Supplemental Protocol 4

iPS2-seq design and analysis with catcheR

This Supplemental Protocol describes the following procedures:

1. Overview
2. Installation
3. Oligonucleotides design
4. Pooled cloning step 1 plasmid QC
5. Pooled cloning step 2 plasmid QC
6. iPS2-10X-seq perturbation deconvolution
7. iPS2-sci-seq perturbation deconvolution
8. Barcode reassignment
9. Perturbation effect analysis

Overview

catcheR is a comprehensive bioinformatic package for designing and analyzing iPS2-seq experiments. It comprises the following functions (Figure SP4.1):

1. *catcheR_design*, which designs oligonucleotides for Supplemental Protocol 1 - Design shRNA oligonucleotides, facilitating shRNA library cloning
2. *catcheR_step1QC*, which analyzes the results of Supplemental Protocol 1 - Intermediate plasmid pool QC, assessing pooled cloning step 1 plasmids for barcode swaps
3. *catcheR_step2QC*, which analyzes the results of Supplemental Protocol 1 - Final plasmid pool QC, assessing pooled cloning step 2 for shRNA representation
4. *catcheR_scicount*, which analyzes 2-level indexing sci-RNA-seq data, facilitating the generation of gene expression matrix for iPS2-sci-seq experiments
5. *catcheR_scicatch*, which assigns shRNA perturbations to single nuclei transcriptomes obtained by Supplemental Protocol 2, enabling the primary analysis of iPS2-sci-seq
6. *catcheR_10Xcatch*, which assigns shRNA perturbations to single cell transcriptomes obtained by Supplemental Protocol 3, enabling the primary analysis of iPS2-10X-seq
7. *catcheR_scicatchQC* and *catcheR_10XcatchQC*, which use the outputs of *catcheR_ scicatch* and *catcheR_10Xcatch*, respectively, to fine-tune shRNA assignment thresholds
8. *catcheR_filtercatch*, which leverages on the output of *catcheR_scicatchQC* and *catcheR_10XcatchQC* to filter single nuclei/cell transcriptomes expressing a single shRNA
9. *catcheR_sortcatch*, which quality controls the cell-by-gene matrix based on the results of *catcheR_step1QC*, reassigning hPSC clones with barcode swaps to the correct shRNA
10. *catcheR_scinocatch* and *catcheR_10Xnocatch*, which identify cells expressing no shRNA in iPS2-sci-seq and iPS2-10X-seq experiments, respectively, adding them to the cell-by-gene matrix to be used as additional controls

11. *catcheR_load* to load the gene expression matrices generated after iPS2-10X-seq perturbation deconvolution and iPS2-sci-seq perturbation deconvolution and pack the data in a monocle object, ready to be analyzed

12. *catcheR_pseudotime*,*catcheR_modules* and *catcheR_enrichment* allow follow up analyses on the annotated perturbed cells

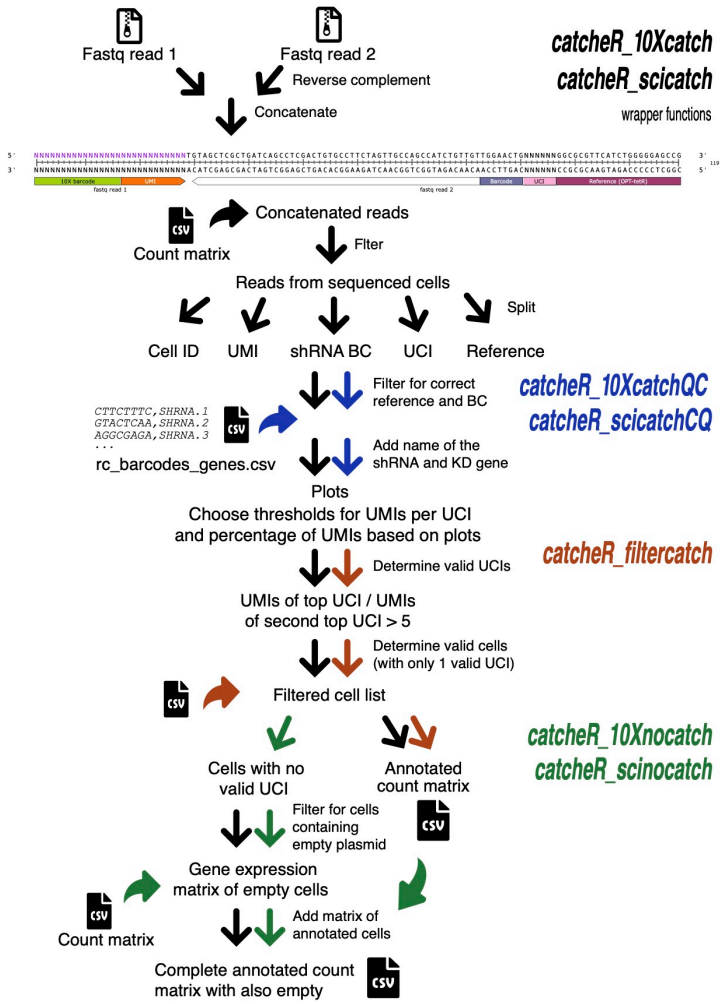


Figure SP4.1. Overview of catcheR
Logical relationship and main inputs/outputs for the main functions used for the catcheR analytical pipeline of iPS2-seq experiments. Functions used for oligo design and plasmid QC are not illustrated.

Installation

catcheR is available at https://github.com/alessandro-bertero/catcheR. The GitHub repository folder "scripts" contains all the bash and R scripts that can be run independently. However, in order to ensure reproducible analyses, we strongly recommend to install catcheR package from GitHub, since its functions run all the analysis inside Docker containers.

1. Install Docker engine, following the instructions at https://docs.docker.com/engine/install/

2. Install catcheR

   (a) In R (≥ 3.0.2) install *devtools*, if not already present

```
install.packages("devtools")
```

(b) Install catcheR from GitHub

```
install_github("alessandro-bertero/catcheR")
```

(c) Install rrundocker from GitHub

```
install_github("Reproducible-Bioinformatics/rrundocker")
```

(d) Load catcheR and rrundocker in your R environment

```
library(catcheR)
library(rrundocker)
```

Oligonucleotides design

This step complements Supplemental Protocol 1 - Design shRNA oligonucleotides

1. In a new working folder, prepare the following files:

   (a) A comma-separated values (CSV) file with three columns listing: (1) the forward oligos from the TRC shRNA library; (2) the corresponding barcodes (BC); (3) the shRNA names. Below is an example for the shRNA described in Figure SP1.2:

   ```
   CCGGCAAGTACTCCTTGCTGGATTGCTCGAGCAATCCAGCAAGGAGTACTTGTTTTTG,CAGTTCCA,SMAD2.1
   ...
   ```

   (b) (Optional) - A txt file with a newline-separated list of 5'-3' restriction sites, or other sequences, to be avoided in the shRNAs. By default these are SalI, SwaI and AscI:

   ```
   GTCGAC
   ATTTAAAT
   GGCGCGCC
   ```

2. Run *catcheR_design*:

```
catcheR_design(
    group=c("docker","sudo"),
    folder,
    sequences,
    gibson.five = "AGTTCCCTATCAGTGATAGAGATCCC",
    gibson.three = "GTAGCTCGCTGATCAGC",
    fixed = "GTCGACATTTAAATGGCGCGCC",
    restriction.sites = NULL)
```

catcheR_design arguments:

(a) group: string with two options: sudo or docker, depending on the user group. For a detailed explanation of Docker user groups, see this page

(b) folder: string with the working folder path

(c) sequences: string with the CSV file name from step 1a

(d) gibson.five: (optional) string with the 5' Gibson homology

(e) gibson.three: (optional) string with the 3' Gibson homology

(f) fixed: (optional) character string with the multicloning site

(g) restriction.sites: (optional) string with the txt file name from 1b

Example usage:

```
catcheR_design(
    group = "docker",
    folder = "path/to/folder",
    sequences = "filename.csv",
    restriction.sites = "filename.txt")
```

catcheR_design outputs:

(a) "output.txt", with the oligo sequences to be used for synthesis (Figure SP1.2)

(b) "bad_oligos.txt", with the shRNAs with forbidden restriction enzyme sites (highlighted in lowercase characters); this information can be used to refine the shRNA list
Example output for "bad_oligos.txt":

```
AGTTCCCTATCAGTGATAGAGATCCCGGACATAATCACTGCGTAATCCTCagatctTACGCAGTGATTATGTCCTTTTTTTGT-
CGACATTTAAATGGCGCGCCNNNNNNGCTGAAGAGTAGCTCGCTGATCAGC,GATA4
...
```

Pooled cloning step 1 plasmid QC

This step complements Supplemental Protocol 1 - Intermediate plasmid pool QC

1. In a new working folder, prepare the following files:

   (a) Fastq/fq or fastq.gz files with demultiplexed read 1 from the NGS run

   (b) A CSV file with the shRNA names and their full sequences

   ```
   SMAD2.1,GCAAGTACTCCTTGCTGGATTGCTCGAGCAATCCAGCAAGGAGTACTTG
   ...
   ```

   (c) (Optional) - A txt file with a newline-separated list of clones of interest as "BC_UCI"

   ```
   CAAGAGCC_CATCGT
   ...
   ```

2. Run *catcheR_step1QC*:

```
catcheR_step1QC(
    group=c("docker","sudo"),
    folder,
    fastq.read1,
    DIs = 100,
    ratio = 10,
    plot.threshold = 2000,
    clones = NULL)
```

catcheR_step1QC arguments:

(a) group: string with two options: sudo or docker, depending on the user group (info)

(b) folder: string with the working folder path

(c) fastq.read1: string with the read 1 filename from step 1a

(d) DIs: integer of the minimum number of diversity indexes (DIs, pseudo-unique reads) of the most represented shRNA matching to a given UCI-BC; in combination with "ratio", it selects UCI-BCs for which it is possible to reliably assign an shRNA

(e) ratio: integer of the minimum ratio between the number of DIs of the most represented and second most represented shRNAs matching to a given UCI-BC

(f) plot.threshold: integer of the minimum number of DIs per UCI-BC for output 2b

(g) clones: (optional) a string with the txt file from step 1b

Example usage:

```
catcheR_step1QC(
    group = "docker",
    folder = "path/to/folder",
    fastq.read1 = "filename.fq",
    clones = "filename.txt")
```

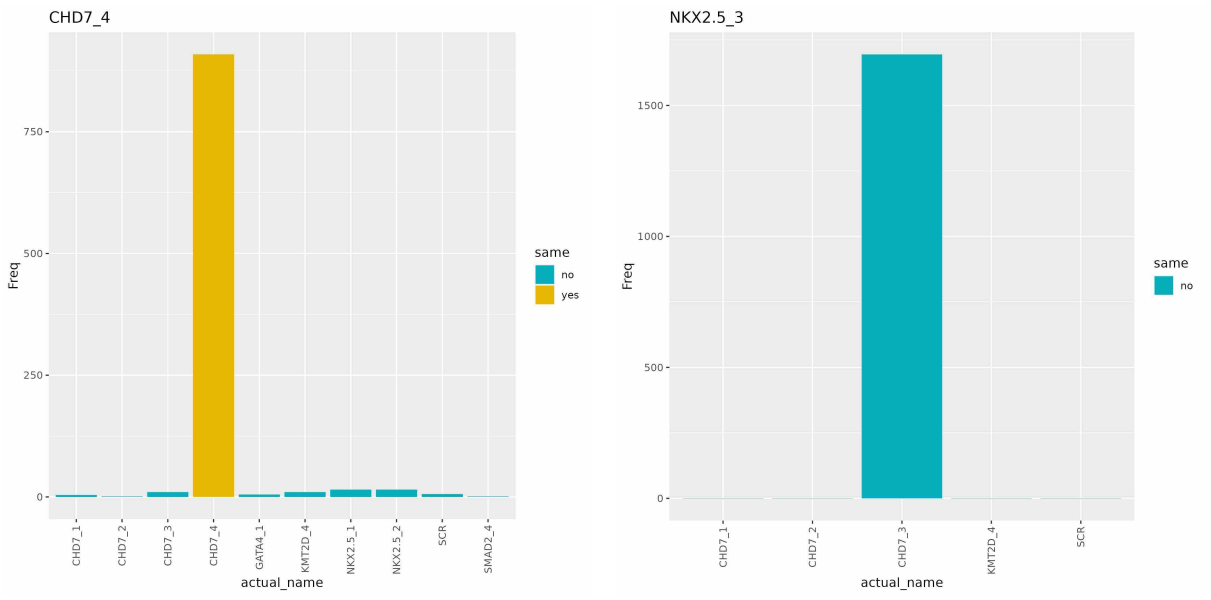
catcheR_step1QC key outputs:

(a) ”reliable_clones_swaps.csv”, which lists UCI-BCs with reliable evidence of shRNA-barcode swap, and is used as input for Barcode reassignment

(b) Bar chart of the number of DIs for each clone above ”plot.threshold”

(c) Bar chart of the number of DIs associated to each BCs, shRNAs, and reliable swaps

(d) (If ”clones” argument is provided) - csv files and bar charts with the number of DIs for each shRNA matching to each clone of interest (Figure SP4.2)



(A) (B)

Figure SP4.2. Example of quantification of DIs for bacterial clones carrying the expected shRNA (A) or with robust evidence of a barcode swap that can be reassigned (B; Figure 1B). These graphs allow visual evaluation of appropriate thresholds for ”DI” and ”ratio”.

Pooled cloning step 2 plasmid QC

This step complements Supplemental Protocol 1 - Final plasmid pool QC

1. In a new working folder, prepare the following files:

(a) Fastq/fq or fastq.gz files with demultiplexed read 1 from the NGS run

(b) ”rc_barcodes_genes.csv”, a CSV file with two columns: (1) the shRNA BCs; (2) the matching shRNA names in the format ”GENE.shRNAID”

```
CAAGAGCC,SMAD2.1
...
```

(c) (Optional) - A txt file with clones of interest (step 1a of the previous section)

2. Run *catcheR_step2QC*:

```
catcheR_step2QC(
    group=c("docker","sudo"),
    folder,
    fastq.read1,
    DIs = 1000,
    clones = NULL)
```

catcheR_step2QC arguments:

(a) group: string with two options: sudo or docker, depending on the user group (info)
(b) folder: string with the working folder path
(c) fastq.read1: string with the read 1 file from step 1a
(d) DIs: integer of the minimum number of DIs for a given UCI-BC; it selects reliably-measured UCI-BCs
(e) clones: (optional) a string with the txt file from step 1c

Example usage:

```
catcheR_step2QC(
    group= "docker",
    folder = "path/to/working/folder",
    fastq.read1 = "filename.fq",
    clones = "filename.txt")
```

catcheR_step2QC key outputs:

(a) Pie charts of DIs per shRNA and per gene targets associated to each clone
(b) Text file listing all clones above the DI threshold
(c) Bar chart of the number of DIs for each clone above the DI threshold (Figure SP4.3A)
(d) Frequency histogram of the percentage of clones above the DI threshold for each shRNA (Figure SP4.3B)

Obtain iPS2-10X-seq count matrices

1. Download and install *cellranger* from the 10X Genomics download resource page; alternatively, a Docker container with cellranger 7 or cellranger 9 can be pulled from our Docker repository. Preprocess iPS2-multi-seq with cellranger ARC, which can be either downloaded from the 10X Genomics download resource page or pulled as a Docker container from our repository

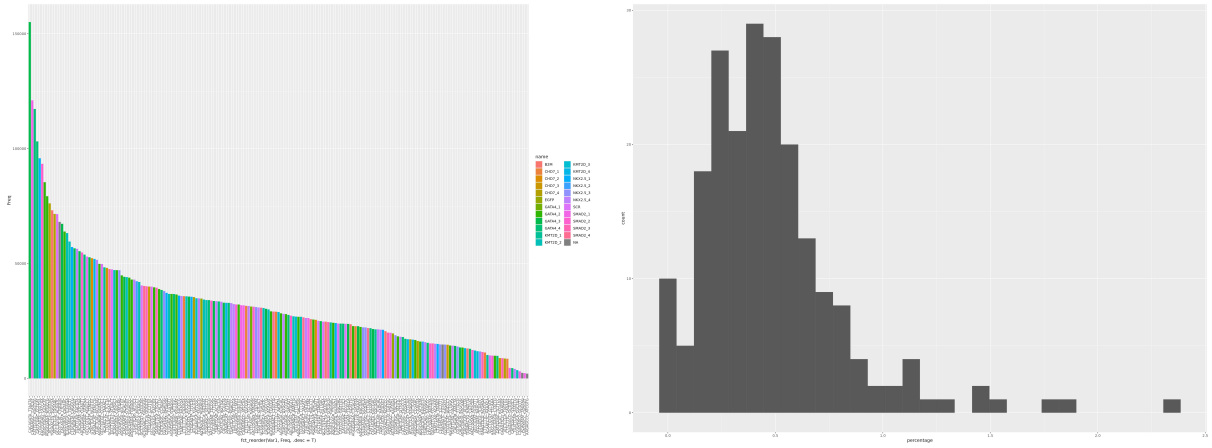
2. Demultiplex Illumina base calls with *cellranger mkfastq*, following 10X Genomics' manual. In the CSV sample sheet, include the 10X Genomics indexes used to generate the GEX, CMO, and UCI-BC libraries in Supplemental Protocol 3

3. Obtain cell-by-gene count matrices using *cellranger count* (single sample) or *cellranger multi* (multiplexed experiment).

   Note: count matrices from multiple samples from the same experiment can be aggregated with *cellranger aggr*, then can be analyzed in a single run of *catcheR_10Xcatch* by providing the number of samples to the "samples" argument.

4. Run *cell ranger matrix2csv* to transform the sparse count matrix in a dense CSV matrix

After obtaining the count matrix, proceed to iPS2-10X-seq perturbation deconvolution.



(A) (B)

Figure SP4.3. Example of quantification of DIs for bacterial clones represented by more than 1,000 DIs (A), and of the frequency of bacterial clones for each shRNA. These graphs confirm whether shRNAs are normally distributed in the final plasmid pool.

Obtain iPS2-sci-seq count matrices

1. *catcheR_scicount* is a wrapper of the ”bbi-sci” pipeline developed by the Brotman Baty Institute for Precision Medicine, which was implemented and dockerized in catcheR to be used on any operating system.

2. Demultiplex Illumina base calls to fastq files

   (a) Create a ”SampleSheet.csv” file with as many sample rows as the PCR wells from Indexing PCR and NGS of Supplemental Protocol 2, where ”Sample_ID” is the well identified with format [A-H][01-12], and index and index2 are the i7 and i5 indexes used in the corresponding row and column (refer to Table SP2.1)

   (b) Run Illumina *bcl2fastq* following Illumina’s manual

   (c) Run fastQC to confirm the quality of the fastq files

3. In a new working folder:

   (a) Create the subfolder ”fastq”, and copy all the demultiplexed fastq.gz files. Ensure that all file names begin with the well coordinate (e.g. A01) followed by an underscore.

   (b) Create a tab separated txt file called ”sci-RNA-seq-8.RT.oligos” with the association between RT wells and RT barcode sequences (refer to Table SP2.1)

   ```
   A01 TTCTCGCATG
   ...
   ```

   (c) Create the subfolder ”GENOMES”, and copy the annotated genome (i.e., GRCh38)

4. Run *catcheR_scicount*:

   ```
   catcheR_scicount(
       group=c("docker","sudo"),
       folder,
       sample.name,
       UMI.cutoff)
   ```

   catcheR_scicount arguments:

   (a) group: string with two options: sudo or docker, depending on the user group (info)

(b) folder: string with the working folder path

(c) sample.name: string with the name of the experiment

(d) UMI.cutoff: integer of the minimum number of UMI per nucleus needed to consider the single cell transcriptome valid

Example usage:

```
catcheR_scicount(
    group = "docker",
    folder = "path/to/file",
    sample.name = "experiment",
    UMI.cutoff = 500)
```

catcheR_scicount outputs, found in the final-output folder: knee-plot of UMI per cell (Figure S2F), statistics files, sparse cell-by-gene count matrix, dense gene matrix "exp_mat.csv" and "exp_mat_no0.csv" (filtered to exclude genes with 0 counts in all cells), also in Rdata format.

After obtaining the count matrix, proceed to iPS2-sci-seq perturbation deconvolution.

iPS2-10X-seq perturbation deconvolution

shRNA perturbations can be assigned to single cells using *catcheR_10Xcatch*, which identifies NGS reads containing UCI-BCs, matches them to the corresponding transcriptome based on their shared cellular barcodes, applies filters for background noise, and selects cells with robust evidence of a single integration. UCI-BC filtering involves: (1) filtering out UCI-BCs supported by less than a certain number of UMIs in a given transcriptome, to account for PCR and sequencing artifacts; (2) filtering out UCI-BCs that represent less than a certain fraction of all UMIs for UCI-BCs in a given transcriptome, to reduce noise arising from free mRNA; and (3) filtering out UCI-BCs whose UMI fraction in a given transcriptome is not several fold greater than the second most common UCI-BC, to eliminate cells that contain multiple UCI-BCs when the second most common one falls just below one of the first two thresholds. The resulting list of *bona fide* shRNA integrations per cell is then used to filter those expressing a single shRNA.

This section describes a typical analysis pipeline starting with *catcheR_ 10Xcatch* to perform a full analysis with automatic thresholds, after which *catcheR_10XcatchQC* can fine-tune the thresholds to re-filter cells with *catcheR_filtercatch*. Lastly, *catcheR_10Xnocatch* can add cells expressing no shRNA as additional controls to the final annotated cell-by-gene count matrix.

1. In a new working folder:

   (a) Copy the fastq/fq or fastq.gz files with demultiplexed read 1 and read 2 of the UCI-BC library (step 2 of Obtain iPS2-10X-seq count matrices)

   (b) Copy the cell-by-gene count matrix in CSV format (step 4 of Obtain iPS2-10X-seq count matrices)

   (c) Create a CSV file named "rc_barcodes_genes.csv" with two columns: (1) the shRNA BCs; (2) the matching shRNA names in the format "GENE.shRNAID"

   ```
   CAAGAGCC,SMAD2.1
   ...
   ```

2. Run *catcheR_10Xcatch* to execute the complete analysis with automatic thresholding:

   ```
   catcheR_10Xcatch(
       group=c("docker","sudo"),
       folder,
       fastq.read1,
   ```

```
    fastq.read2,
    expression.matrix,
    reference = "GGCGCGTTCATCTGGGGGAGCCG",
    UCI.length = 6,
    threads = 2,
    percentage = 15,
    mode = "bimodal",
    ratio = 5,
    samples = 1,
    x = 100,
    y = 400)
```

catcheR_10Xcatch arguments:

(a) group: string with two options: sudo or docker, depending on the user group (info)

(b) folder: string with the working folder path

(c) fastq.read1: string with the filename of read 1 fastq/fq or fastq.gz (step 1a)

(d) fastq.read2: string with the filename of read 2 fastq/fq or fastq.gz (step 1a)

(e) expression.matrix: string with the filename of the count matrix file CSV (step 1b)

(f) reference: string with the reverse complement of the sequence before the shRNA BC at the start of read 2 (default is the 3' end of the OPTtetR cDNA, optional argument; Figure S3F)

(g) UCI.length: integer of the UCI length (default is 6, optional argument)

(h) threads: integer of the threads to be used for parallelization (default is 2, optional argument)

(i) percentage: integer of the percentage of UMIs supporting a given UCI over the total UMIs supporting all UCIs in a given cell; it is used as threshold to consider the UCI valid (Figure Figure SP4.4D); the recommended default is 15 (optional argument)

(j) mode: string with two options: bimodal or noise, defining the mode for automatic thresholding the minimum number of UMIs per UCI to consider the UCI valid (optional argument)

   i. "bimodal" (default) sets the threshold at the valley of the bimodal UMIxUCI distribution (Figure SP4.4C)

   ii. "noise" sets the threshold at 1.35 * the number of UCIs supported by a single UMI; used if the UMIxUCI distribution is not bimodal

(k) ratio: the ratio between the number of UMIs supporting the most represented UCI and the number of UMIs supporting the second most represented UCI. The parameter is needed to identify the cell as a single integration cell. The default is 5 (optional argument)

(l) samples: the number of samples present in the same experiment (cells from different experiments will have the corresponding number after the cell ID in the gene expression matrix, so that multiple reactions of the same experiment can be unified). The default is 1 (optional argument)

(m) x: an integer indicating the upper limit on the x-axis of the cropped version of the plot "UMIxUCI". Default is 100 (optional argument)

(n) y: an integer indicating the upper limit on the y axis of the cropped version of the plot "UMIxUCI". Default is 400 (optional argument)

Example usage:

```
catcheR_10Xcatch(group = "docker",
    folder = "path/to/folder",
    fastq.read1 = "R1.fastq.gz",
    fastq.read2 = "R2.fastq.gz",
    expression.matrix = "filename.csv",
    threads = 12)
```

catcheR_10Xcatch key outputs (will be stored inside the ”Result” folder of each sample):

(a) ”log.txt” with the starting number of reads

(b) ”log2.txt” with the number of cells, UMIs, UCIs, and the calculated thresholds of UMI per UCI for both ”bimodal” and ”noise” (only one is chosen based on the ”mode” argument; the other one is provided for a potential reanalysis)

(c) Bar charts of UMI counts per shRNA and target gene (Figure SP4.4A and Figure SP4.4B)

(d) Frequency histograms of UCIs supported by a certain number of UMIs (UMIxUCI), to interpret the signal/noise of the experiment and possibly set a custom UMIxUCI threshold for subsequent reanalysis (Figure SP4.4C).

Note: here and below, copies of the same UCI expressed by different cells are plotted and analyzed separately

(e) Frequency histogram of UCIs supported by a certain fraction of UMIs over the total number of UMIs supporting all UCIs in a given cell (UMIpercentagexUCI), to further assess signal/noise and possibly adjust the default threshold (Figure SP4.4D)

(f) Dot plots that combine the UMIpercentagexUCI and UMIxUCI data on the x and y axes, respectively, with each dot representing one or more UCI (quantified by the dot size). Dot colors indicate either the number of valid UCIs (i.e., shRNAs) in the cell containing a given UCI (Figure SP4.4E), or whether said UCI is the only valid one in the relevant cell and is thus assigned to such cell (Figure SP4.4F)

(g) ”log_part3.txt” with how many cells were assigned to a single shRNA or were filtered out due to zero or multiple shRNAs

(h) ”silencing_matrix.csv” is the key output used for the secondary analyses: the cell-by-gene count matrix provided as input, filtered and annotated with the shRNA encoded in each cell. It is also provided in RDS format for easy loading into R. Cell names are modified as follows:

```
cellID_UMIxUCI_BC_GENE_UCI
```

Where:

i. cellID is the original cell name (i.e., the 16 bp of the 10X RT barcode)
ii. UMIsxUCI is the number of UMI associated with that perturbation
iii. BC is the shRNA barcode
iv. GENE is the shRNA target
v. UCI is the Unique Clonal Identifier (shared by all cells originating from the same hPSC clone)

Example of a cell name after the analysis:

```
TTCTAACCACAGTCGC_180_CGTGATGC_NKX2.5_ACAGTG
```

This annotation can be leveraged with the scripts available on our GitHub repository to enable a variety of secondary analyses described in the manuscript, or using custom scripts for other types of secondary analyses or with the functions implemented within catcheR and described in the Perturbation effect analysis.

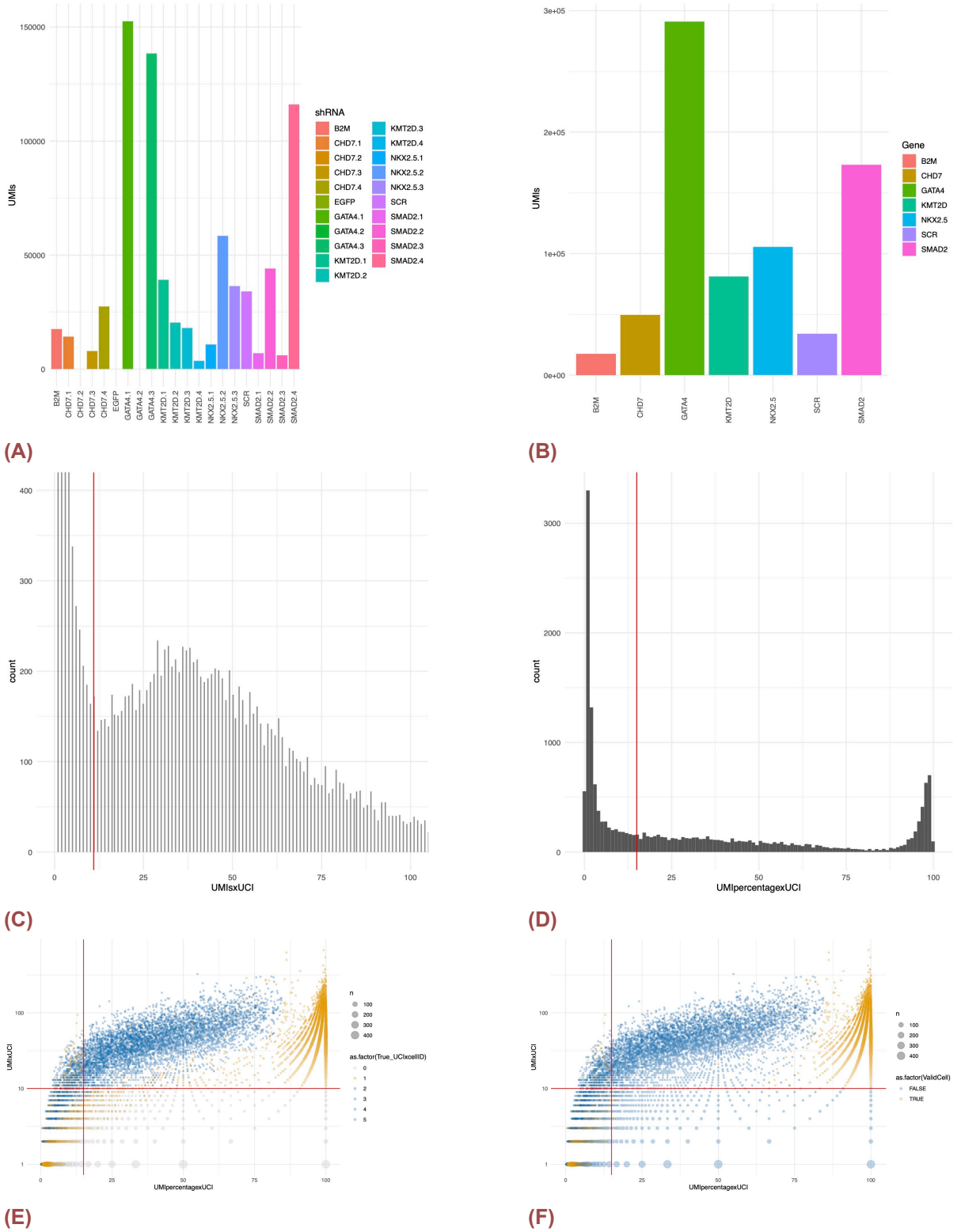


Figure SP4.4. Exemplary key outputs of a *catcheR_10Xcatch* analysis: ”barcode_distribution.pdf” (A); ”gene_distribution.pdf” (B); ”UMIxUCI_400_100.pdf” with the automatic UMIxUCI threshold based on ”bimodal” (C); ”percentage_of_UMIxUCI_dist.pdf” with the default 15% ”percentage” threshold (D); ”2D_percentage_of_UMIxUCI_UMI_count_trueorfalse.pdf” (E); and ”2D_percentage_of_UMIxUCI_UMI_ValidCells.pdf” (F)

OPTIONAL: fine tune iPS2-10X-seq perturbation assignment

1. In the same working folder used to run *catcheR_10Xcatch*, run *catcheR_10XcatchQC*, which uses the txt file outputs to regenerate the quality control plots described in Figure SP4.4 and suggest new thresholds to enable subsequent filtering

```
catcheR_10XcatchQC(
    group=c("docker","sudo"),
    folder,
    reference = "GGCGCGTTCATCTGGGGGAGCCG",
    mode = "bimodal",
    sample = 1
    x = 100,
    y = 400)
```

Example usage:

```
catcheR_10XcatchQC(
    group = "docker",
    folder = "path/to/working/folder",
    mode = "noise")
```

The function is to be run separately for each sample (specified by argument sample with default 1). *catcheR_10XcatchQC* arguments and outputs are the same as *catcheR_10Xcatch*

2. In the same working folder used to run *catcheR_10Xcatch*, run *catcheR_filtercatch*, which uses the output of *catcheR_10Xcatch* to filter cells based on custom thresholds

```
catcheR_filtercatch(
    group=c("docker","sudo"),
    folder,
    expression.matrix,
    UMI.count,
    percentage = 15,
    ratio = 5,
    sample = 1)
```

Example usage:

```
catcheR_filtercatch(group = "docker",
    folder = "path/to/working/folder",
    expression.matrix = "filename.csv",
    UMI.count = 5,
    sample = 1)
```

The function is to be run separately for each sample (specified by argument sample with default 1). *catcheR_filtercatch* argument *percentage* is the minimum percentage of UMI for a given UCI over the total UMIs of UCIs in that cell, to consider the UCI valid. *UMI.count* is an integer of the custom UMI threshold. *ratio* is the minimum ration between UMIs of top UCI and UMIs of 2nd top UCI. The other arguments and the outputs are the same as *catcheR_10Xcatch*.

OPTIONAL: identify cells expressing no shRNA in iPS2-10X-seq

1. In the same working folder used to run *catcheR_10Xcatch*, run *catcheR_10Xnocatch*:

```
catcheR_10Xnocatch(
    group=c("docker","sudo"),
    folder,
    expression.matrix,
```

```
    threshold,
    sample = 1,
    reference = "TACGCGTTCATCTGGGGGAGCCG")
```

Example usage:

```
    catcheR_10Xnocatch(group = "docker",
    folder = "path/to/folder",
    expression.matrix = "filename.csv",
    threshold = 5)
```

catcheR_10Xnocatch argument *threshold* is an integer of the minimal UMI count for the "empty" reference Figure SP4.3B needed to confidently identify it as having integrated said plasmid; all other arguments are the same as *catcheR_10Xcatch*

catcheR_10Xnocatch output is an updated "silencing_matrix.csv", also available in RDS format, in which names of empty cells are modified as "cellID_?_empty_NA_empty".

OPTIONAL: merge multiple samples

1. In case fine-tuning is applied after *catcheR_10Xcatch* and multiple samples are present, run *catcherR_merge* to aggregate the results in a single matrix. This step is done automatically when running *catcheR_10Xcatch*.

```
catcheR_merge(
    group=c("docker","sudo"),
    folder,
    samples = 2,
    empty = T
    )
```

The argument "empty" determines whether the cells identified by *catcheR_nocatch* should be added to the matrix.

Example usage:

```
    catcheR_merge(group = "docker",
    folder = "/20tb/ratto/catcheR/test_CM5/",
    samples = 4)
```

iPS2-sci-seq perturbation deconvolution

This section describes a typical analysis pipeline similar to the one described in the previous section, but leveraging on the *catcheR_scicatch*, *catcheR_scicatchQC*, *catcheR_filtercatch*, and *catcheR_scinocatch* functions.

1. In a new working folder:

   (a) Create a subfolder called "fastq", and copy all the demultiplexed fastq.gz files (step 2 of Obtain iPS2-sci-seq count matrices). Ensure that all file names begin with the well coordinate (e.g. A01)

   (b) Copy the cell-by-gene expression matrix CSV file obtained with *catcheR_scicount* function (step 4 of Obtain iPS2-sci-seq count matrices).

   (c) Create a CSV file called "rc_barcodes_genes.csv" with two columns: (1) the shRNA BCs; (2) the matching shRNA names in the format "GENE.shRNAID"

```
CAAGAGCC,SMAD2.1
...
```

(d) Copy the text file ”sci-RNA-seq-8.RT.oligos” used by *catcheR_scicount* (step 3b of Obtain iPS2-sci-seq count matrices).

2. Run *catcheR_scicatch* to execute the complete analysis with automatic thresholding; the arguments are the same as for *catcheR_10Xcatch* (step 2 of iPS2-10X-seq perturbation deconvolution), except that no filenames are provided since fastq files must be in the ”fastq” subfolder in the working directory:

```
catcheR_scicatch(
    group=c("docker","sudo"),
    folder,
    expression.matrix,
    reference = "GGCGCGTTCATCTGGGGGAGCCG",
    UCI.length = 6,
    threads = 2,
    percentage = 15,
    ratio = 5,
    mode = "bimodal",
    x = 100,
    y = 400)
```

Example usage:

```
catcheR_scicatch(group = "docker",
    folder = "path/to/working/folder",
    expression.matrix = "filename.csv",
    threads = 12)
```

catcheR_scicatch key outputs are the same as for *catcheR_10Xcatch* (step 2 of **??** and Figure SP4.3), except that in ”silencing_matrix.csv” ”cell ID” indicates the PCR well and RT barcode ID (which can be leveraged to identify nuclei from different samples pooled on the same RT plate)

Example of cell name after the analysis:

```
P24__RT_27_7_GCCTGTGT_SCR_ACGGTC
```

In addition, *catcheR_scicatch* provides two quality control outputs to evaluate experimental biases during sci-RNA-seq library preparation: *demux* and *RT* detail about how many cells were identified from each row and column of the PCR and RT plates, respectively.

OPTIONAL: fine tune iPS2-sci-seq perturbation assignment

1. Run *catcheR_scicatchQC* following the steps described for *catcheR_10XcatchQC* in OPTIONAL: fine tune iPS2-10X-seq perturbation assignment: the functions share the same arguments and outputs, but *catcheR_scicatchQC* used the output of *catcheR_scicatch*

Example usage:

```
catcheR_scicatchQC(
    group = "docker",
    folder = "path/to/working/folder",
    mode = "noise")
```

2. Run catcheR_filtercatch function as described in in OPTIONAL: fine tune iPS2-10X-seq perturbation assignment: this function also works on the output of *catcheR_scicatch* or *catcheR_scicatchQC*

OPTIONAL: identify cells expressing no shRNA in iPS2-sci-seq

1. Run *catcheR_scinocatch* following the steps described for *catcheR_10Xnocatch* in OPTIONAL: identify cells expressing no shRNA in iPS2-10X-seq: the functions share the same arguments and outputs, but *catcheR_scinocatch* uses the output of *catcheR_scicatch*

   Example usage:

```
catcheR_scinocatch(group = "docker",
    folder = "path/to/working/folder",
    expression.matrix = "filename.csv",
    threshold = 5,
    reference = "TACGCGTTCATCTGGGGGAG")
```

Barcode reassignment

catcheR_sortcatch is an optional function that corrects the annotated cell-by-gene count matrix obtained with *catcheR_10Xcatch* or *catcheR_scicatch* reassigning the perturbation of any cell belonging to a hPSC clone with reliable evidence of a shRNA-barcode swap, based on the results of *catcheR_step1QC*.

1. In a new working folder:

   (a) Copy the annotated count matrix (i.e., ”silencing_matrix.csv”)

   (b) Copy the CSV file with the list of UCI-BCs with reliable evidence of a shRNA-barcode swap (i.e., ”reliable_clones_swaps.csv”; step 2a of Pooled cloning step 1 plasmid QC)

2. Run *catcheR_sortcatch*:

```
catcheR_sortcatch(
    group=c("sudo","docker"),
    folder,
    expression.matrix,
    swaps)
```

   Example usage:

```
catcheR_sortcatch(
    group = "docker",
    folder = "path/to/working/folder",
    expression.matrix = "silencing_matrix.csv",
    swaps = "reliable_clones_swaps.csv")
```

   catcheR_sortcatch arguments:

   (a) group: string with two options: sudo or docker, depending on the user group (info)

   (b) folder: string with the working folder path

   (c) expression.matrix: string with the filename of the annotated count matrix CSV

   (d) swaps: a character string with the filename of the txt file (step 1b)

   catcheR_sortcatch output is an updated annotated gene expression matrix CSV file called ”silencing_matrix_updated.csv”, in which ”BC” and ”GENE” have been modified to reflect the actual shRNA encoded in each cell

Perturbation effect analysis

Once each cell is assigned to its corresponding perturbation, the impact of each perturbation on gene expression can be assessed by comparing it to control conditions.

The second part of catcheR offers a comprehensive exploratory analysis, featuring visualizations and statistical summaries that highlight the biological effects of the perturbations of interest.

Annotation

Before proceeding, the gene expression matrix should be annotated with gene symbols with *scannobyGtf* function from the R package rCASC. Optionally, before that, ribosomal and mitochondrial genes can be assessed (e.g. with the function *mitoRiboUmi* from the same package).

Note: After this step, row names of the matrix (the genes) will have the following format:

```
GeneSymbol:EnsemblID
```

E.g.:

```
ENSG00000000003:TSPAN6
```

Data loading

This step enables the loading of single-cell data generated by either *catcheR_10Xcatch* or *catcheR_scicatch*, following GTF-based annotation. The data is imported into a Monocle object, where experimental design information is added, followed by normalization and clustering.

1. In a new working folder:
   (a) Copy the annotated count matrix (filtered_annotated_silencing_matrix_complete_all_samples.csv).
   (b) Copy the file rc_barcodes_genes.csv described above.
   (c) Create a new-line separated file listing the control genes (e.g. SCR, B2M).
   (d) Create a new-line separated file listing the control samples, if any (e.g. 1,3). These are the sample names also used by *aggr* (see CSV file used as input for *aggr*).
   (e) Create a newline-separated file listing the sample replicates (e.g., batch1, batch1, batch2, batch2). When multiple samples from different experiments are included, batch correction is recommended. The replicate names defined in this file must correspond to the samples in the same order.
   (f) Create a CSV file listing each sample along with its corresponding annotation name, which will be used for display instead of the sample number (this file is mandatory). Example of CSV file to download on GitHub
   (g) Create a newline-separated file listing the genes of interest whose expression you wish to visualize on the UMAP (optional).
2. Run *catcheR_load*:

```
catcheR_load(
  group="docker",
  folder,
  expression.matrix,
  control_genes,
  control_samples = NULL,
  replicates = NULL,
  sample_names,
  resolution = 8e-4,
  genes = NULL)
```

Example usage:

```
catcheR_load(
  group="docker",
  folder="/path/to/working/folder/",
  expression.matrix =
   "annotated_silencing_matrix_complete_all_samples.csv",
  control_genes =  "controls.txt",
  control_samples = "noTET.txt",
  replicates = "replicates.txt",
  sample_names = "samples.csv",
  resolution = 8e-4,
  genes = "genelist.txt")
```

The argument *resolution* refers to the resolution parameter used by Monocle's *cluster_cells* function for clustering.

catcheR_load outputs:

1. expression_data.csv and cell_metadata.csv : These files can be used to create a Monocle Cell Data Set (CDS) and are also included in the ready-to-load R object starting_cds.Rdata.

2. UMAP.pdf plots the dimensionality reduction and UMAP_gene_expression.pdf shows the gene expression on the UMAP of the genes provided by the argument "genes".

3. UMAP_clustering.pdf show the clustering obtained on the UMAP with the provided resolution.

4. processed_cds.RData is the Cell Data Set after normalization, dimensionality reduction, clustering and calculation of trajectories.

At the end of this step, the data are structured in a format compatible with the Monocle3 package. However, switching to other platforms such as Seurat or Scanpy is possible at any point in the analysis. **Note**: When performing the standard iPS2-seq perturbation analysis, always use the CDS object generated by catcheR after each step.

```
library(SeuratWrappers)
library(Seurat)
seurat <- as.Seurat(cds, assay = NULL)
scanpy_sce <- as.SingleCellExperiment(seurat)
```

The follow up analysis are: *catcheR_pseudotime*, *catcheR_modules* and *catcheR_enrichment*.

1. With the Pseudotime function, catcheR calculates the cumulative frequency of cells sharing the same gene, shRNA, or clone, compared to each control, along the pseudotime trajectory. CatcheR can compare the cumulative frequency curves by computing directional Kolmogorov-Smirnov statistics between the target and control.

2. With the Genes modules function, catcheR identifies gene modules within the dataset and computes the cumulative frequency of cells with the same gene, shRNA, or clone—relative to each control—based on the expression of each gene module. Also in this case, cumulative frequency curves are compared using a directional Kolmogorov-Smirnov test.

3. With the Enrichment / depletion analysis function, catcheR calculates the enrichment of cells with the same gene, shRNA, or clone within clusters, in comparison to each control. CatcheR can compare each target and control quantity and obtain statistics using the Fisher exact test.

Pseudotime

Since monocle pseudotime calculation requires the user to select the starting point interactively, the first step of the pseudotime analysis needs to be done separately within monocle3.

1. After loading the processed_cds.RData in R, calculate the pseudotime trajectory and plot it with the R script below:

```
library(monocle3)
cds <- order_cells(cds)
pt = as.data.frame(pseudotime(cds))
names(pt) = c("pseudotime")
write.csv(pt, paste0(dir, "/pseudotime.csv"))

plot_cells(cds,
           color_cells_by = "pseudotime",
           label_cell_groups=FALSE,
           label_leaves=FALSE,
           label_branch_points=FALSE,
           graph_label_size=1.5)
```

2. The working folder will contain the output of *catcheR_load* and the CSV file obtained after running the monocle pseudotime function (e.g. ”pseudotime.csv”).

3. Run *catcheR_pseudotime*:

```
catcheR_pseudotime(
    group=c("docker","sudo"),
    folder,
    cds,
    pseudotime,
    all = FALSE)
```

   Example usage:

```
catcheR_pseudotime(
    group="docker",
    folder="/path/to/working/folder/",
    cds = "processed_cds.RData",
    pseudotime = "pseudotime.csv")
```

   Argument ”all” is a logical operator indicating whether to perform the Kolmogorov-Smirnov test against the controls together (all = F) or against each control separately (all = T). The default argument is false.

   Below is a list of the *catcheR_pseudotime* outputs:

   (a) The ”cumulative_frequency_pseudotime” plots show the number of cells at each given point of the pseudotime for different groups of cells at different comparison levels - gene, shRNA, and clone.

   (b) The ”ks_statistics” CSV files that include the Kolmogorov-Smirnov test results comparing cumulative the frequency based on the pseudotime between KD and different controls

   (c) The Volcano plots of the results of the Kolmogorov-Smirnov test show the significance and fold change between the pseudotime cumulative curves. Different controls and levels are used.

   (d) The ”correlated_pseudotime_gene_exp.pdf” showing the expression on the UMAP of the genes most correlated with the pseudotime.

Genes modules

This function uses Monocle to find gene modules which expression varies in different perturbation groups, i.e., perturbed gene or shRNA or clones.

1. Run *catcheR_modules*:

```
catcheR_modules(group=c("docker", "sudo"),
    folder,
    cds,
    resolution=1e-2)
```

   Example usage:

```
catcheR_modules(group="docker",
    folder="/30tb/3tb/data/ratto/testing/",
    cds = "processed_cds.RData")
```

   The resolution parameter refers to the value used in Monocle's *find_gene_modules* function, which influences the number of gene modules identified and the number of genes included in each module—higher resolution typically results in more, smaller modules.

   Below are listed the outputs of *catcheR_modules*:

   (a) gene_modules.csv, listing the genes present in each module.

   (b) Heatmap plots display the Z-scores of each module—either all modules or the top 10 most variable ones—across perturbation groups.

   (c) The "modules_cells" folder contains tables with aggregated module expression values for each cell. These can be used as input for *catcheR_pseudotime* in place of the pseudotime CSV file, allowing the analysis of cumulative frequency based on module expression.

2. Run *catcheR_pseudotime* on the CSV files stored in the "modules_cells" folder to use the data generated by *catcheR_modules* as described in the previous example in the Pseudotime section

Enrichment / depletion analysis

This function evaluates whether perturbation groups are enriched or depleted in cells (number) or in specific cell subpopulations (clusters.

1. Run *catcheR_enrichment*:

```
catcheR_enrichment(
    group=c("docker","sudo"),
    folder,
    file,
    meta,
    timepoint = "PSC",
    control_gene = "SCR",
    min_cells_cluster = 70,
    min_cells_shRNA = 40)
```

   Example usage:

```
catcheR_enrichment(group = "docker",
folder ="/3tb/data/ratto/aggr/test /"
file = "processed_cds.RData",
meta = "cell_metadata.csv",
timepoint = "PSC",
control_gene = "SCR")
```

The required input is the cds file generated from *catcheR_load*.

Timepoint refers to the baseline time point used as a control for statistical analysis. It is required when experiments include multiple time points and aim to assess enrichment or depletion across these time points. *control_gene* specifies the control gene used as a reference for statistical comparisons.

Below are the outputs of *catcheR_enrichment*:

(a) Plots of cells in each perturbation group.

(b) Volcano plot showing enrichment or depletion of cell numbers in perturbation groups compared to the control,based on fold change (log2ratio and FC)and statistical significance. A corresponding bar plot displays the log2FC values for each perturbation.

(c) Barplots showing the distribution of cells from different perturbation groups in the clusters.

(d) Volcano plot showing the results of Fisher's exact test, comparing the distribution of cells from perturbation groups across Monocle-derived clusters. The plot displays −log10 adjusted p-values versus log2 fold changes relative to the control group.

(e) Table with Fisher's statistics.