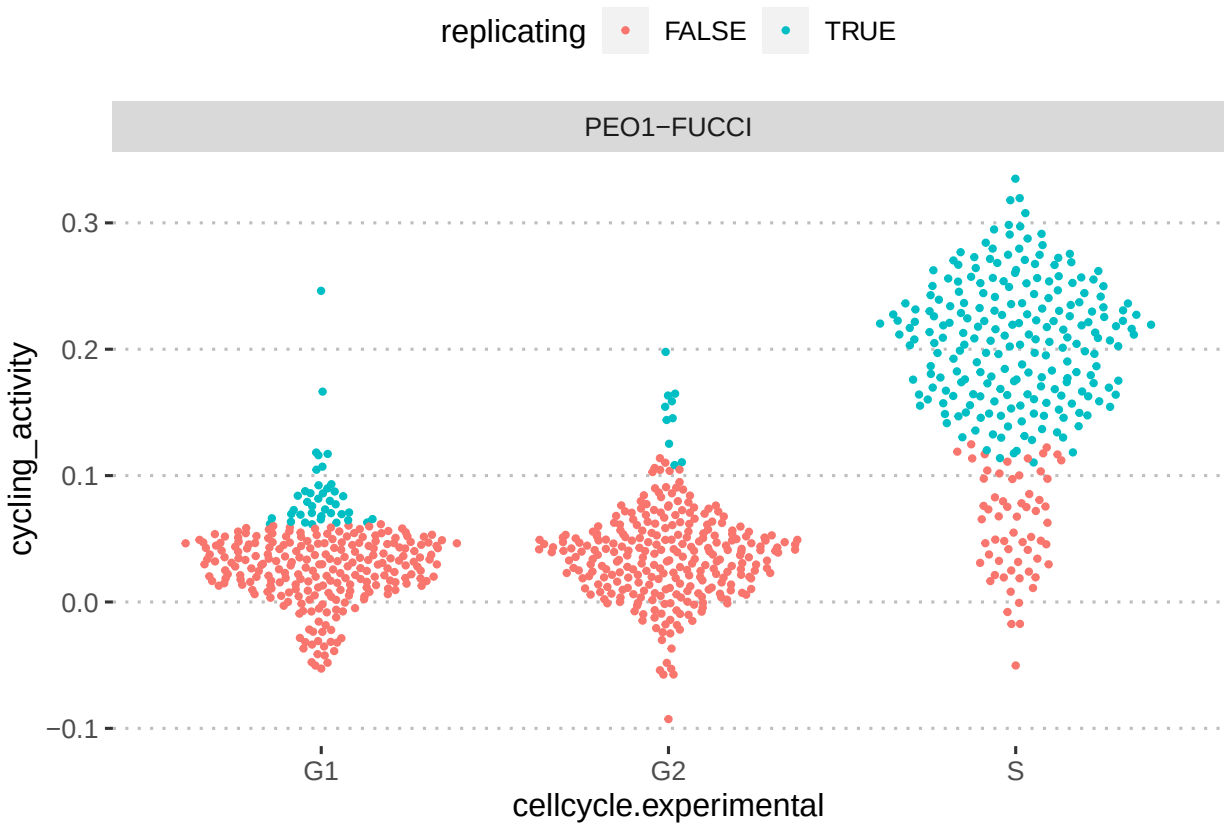
```
#source("~/scAbsolute/R/core.R") # load predict_replicating function

df = predict_replicating(Biobase::pData(object) %>%
  tidyr::separate(name, sep="_", into=c("UID", "SLX", "cellid", "celltag"), remove=FALSE) %>%
  tidyr::separate(celltag, into=c("A", "B", "cellcycle.experimental", "D"), sep="-", extra="drop"), cut
  dplyr::mutate(UID = "PEO1-FUCCI")
# this data set contains cells downsampled to identical read depth
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
```

By varying the cutoff value, we can vary the number of cells excluded resulting in a greater number of correctly excluded cycling cells, but at the same time also the removal of some false positives. We can compare our analysis with the experimental cell cycle annotation available for this set of cells, which is not perfect either.

```
fig_cellcycle_eval = ggplot(data = df) + geom_quasirandom(aes(x=cellcycle.experimental, y=cycling_activi
  color=replicating), size=0.8, alpha=1.0) +
```

```
  facet_wrap(~UID, scales = "free_x") + theme_pubclean()
fig_cellcycle_eval
```



2 Visualization

2.1 Loading packages and samples

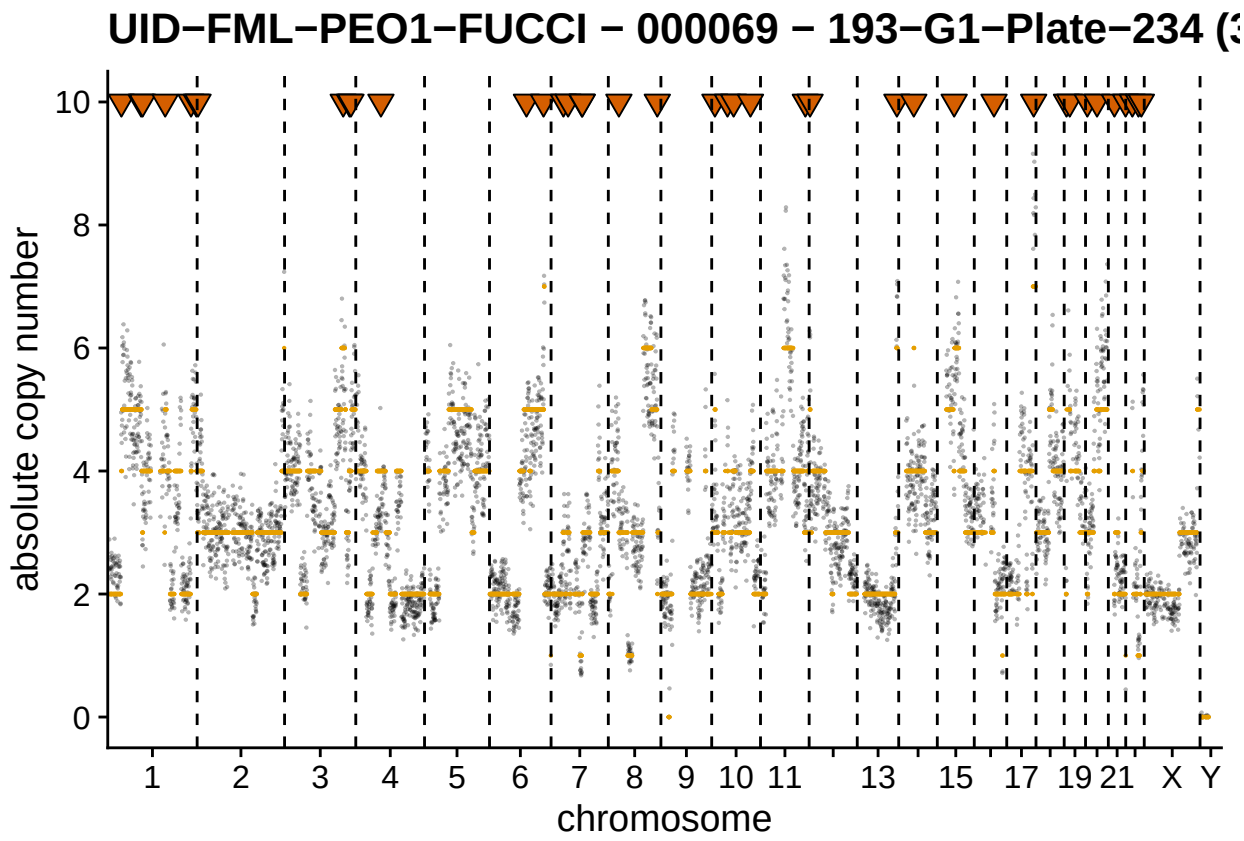
Again, we recommend to use the conda environment provided in envs/copynumber.yml to run this part of the analysis.

```
source(paste0(WORKFLOW_PATH, "envs/dependencies.R"))
#> [1] "BASEDIR is set to: ~/"
CN = readRDS(paste0(WORKFLOW_PATH, "vignettes/data/UID-FML-PEO1-FUCCI-DOWNSAMPLED-EQUAL-FORDLP_500.rds")
```

2.2 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.

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



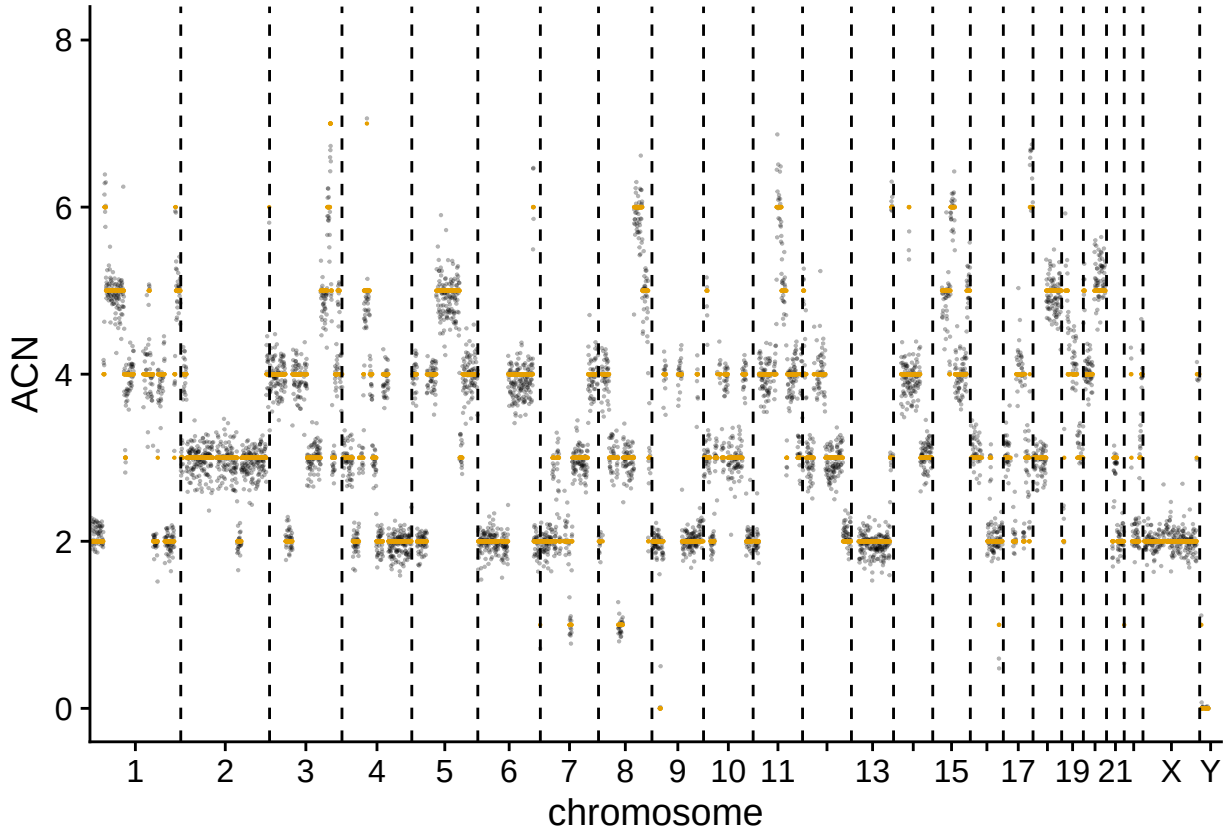
2.3 Parameters

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, changes in segmentation between single and joint copy number analysis 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.

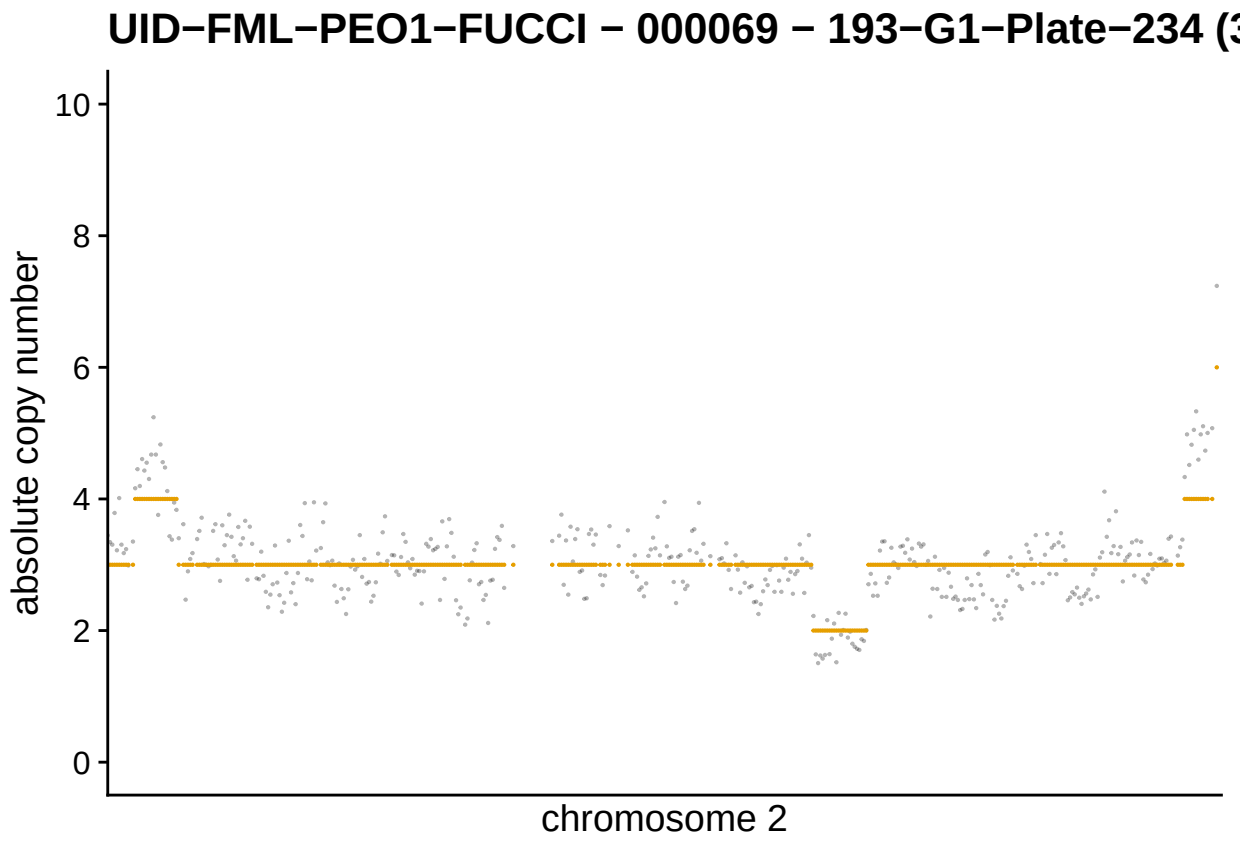
```
plotCopynumber(CN[, 3], correction=TRUE, ylim=c(0, 8),
               readinfo=FALSE, main="", ylab="ACN", showMarker=FALSE,verbose = FALSE)
```



2.4 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.

```
chr2 = startsWith(x = rownames(CN), prefix="2:")
plotCopynumber(CN[chr2, 163], verbose = FALSE, showMarker = FALSE)
```



2.5 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="~/examplePlotting.pdf",
            f=plotCopynumber, verbose=FALSE)
```

This will generate a pdf files "examplePlotting.pdf".