Vignette for running the scDNAseq pipeline to correctly identify single cell ploidy and cases of potential WGD

Michael Schneider

2024-04-04

Contents

1 Introduction

Here, I will run you through the basic ploidy analysis of a cell line sample and subsequent analysis of cells for WGD. As an example, and to show that the method is indeed working, we use a cell line sample with known ploidy status (PEO1-FUCCI cell line with FUCCI cell cycle marker), and we demonstrate how we can sucessfully recover most of the cells' ploidy.

Please use the snakemake workflow provided in the GitHub repository for your analysis. This workflow automates the process and ensures reproducibility.

In order to run the pipeline, you will require a working snakemake (tested with version 8) and singularity installation. Input bam files should be coordinate-sorted and duplicate-marked. In addition, the scAbsolute git directory should be cloned to your home directory (by default "~/"). You can also modify the directory via the BASEDIR variable and via the singularity mount options when executing the containers. We also highly recommend using our naming scheme for the reads, described in the basic vignette. Every name consists of four parts, separated by an underscore.

```
[unique sample identifier]_[library identifier]_[running cell id]_[cell tag/all other information].bam
UID-10X-Fibroblast-cell_SLX-00000_000001_AAACCTGAGCAGCCTC-1.bam
```

2 PEO1-FUCCI example (known ploidy)

2.1 Loading libraries and data for interactive analysis

We recommend using the copynumber conda environment provided in config/envs.yml. To visualize results and conduct the quality controls, we first need to load dependencies.

```
  WORKFLOW_PATH="~/scDNAseq-workflow/"
  BASEDIR="~/"
  source(paste0(WORKFLOW_PATH, "envs/dependencies.R"))
#> [1] "BASEDIR is set to: ~/"
```

```
df_init = dplyr::bind_rows(lapply(
  c("vignettes/data/PEO1-FUCCI-example_downsampled_and_subset.rds"), function(x){
    return(Biobase::pData(readRDS(paste0(WORKFLOW_PATH, x))))
})) %>%
  tidyr::separate(name, sep="_", into=c("UID", "SLX", "cellid", "celltag"), remove=FALSE) %>%
  tidyr::separate(celltag, sep="-", into=c("A", "B", "cellcycle_measurement"),
                  extra="drop", remove=FALSE) %>%
  dplyr::mutate(UID2 = gsub("UID-FML-", "", UID), SLX=gsub("SLX-", "", SLX), UID="PEO1-FUCCI")
```

2.2 Initial analysis

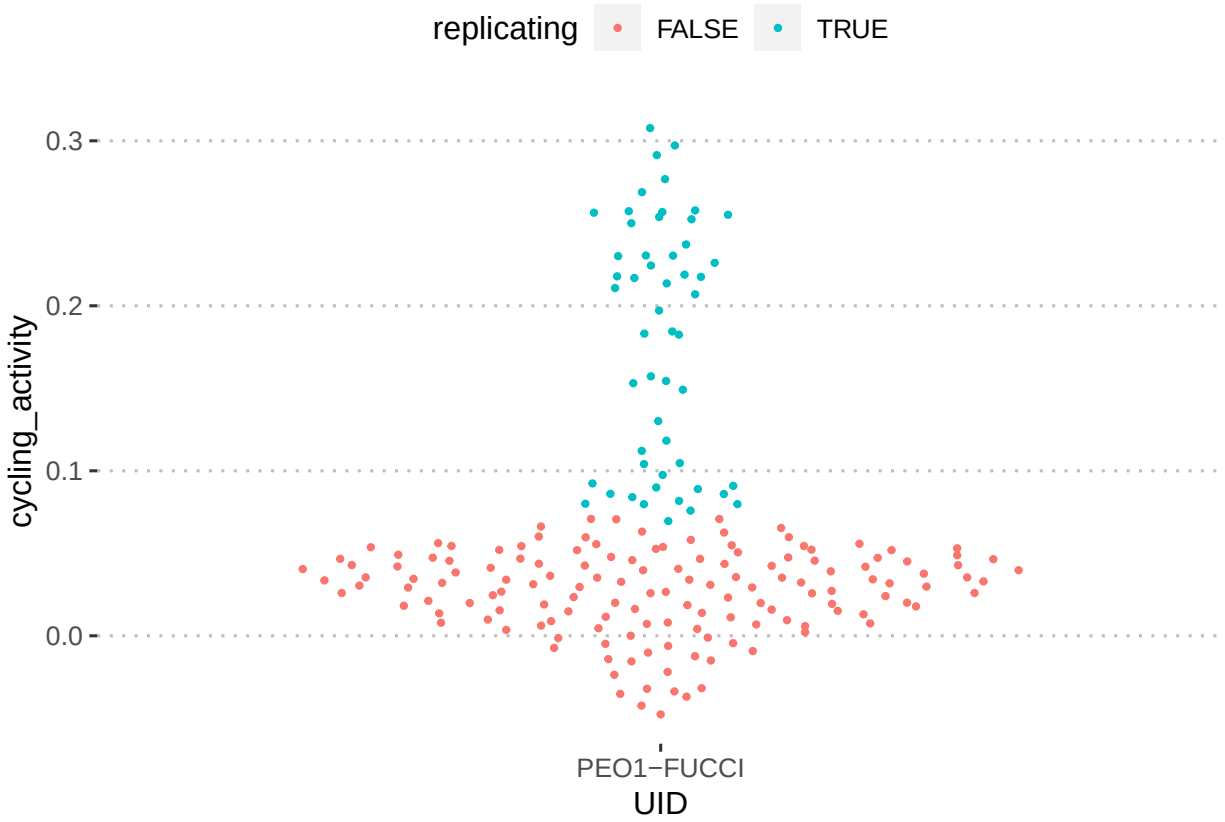
We initially run the first part of the scDNAseq workflow with a ploidy window of 1.1 to 8.0. This corresponds to the default pipeline settings, more fine-grained per-cell ploidy constraints can be added as shown in config/sample_PEO1.tsv (last two columns).

Initially, we check for the read depth of our cells. For any ploidy analysis to be reliable, **we recommend a read depth of rho=25 (rho is represented as rpc in the code) and greater**. Samples with read depth lower than rho=25 are not reliably segmented by PELT, so ploidy estimates tend to show higher error rates and we would recommend using larger bin size to analyse the data or alternatively deeper sequencing.

We directly remove cells that are predicted to be replicating. We would expect ploidy predictions for these cells to be relatively unreliable, as transient copy number states due to partial genome amplification will affect the estimation procedure.

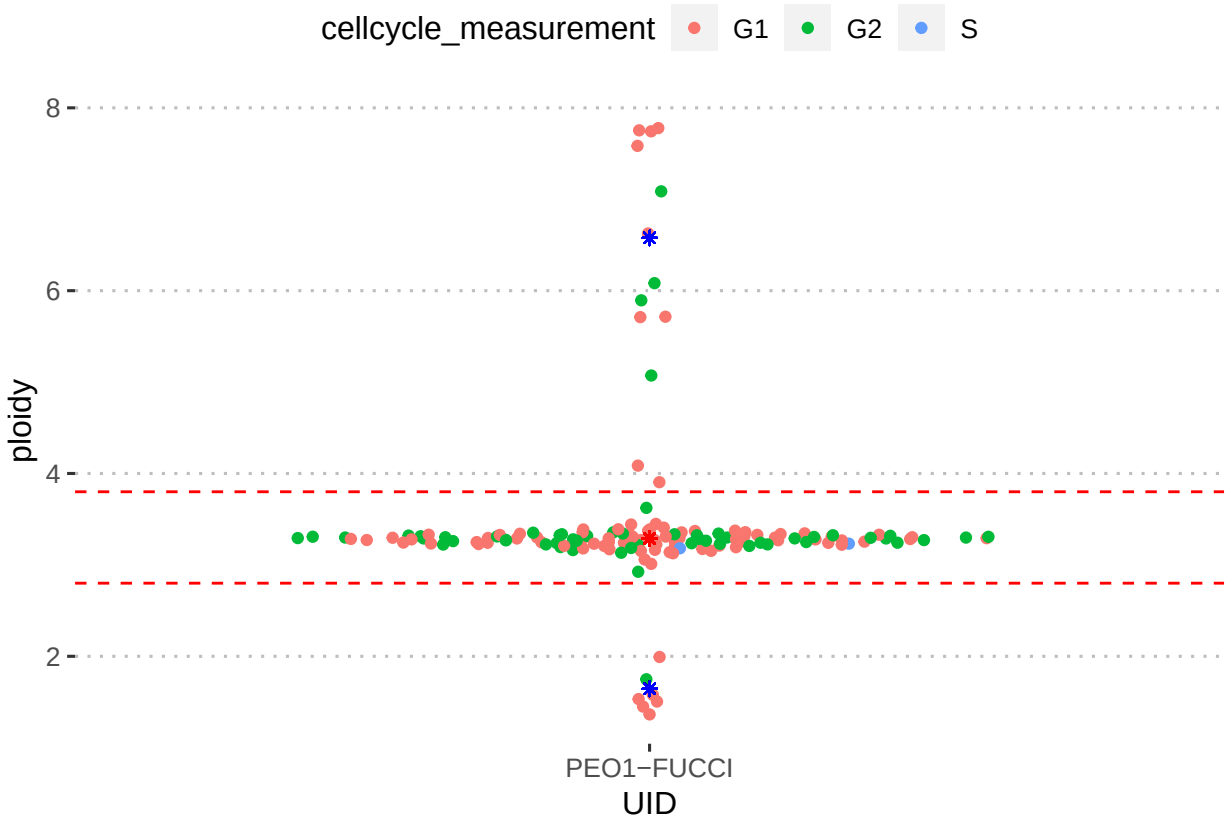
As you can see, we manage to detect most of the S phase cells (38/40), but we also have some false positives. The classification threshold can be tuned via the cutoff_value variable in the predict_replicating function call. This value directly affects the cutoff as visible in the below plot. Additionally, you can also adjust the iqr_value parameter (this parameter has less of an effect, though, and affects another variable indicative of cells undergoing replication).

```
summary(df_init$rpc)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   48.24  173.54  217.35  207.40  237.53  538.22

df_post = predict_replicating(df_init, cutoff_value = 1.0, iqr_value = 1.5)

table(df_post$replicating, df_post$cellcycle_measurement)
#>
#>         G1 G2  S
#>   FALSE 89 56  2
#>   TRUE  11  4 38
fig_cellcycle = ggplot(data = df_post) + geom_quasirandom(aes(x=UID, y=cycling_activity,
  color=replicating, name=name), size=0.8, alpha=1.0) +
  theme_pubclean()
fig_cellcycle
```



We run the subsequent analysis only on the cells that are not replicating according to our prediction. We observe a relatively consistent ploidy estimation. Note that the initial estimation cannot distinguish between G1/G2 or G1/WGD cells, they are both assigned to the same main ploidy estimate of around 3.3. It is also common to observe some cells at 2 and 1/2 the values of the main ploidy estimate (as indicated by the blue stars). Often these cells are just erroneously assigned these ploidy states, as it is difficult to always distinguish these cases. Note that estimates tend to become more accurate with higher read depth and smaller bin sizes.

```
df = df_post %>% dplyr::filter(!replicating)

median_ploidy = median(df$ploidy)
median_ploidy
#> [1] 3.289039
fig_ploidy = ggplot(data = df) + geom_quasirandom(aes(x=UID, y=ploidy, color=cellcycle_measurement)) +
  geom_point(aes(x=UID, y=median_ploidy), color="red", shape=8) +
  geom_point(aes(x=UID, y=0.5*median_ploidy), color="blue", shape=8) +
  geom_point(aes(x=UID, y=2.0*median_ploidy), color="blue", shape=8) +
  geom_hline(yintercept=c(2.8, 3.8), color="red", linetype="dashed") +
  theme_pubclean()
fig_ploidy
```




```
prop.table(table(df$ploidy > 2.8 & df$ploidy < 3.8))
#>
#>     FALSE      TRUE
#> 0.1360544 0.8639456
```

2.3 Iterative ploidy estimation

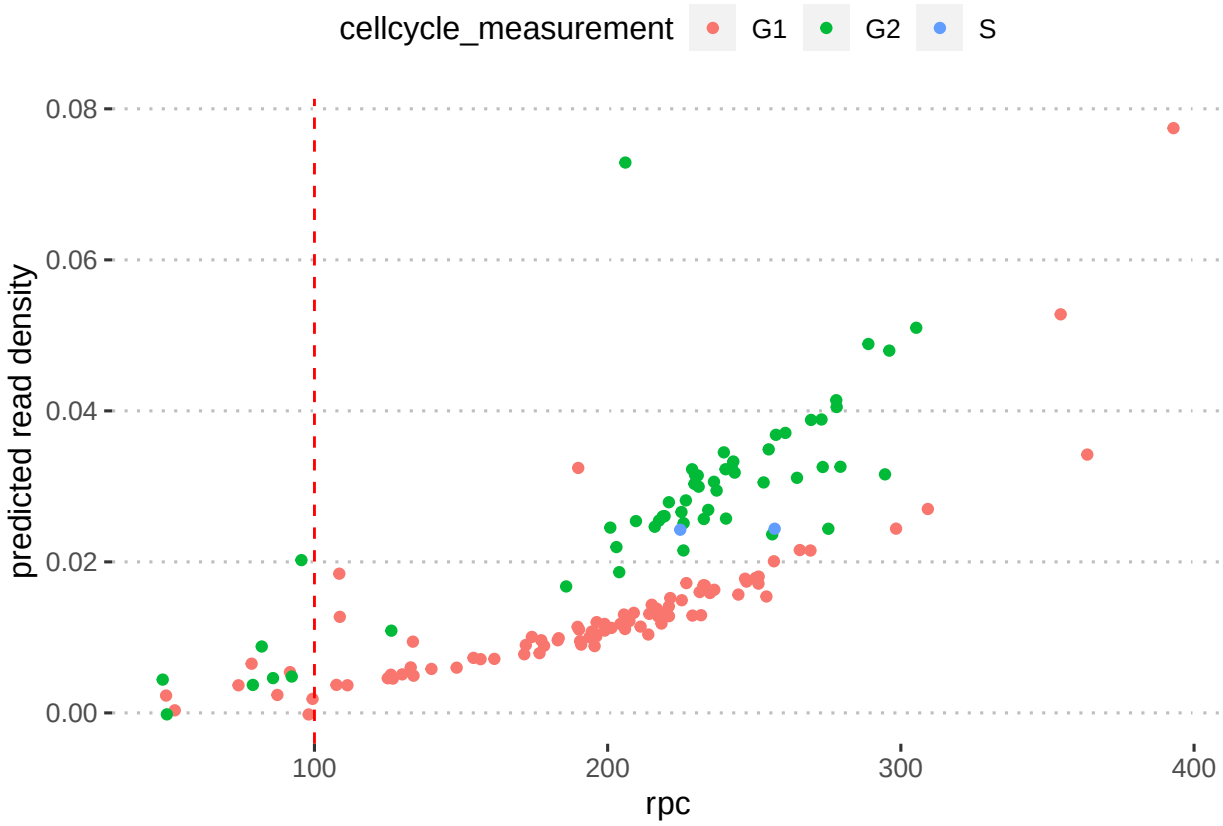
We observe that about 14% of cells have been fitted outside of the main ploidy window (2.8-3.8). We now have multiple options. 1. We can assume the cells with differing ploidy are potential outliers, and remove them from further analysis. 2. We can refit all the cells with the ploidy window (usually median ploidy +- 0.5 for unimodal distributions) 3. We assume the outliers represent true ploidy variation. This approach makes sense in the case of estimates that are not multiples of 2 or where the number of cells is very substantial and indicates true ploidy heterogeneity. In this case, one could treat the subpopulations separately.

Here, we are faced with a cell line and a very homogeneous ploidy estimation, so I would recommend approach 2, i.e. refit the cells with a ploidy window of 2.8-3.8. This requires re-running the first part of the pipeline with an updated per-cell ploidy constraint.

2.4 WGD/G2 cells

Finally, we are generally interested in cells that have recently undergone a whole genome doubling or just want to make sure we correctl exclude G2 cells from the subsequent analysis. In the **scAbsolute** manuscript, we have introduced the concept of read density to distinguish these kinds of cells. The measure of read density is encoded in the prediction variable (as we predict an estimate of read density at a copy number state of 2 for all cells).

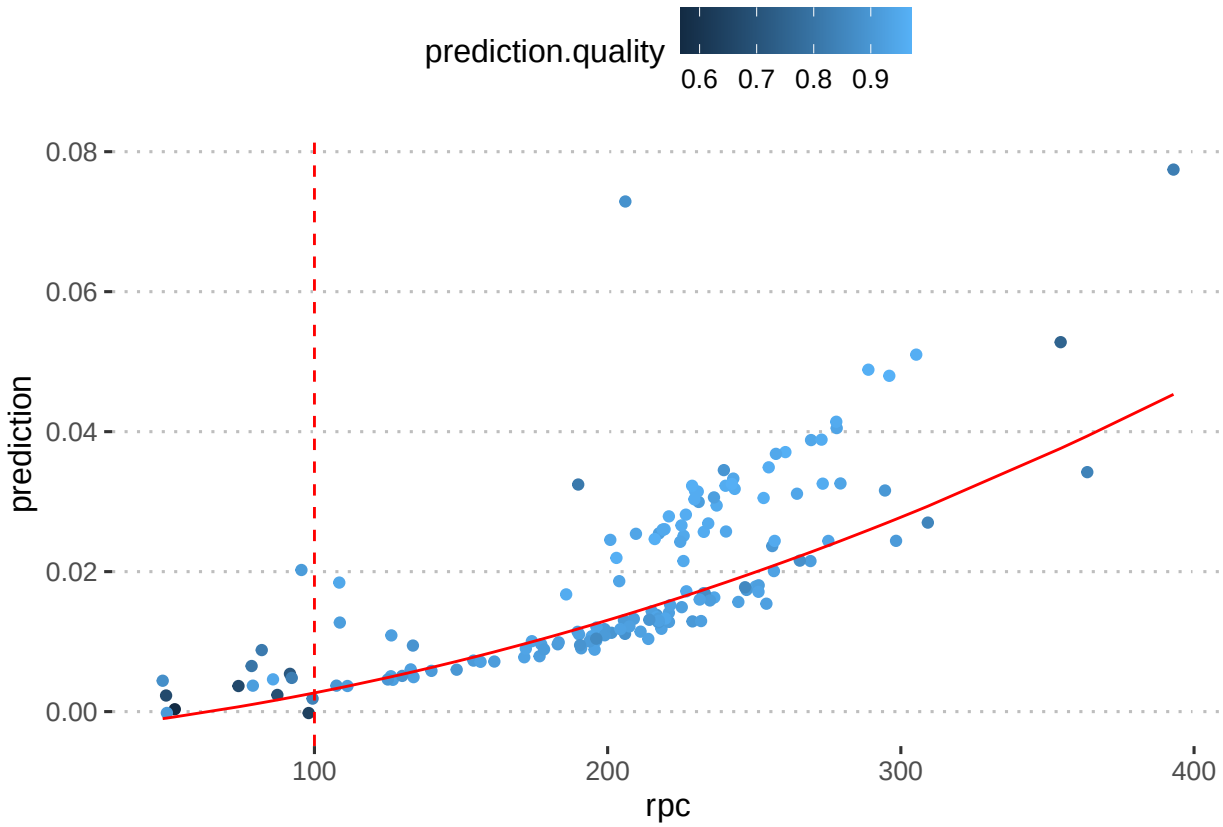
```
fig_density_annotated = ggplot(data = df) + geom_point(aes(x=rpc, y=prediction, color=cellcycle_measurer
  geom_vline(xintercept=100, color="red", linetype="dashed") +
  ylab("predicted read density") +
  theme_pubclean()
fig_density_annotated
```



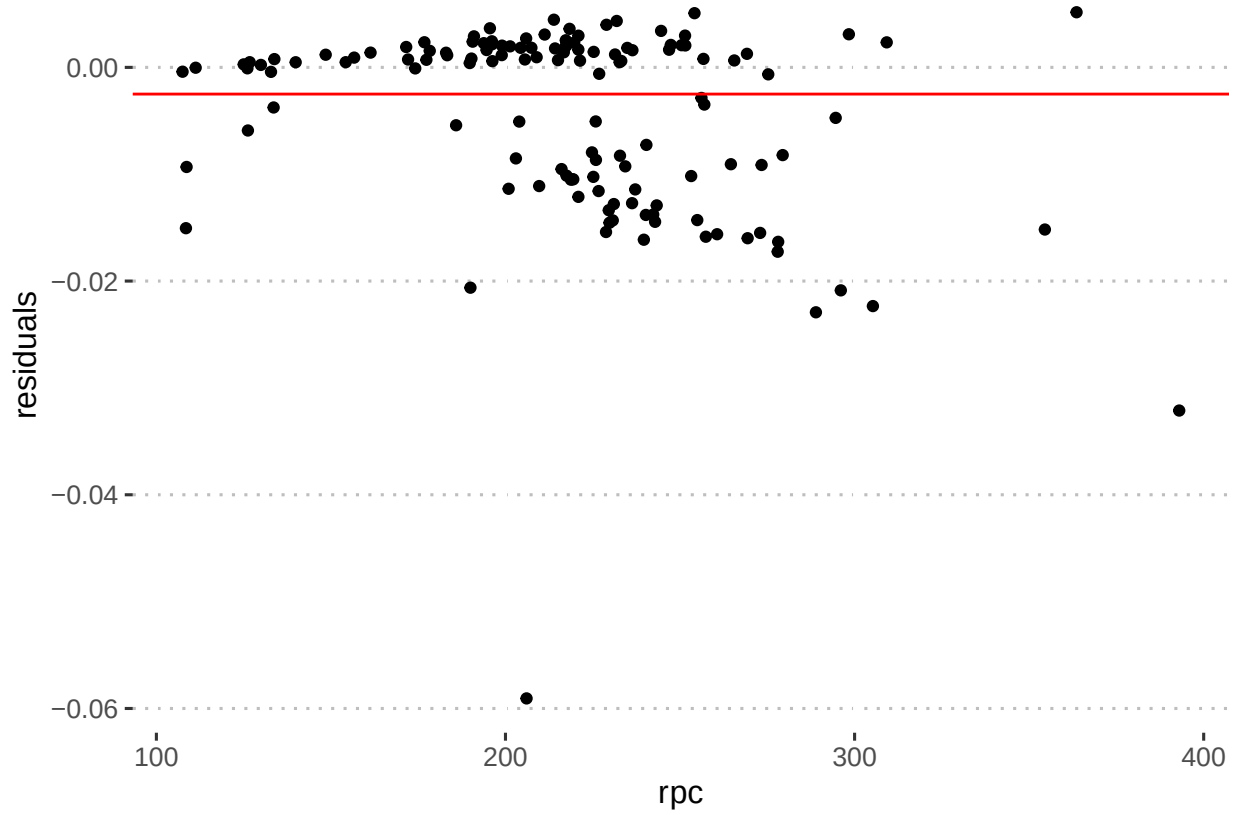
We can clearly see cells separating based on the measure of read density. However, in practive we do not have access to the empirical cellcycle measurement variable that tells us the state the cells are in, so let's try to predict G2 cells instead. Here, we use the robustbase regression as a simple way to separate the two populations, other approaches are obviously also possible. It's also possible to use a reference set of cells (ideally sequenced with similar sequencing parameters) and use a model of read density based on these cells to separate the populations. This is also a way to independently verify that the ploidy estimation procedure was correct, if cells are predicted at an unexpected ploidy level and experimental ploidy information is not available.

```
# create read density model
model <- robustbase::lmrob(prediction ~ I(rpc) + I(rpc^2), data = df %>% dplyr::filter(rpc > 100))
#summary(model)

# save predictions of the model in the new data frame
predicted_df <- data.frame(density_pred = predict(model, df),
                           rpc=df$rpc, density_measured=df$prediction,
                           groundtruth_cellcycle=df$cellcycle_measurement)

fig_density = ggplot(data = df) + geom_point(aes(x=rpc, y=prediction, color=prediction.quality)) +
  geom_vline(xintercept=100, color="red", linetype="dashed") +
  geom_line(color='red',data = predicted_df, aes(x=rpc, y=density_pred)) +
```

```
  theme_pubclean()
fig_density
```




```
predicted_df$residuals = predicted_df$density_pred - predicted_df$density_measured

fig_residual = ggplot(data = predicted_df %>% dplyr::filter(rpc > 100)) +
  geom_point(aes(x=rpc, y=residuals)) +
  geom_hline(yintercept=-0.0025, color="red") +
  theme_pubclean()
fig_residual
```




```
table(predicted_df$residuals < -0.0025, predicted_df$groundtruth_cellcycle,
      predicted_df$rpc > 100)
#> , ,  = FALSE
#>
#>
#>         G1 G2  S
#>   FALSE  4  1  0
#>   TRUE   4  6  0
#>
#> , ,  = TRUE
#>
#>
#>         G1 G2  S
#>   FALSE 75  1  0
#>   TRUE   6 48  2
```

Here, we simply eyeball a reasonable cutoff, and we find that we correctly identify 48/49 of the G2 cells, with a small number of false positive G1 cells, when we only consider cells with a suffciently high sequencing depth. Two things are worth mentioning here. First, notice that the G2 cells tend to have higher sequencing depth (a higher rho value), because of an experimental bias. Cells with more DNA tend to proportionally receive more reads in our experience. Second comparison requires cells to have similar values of rho, which in practice means you might have to downsample your reads to achieve approximately the same rho value distribution for all cells. In other words, if you have naively sequenced a population, the WGS/G2 cells will have half the rho value of the G1 population, as they will have twice the amount of DNA. Downsampling can address this. Alternatively, you can also use another set of cells as your reference set to build a model of read density.