UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C

*Mireia Ramos-Rodríguez*1,2, *Marc Subirana-Granés*1,2, and *Lorenzo Pasquali*1,2

1Equal contribution
2Endocrine Regulatory Genomics, Department of Experimental and Health Sciences, University Pompeu Fabra, 08003, Barcelona, Spain

*mireia.ramos@upf.edu* †*lorenzo.pasquali@upf.edu*

Supplementary Information

Package

UMI4Cats 1.1.10

Contents

1 UMI4Cats algorithms ................................. 2
   1.1 Collapsing duplicated UMIs ......................... 2
   1.2 Optimizing UMI-4C visualization: normalization, domainogram and adaptive smoothing trend .... 2
   1.3 Calling significant interactions .................... 3

2 Performance ........................................... 3

3 Installation ........................................ 4

4 UMI-4C analysis example ............................ 4
   4.1 Example datasets .................................... 5
   4.2 Preparing necessary files ............................ 5
   4.3 Processing UMI-4C FASTQ files .................. 7
   4.4 Loading UMI-4C data into R ....................... 8
   4.5 Calling significant interactions ................. 12
   4.6 Differential analysis ............................. 15
   4.7 Visualizing UMI-4C contact data .................. 17
   4.8 Session Information ............................... 20

References ............................................. 23
1 UMI4Cats algorithms

1.1 Collapsing duplicated UMIs

The UMI-4C experimental design allows the removal of PCR duplication bias using Unique Molecular Identifiers (UMIs) generated by the sonication process of the ligated fragments. Collapsing reads containing the same UMI allows removing the duplication bias, potential sequencing errors should also be taken into account in this process.

counterUMI4C() implements a UMI filtering algorithm which exploits both the UMI coordinates and sequence for inferring unbiased ligation events. Taking as input sequencing reads split by the restriction enzyme sequence and aligned to a reference genome, the algorithm runs the following steps:

1. For each read, the algorithm selects the 3’ end of the fragment as representative of that specific ligation event. This sequence, resulting from the sonication process, contains the sequence used as UMI.
2. All representative fragments that have the exact same coordinates are collapsed.
3. UMIs from the remaining representative fragments are extracted and compared. If they show < 2 mismatches, the corresponding representative fragments are collapsed.
4. The function recovers all fragments from the same ligation event if at least a fragment of such event is included in the collapsed list created in step 3.

1.2 Optimizing UMI-4C visualization: normalization, domainogram and adaptive smoothing trend

To optimize visualization of UMI-4C contact profiles, we applied methods of normalization and adaptive smoothing for generating both the domainogram and the contact profile trend, based on the methods described in the original UMI-4C publication (Schwartzman et al., 2016).

1. Normalize counts to a reference sample or group (getNormalizationMatrix()). This function creates a normalization matrix to scale UMI counts to a reference sample/group. Different scaling parameters are obtained from three different genomic windows around the viewpoint: 1-10 kb, 10-100 kb and 0.1-1 Mb. These values are then linearly smoothed in the 50 fragment ends next to the different window limits and divided by the reference sample/group.

2. Obtain sample/group domainograms (calculateDomainogram()). The domainogram contains normalized UMI counts in increasing window sizes, composed from 5 up to 150 restriction fragment ends. Normalized and scaled values from the previous step are used to obtain the mean normalized contact intensities in each window.

3. Produce sample/group adaptive smoothing trends (calculateAdaptiveTrend()). To plot the contact profile for each sample/group, an adaptive smoothing trend is used. A threshold factor is set to fix the number of restriction fragment ends to be combined into the same window. The algorithm will merge restriction fragment ends together until they contain the number of UMIs established by the threshold factor, and will return the mean normalized UMI counts in a window containing those specific restriction fragment ends.
1.3 Calling significant interactions

4C-seq data are affected by heteroscedasticity and a signal decay from the viewpoint. These characteristics, typical of 4C-seq experiments, have to be corrected before calling statistically significant interactions with the viewpoint. To deal with heteroscedasticity in UMI4Cats, a variance stabilizing transformation (VST) is applied to the raw counts. On the other hand, signal decay is modeled using a smooth monotone function. This method is based on work by Klein et al. (2015).

Variance stabilizing transformation (VST) of the raw counts. In 4C-seq experiments, the standard deviation across samples is large for fragments with high number of contact counts. Variance stabilizing transformation (VST) is used to remove the dependence of the variance on the mean, thus correcting the dependence of the standard deviations to the contact counts abundance. This VST is performed by varianceStabilizingTransformation() from DESeq2 (Love et al., 2014).

Monotone smoothing modeling. The 4C-seq signal decays with genomic distance from the viewpoint and converges towards a constant level of background. 4C-seq data reflects a smooth strictly increasing or strictly decreasing function. Thus, the general decay of the 4C-seq signal with genomic distance from the viewpoint is fitted using a symmetric monotone fit. The monotone smoothing function is calculated from the transformed raw counts, using the fda package (Ramsay and Silverman, 2005). This function is then used to generate the fitted count values.

Once the counts are VST transformed and the signal decay is fitted using a symmetric monotone fit, the z-scores are inferred to identify statistically significant interactions with the viewpoint.

Z-score calculation. Z-scores are calculated dividing the residuals values obtained from the VST-normalized and fitted counts, by the median absolute deviation (MAD) of all the sample’s residuals. Z-scores are then converted into one-sided p-values using the standard normal cumulative distribution function. Finally, a false discovery rate (FDR) multiple testing correction is performed to reduce type I error. Regions significantly interacting with the viewpoint can then be defined as those fragments with a significant adjusted p-values and passing the z-score threshold.

2 Performance

To assess the package performance we have used UMI-4C paired end FastQ files from one sample in two conditions, with a mean size of \( \approx 250 \) Mb containing \( 833243 \pm 85193 \) mean reads. The analysis was run in a workstation with the following specifications: Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz with 6 physical cores and 32Gb of RAM.

In Supplementary Table 1 we show the running time, in seconds, of the digestGenome() function. This function needs to be run only once for the same genome and restriction enzyme. In Supplementary Figure 1 we show the running time of a full analysis of the above-mentioned UMI-4C files, using an increasing number of threads.

Supplementary Table 1: Example of computational time for the generation of DpnII and Csp6i custom digested genome.

| Genome                  | Restriction enzyme | Computational time (seconds) |
|-------------------------|--------------------|------------------------------|
| BSgenome.Hsapiens.UCSC.hg19 | DpnII              | 62.4                         |
UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C

| Genome                  | Restriction enzyme | Computational time (seconds) |
|-------------------------|--------------------|------------------------------|
| BSgenome.Hsapiens.UCSC.hg19 | Csp6i              | 64.3                         |

Supplementary Figure 1: Running time of a typical UMI-4C analysis using UMI4Cats with different number of threads.

3 Installation

The latest release of UMI4Cats can be installed from Bioconductor:

```r
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("UMI4Cats")
```

The development version can be installed from the github repository:

```r
BiocManager::install("Pasquali-lab/UMI4Cats")
```

After installation, the package can be loaded using: `library(UMI4Cats)`.

4 UMI-4C analysis example

In this section we provide a full example of a typical analysis of UMI-4C data using the UMI4Cats package.

```r
library(UMI4Cats)
```
4.1 Example datasets

The datasets used in this example (obtained from Ramos-Rodríguez et al. (2019)) are available for download at the following location:

```
wget http://pasquallab.upf.edu/gbrowser/UMI4Cats_supplementary.tar.gz
tar -xzvf UMI4Cats_supplementary.tar.gz
```

```
path <- "UMI4Cats_supplementary/"
```

Briefly, the datasets correspond to human pancreatic islets exposed (cyt) or not (ctrl) to proinflammatory cytokines for 48 hours. In this example we are using the UMI-4C data generated from two different biological replicates (HI24 and HI32) using the promoter of the CIITA gene as a viewpoint.

4.2 Preparing necessary files

4.2.1 Demultiplexing FastQ files containing multiple baits

One of the many advantages of using the UMI-4C protocol is that it allows multiplexing of different baits starting from the same sample.

To facilitate the analysis, UMI4Cats provides a function for demultiplexing the paired-end FastQ files returned by the sequencing facility: demultiplexFastq().

This function requires the following inputs:

- Name of the R1 file as input – it will automatically detect the R2.
- Barcode sequences.
- Path and name for the output files.

The barcode sequences and names to be used for each output file need to be provided as a data.frame with column names sample and barcode.

```
## Input files
fastq <- file.path(path, "CIITA", "fastq", "ctrl_hi24_CIITA_R1.fastq.gz")

## Barcode info
barcodes <- data.frame(
  sample = c("CIITA"),
  barcode = c("GGACAAGCTCCCTGCAACTCA")
)

## Demultiplex baits inside FastQ file
demultiplexFastq(
  fastq = fastq, 
  barcodes = barcodes, 
  out_path = tempdir() 
)
```

The example FastQ file does not need demultiplexing, but the snippet above shows the steps to follow for demultiplexing a FastQ file.
4.2.2 Reference genome digestion

For the processing of the UMI-4C FastQ files it is necessary to construct a digested genome using the same restriction enzyme that was used in the UMI-4C experiments.

The UMI4Cats package includes the \texttt{digestGenome()} function to make this process easier. The function uses a \texttt{BSgenome} object as the reference genome, which is digested \textit{in silico} at a given restriction enzyme cutting sequence (\texttt{res_enz}). Besides the restriction sequence, it is also necessary to provide, as a zero-based numeric integer, the position at which the restriction enzyme cuts (\texttt{cut_pos}).

In Supplementary Table 2 you can see three examples of the different cutting sequences for \textit{DpnII}, \textit{Csp6I} and \textit{HindIII}.

### Supplementary Table 2: Example of different restriction enzymes and the input used by \texttt{digestGenome()}.

| Restriction enzyme | Restriction seq | res_enz | cut_pos |
|--------------------|-----------------|---------|---------|
| DpnII              | GATC            | GATC    | 0       |
| Csp6I              | G: TAC          | GTAC    | 1       |
| HindIII            | A: AGCTT        | AAGCTT  | 1       |

For this example, we are using the hg19 \texttt{BSgenome} object and we are going to digest it using the \textit{DpnII} enzyme.

```r

library(BSgenome.Hsapiens.UCSC.hg19)
## Loading required package: BSgenome
## Loading required package: Biostrings
## Loading required package: XVector
##
## Attaching package: 'Biostrings'
## The following object is masked from 'package:base':
##
##   strsplit
## Loading required package: rtracklayer

refgen <- BSgenome.Hsapiens.UCSC.hg19

hgl9_dpnii <- digestGenome(
  res_enz = "GATC",
  cut_pos = 0,
  name_RE = "dpnII",
  ref_gen = refgen,
  sel_chr = "chr16", # Select bait's chr (chr16) to make example faster
  out_path = file.path(path, "digested_genome/"))

## Generating digested genome using:
## > Restriction enzyme sequence: GATC
## > Restriction enzyme cut position: 0
## > Restriction enzyme name: dpnII
## > Reference genome: BSgenome.Hsapiens.UCSC.hg19
## > Output path: UMI4Cats_supplementary//digested_genome/
## Finished genome digestion.
```


UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C

The digested genome chromosomes will be saved in the `out_path` directory as RData files. The path of these files is outputted by the function, so that it can be saved as a variable (in this case `hg19_dpnii`) and used for downstream analyses.

4.3 Processing UMI-4C FASTQ files

Before doing any analysis, the paired-end reads stored in the FastQ files are converted to UMI counts in the digested fragments. These counts represent the contact frequencies of the viewpoint with that specific fragment. This process is implemented in the function `contactsUMI4C()`, which should be run in samples generated with the same experimental design (same viewpoint and restriction enzyme).

The function will consider all FastQ files in the same folder `fastq_dir` to be part of the same experiment (viewpoint + restriction enzyme). The R1 and R2 files for each sample need to contain `_R1` or `_R2` and one of the following FastQ suffixes: `.fastq`, `.fq`, `.fq.gz` or `.fastq.gz`.

For each experiment, the user needs to define 3 different sequences (Supplementary Figure 2):

- **Bait/viewpoint sequence** (`bait_seq`). This is the downstream primer sequence (DS primer) that matches the sequence of the queried viewpoint.
- **Padding sequence** (`bait_pad`). The padding sequence corresponds to the nucleotides between the DS primer end and the next restriction enzyme site.
- **Restriction enzyme sequence** (`res_enz`). This sequence is the sequence recognized by the restriction enzyme used in the experiment.

![Supplementary Figure 2: Schematic of a UMI-4C read detailing the different elements that need to be used as input for processing the data.](image)

Additionally, it is necessary to define the restriction enzyme cutting position (`cut_pos`) as previously done for the digested genome generation, together with the path of the corresponding digested genome (`digested_genome`) returned by the `digestGenome()` function.

`contactsUMI4C()` performs the alignment using the Rbowtie2 R package. It is thus needed to provide the corresponding reference genome indexes generated with Bowtie2 (Langmead and Salzberg, 2012). Is important to make sure that both the Bowtie2 index and the reference and digested genomes correspond to the same build (in this example, hg19).

```r
raw_dir <- file.path(path, "CIITA", "fastq")
index_path <- file.path(path, "ref_genome", "ucsc.hg19.chr16")

## Run main function to process UMI-4C contacts
contactsUMI4C(
  fastq_dir = raw_dir,
  wk_dir = file.path(path, "CIITA"),
)```
bait_seq = "GGACAAGCTCCCTGCAACTCA",
bait_pad = "GGACTTGCA",
res_enz = "GATC",
cut_pos = 0,
digested_genome = hg19.dpnii,
bowtie_index = index_path,
ref_gen = BSgenome.Hsapiens.UCSC.hg19::BSgenome.Hsapiens.UCSC.hg19,
sel_seqname = "chr16", # Input bait chr to reduce running time
threads = 4
)

Internally, contactsUMI4C() runs the following processes sequentially:

1. **FastQ files preparation** (**prepUMI4C**). In this processing step, only reads containing the bait_seq + bait_pad + res_enz are selected. Reads with mean Phread quality scores < 20 are filtered out.

2. **Split reads at restriction sites** (**splitUMI4C**). Using the res_enz sequence and the cutting position (cut_pos), all R1 and R2 reads are split to mimic the fragments generated experimentally.

3. **Align split reads to the reference genome** (**alignmentUMI4C**).

4. **Collapse reads using UMIs** (**counterUMI4C**). This step is done to count real molecular events and reduce artifacts due to PCR duplicates. The function returns contacts with restriction fragments < 5Mb from the viewpoint.

The final output of this process is a compressed tsv file stored in wk_dir/count, which contains the coordinates for each contact (viewpoint + contact) and the number of UMIs that support that specific interaction. These files will be used as input for the analyses performed in the following section.

### 4.3.1 Quality control measures

Once the processing step has been run, the statistics of the UMI-4C filtering, alignment and final number of UMIs can be generated from the logs returned by the contactsUMI4C() function. By using these logs, the function statsUMI4C() will produce a summary plot (Supplementary Figure 3) and a summary table with all statistics (wk_dir/logs/stats_summary.txt).

# Using the full dataset included in the package
statsUMI4C(wk_dir = file.path(path, "CIITA"))
## Joining, by = "sample_id"
## Joining, by = "sample_id"
## Using sample_id as id variables

### 4.4 Loading UMI-4C data into R

After processing the FastQ reads and obtaining UMI count tables, the next step is to analyze such data by detecting differential contacts and visualizing the genomic interactions.
4.4.1 Build the UMI4C object

The first step of the UMI-4C data analysis consists in loading the tables generated by the function `contactsUMI4C()` and use them to construct a UMI4C object, which is based on the `SummarizedExperiment` class. All these steps are performed automatically by the `makeUMI4C()` function.

The `makeUMI4C` will need as input, a data frame (`colData`) containing all relevant experiment information that will be needed for analyzing the data later on. The mandatory columns are:

1. `sampleID`: Unique identifier for the sample.
2. `replicate`: Replicate character or number identifier.
3. `condition`: Grouping variable for performing the differential analysis. For example: “control” and “treatment”, two different cell types, etc. The condition column should only have two different values. If more condition variables are provided, the differential analysis will fail.
4. `file`: Complete path and filename of the tsv files generated by `contactsUMI4C()`.

You can also include other additional columns to `colData`.

The UMI4C object will contain data from all the replicates. However, it might of interest group samples based on a specific variable, such as `condition`, to plot combined profiles or perform differential tests on merged replicates. The argument `grouping` controls this behavior. By default, the grouping argument is set to `grouping = "condition"`, which will group the samples according to the variables in the `condition` column. These grouped UMI4C object can be accessed using `groupsUMI4C(umi4c)$condition`. You can also add additional groupings to a specific UMI-4C object using the `addGrouping()` function or avoid the calculation of grouped sample setting `grouping = NULL`.

```r
# Load sample processed file paths
files <- list.files(file.path(path, "CIITA", "count"),
```
**UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C**

```r
library(dplyr)
library(tidyr)
library(S4Vectors)

# Create colData including all relevant information
colData <- data.frame(
  sampleID = gsub(".counts.tsv.gz", "", basename(files)),
  file = files,
  stringsAsFactors = FALSE
)

# Load UMI-4C data and generate UMI4C object
umi <- makeUMI4C(
  colData = colData,
  viewpoint_name = "CIITA",
  grouping = "condition",
  ref_umi4c = c("condition"="ctrl"),
  bait_expansion = 2e6
)

umi
```

The `makeUMI4C` function will perform the following steps to generate the `umi4c` object:

1. **Remove fragment ends around the bait**, as they are generally biased because of their proximity to the viewpoint. The value of the region that will be excluded from the analysis can be specified using the `bait_exclusion` argument. The default is 3 kb.
2. **Focus the scope** of the analysis in a specific genomic region around the bait, by default this is a 2Mb window centered on the viewpoint. The default value can be changed using the `bait_expansion` argument.

3. Sum the UMIs of the different samples belonging to the same group (defined by the `grouping` variable).

4. **Obtain the normalization matrices** that will be used to scale the groups to the reference, by default the group with less UMIs. If you want to avoid this normalization step, you can set `normalized` to `FALSE`.

5. Calculate the **domainograms** for each group.

6. Calculate the **adaptive trend** for each group.

### 4.4.2 Accessing UMI4C object information

The usual accessor functions from the `SummarizedExperiment-class` also work with the UMI4C class (for example; `assay`, `colData`, etc.). Other additional accessors have been created to retrieve different information:

- `dgram()`. Get a list of the domaingorams for each group.
- `bait()`. Retrieve a GRanges object with the bait position.
- `trend()`. Obtain a data.frame in long format with the adaptive smoothing trend.
- `resultsUMI4C()`. Retrieve results from the differential analysis. This only works if a differential analysis has been performed on the UMI4C object.

These functions can be used in the per-sample UMI4C object or in the grouped UMI4C object, which can be accessed using `groupsUMI4C(umi4c)$<grouping-variable>`. See an example below.

```r
groupsUMI4C(umi) # Available grouped UMI-4C objects
## List of length 1
## names(1): condition

head(assay(umi)) # Retrieve raw UMIs
## ctrl_hi24_CIITA ctrl_hi32_CIITA cyt_hi24_CIITA cyt_hi32_CIITA
## frag_24651 0 0 0 4
## frag_24652 0 0 0 0
## frag_24653 0 0 0 0
## frag_24654 0 0 0 0
## frag_24655 0 0 0 0
## frag_24656 0 0 0 0

head(assay(groupsUMI4C(umi)$condition)) # Retrieve UMIs grouped by condition
## ctrl cyt
## frag_24651 0 4
## frag_24652 0 0
## frag_24653 0 0
## frag_24654 0 0
## frag_24655 0 0
## frag_24656 0 0

colData(umi) # Retrieve column information
## DataFrame with 4 rows and 5 columns
## sampleID condition replicate viewpoint
## <character> <character> <character> <character>
## ctrl_hi24_CIITA ctrl_hi24_CIITA ctrl hi24 CIITA
```
4.5 Calling significant interactions

To identify significant interactions, the user needs to provide a set of regions that will be used to calculate the z-scores. These regions can be enhancers, open chromatin regions, a list of putative regulatory elements in the locus or, when no candidate regions are available, the user can take advantage of the `makeWindowFragments()` function to join a fixed number of restriction fragments into windows.

Next, these candidate regions (query_regions) and the UMI4C object need to be provided to the `callInteractions()` function. The function will return a GRangesList with each element corresponding to the specific z-scores and significance status of the query_regions in a specific sample. This information can be visualized by using the `plotInteractionsUMI4C()` function.

```r
rowRanges(umi) # Retrieve fragment coordinates

bait(umi) # Retrieve bait coordinates

4.5 Calling significant interactions
```
Finally, the function `getSignInteractions()` can be used to obtain a GRanges object with the regions that were found to be significant in at least one of the samples. This output can be used later to guide the identification of differential interactions.

```r
# Generate windows
win_frags <- makeWindowFragments(umi, n_frags=8)

# Call interactions
gr <- callInteractions(umi4c = umi,
  design = ~condition,
  query_regions = win_frags,
  padj_threshold = 0.01,
  zscore_threshold=2)

## converting counts to integer mode
##
## [2021-04-30 12:50:28] Starting vstUMI4C
## > Samples of DDS object:
## ctrl_hi24_CIITA, ctrl_hi32_CIITA, cyt_hi24_CIITA, cyt_hi32_CIITA
## [2021-04-30 12:50:28] Finished vstUMI4C
##
## [2021-04-30 12:50:28] Starting smoothMonotoneUMI4C using:
## > Samples of DDS object:
## ctrl_hi24_CIITA, ctrl_hi32_CIITA, cyt_hi24_CIITA, cyt_hi32_CIITA
## > Alpha:
## 20
## > Penalty:
## 0.1
## [2021-04-30 12:50:33] Finished smoothMonotoneUMI4C

# Plot interactions
all <- plotInteractionsUMI4C(umi, gr, grouping = NULL,
  significant=FALSE, # Plot all regions
  xlim=c(10.75e6, 11.1e6))

## Joining, by = "tx_id"
sign <- plotInteractionsUMI4C(umi, gr, grouping = NULL,
  significant=TRUE, # Plot only significant regions
  xlim=c(10.75e6, 11.1e6))

## Joining, by = "tx_id"
cowplot::plot_grid(all, sign, ncol=2, labels=c("All", "Significant"))
```
# Obtain unique significant interactions

```r
inter <- getSignInteractions(gr)
```

## Results

| Iter. | PENSSE  | Grad  | Length | Intercept | Slope |
|-------|---------|-------|--------|-----------|-------|
| 0     | 0.2852  | 0.0177| 4.743  | -0.4549   |       |
| 1     | 0.2701  | 0.0144| 5.5879 | -0.595    |       |
| 2     | 0.2633  | 0.0095| 6.1463 | -0.6783   |       |
| 3     | 0.2614  | 0.0042| 6.3699 | -0.7173   |       |
| 4     | 0.2608  | 0.0034| 6.5882 | -0.7545   |       |
| 5     | 0.2607  | 0.0011| 6.642  | -0.7641   |       |

## Results

| Iter. | PENSSE  | Grad  | Length | Intercept | Slope |
|-------|---------|-------|--------|-----------|-------|
| 0     | 0.2551  | 0.0164| 5.0961 | -0.4904   |       |
| 1     | 0.2459  | 0.012 | 5.6846 | -0.592    |       |
| 2     | 0.2415  | 0.0087| 6.0859 | -0.6555   |       |
| 3     | 0.2402  | 0.0039| 6.2613 | -0.687    |       |
| 4     | 0.2397  | 0.0034| 6.4243 | -0.7157   |       |
| 5     | 0.2396  | 0.0012| 6.4736 | -0.7248   |       |

## Results

| Iter. | PENSSE  | Grad  | Length | Intercept | Slope |
|-------|---------|-------|--------|-----------|-------|
| 0     | 0.2869  | 0.0141| 4.56   | -0.4171   |       |
| 1     | 0.276   | 0.0098| 5.2044 | -0.5267   |       |
| 2     | 0.2712  | 0.0059| 5.6556 | -0.597    |       |
| 3     | 0.2699  | 0.0027| 5.8335 | -0.6286   |       |
| 4     | 0.2695  | 0.0017| 5.998  | -0.6574   |       |

## Results
## Iter. PENSSE Grad Length Intercept Slope
## 0 0.296 0.0214 5.0541 -0.4851
## 1 0.281 0.0115 5.6543 -0.5882
## 2 0.2732 0.0093 6.4732 -0.7064
## 3 0.2711 0.002 6.5915 -0.7265
## 4 0.2708 0.0011 6.7898 -0.7607

### 4.6 Differential analysis

Once the UMI4C object is generated, you can perform a differential analysis between conditions using two different approaches. In both cases you can provide `query_regions`, such as enhancers, open chromatin regions or the output of `getSignInteractions()` to focus the analysis in regions that are more likely to have differential interactions.

- **DESeq2's Wald Test** (`waldUMI4C()`). We recommend using this test to detect significant differences, as it performs a more sophisticated modeling and testing of count data (Love et al., 2014). To obtain reliable results we recommend using several replicates with a high number of UMIs per sample.

- **Fisher’s Exact Test** (`fisherUMI4C()`). In some instances, having insufficient replicates or UMI depth precludes the use of dESeq2’s Wald Test. In such cases, the user can opt for the Fisher’s Exact Test, which can provide useful results when used in a set of candidate regions, such as enhancers or the output of `callInteractions()`.

Taking into account that, for this example, we have two replicates per condition and high number of UMIs, we will use the `waldUMI4C()` function.

```r
umi_wald <- waldUMI4C(umi,
  query_regions = inter,
  design = ~condition)
```

#### 4.6.1 Retrieve differential analysis results

Results from both Fisher’s Exact test and DESeq2 can be retrieved using the `resultsUMI4C()` on the UMI4C object returned by both functions.

```r
resultsUMI4C(umi_wald, ordered = TRUE, counts = TRUE, format = "data.frame")
```

```r
# seqnames start end id mcols.position baseMean
# 6  chr16 10927967 10930640 window_UP_16 upstream 35.080264
# 9  chr16 11371374 11372607 window_DOWN_131 downstream 4.213888
# 10 chr16 11634946 11636286 window_DOWN_229 downstream 3.349065
```
UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C

## 4 chr16 10862415 10865302 window_UP_44 upstream 14.838766
## 11 chr16 11679115 11681791 window_DOWN_249 downstream 4.638962
## 2 chr16 10528949 10525743 window_UP_169 upstream 7.977030
## 3 chr16 10860814 10862414 window_UP_45 upstream 16.683244
## 7 chr16 10954198 10956424 window_UP_6 upstream 37.204416
## 1 chr16 10261046 10263505 window_UP_269 upstream 8.857582
## 8 chr16 11008168 11003892 window_DOWN_10 downstream 25.568255
## 5 chr16 10867157 10870182 window_UP_42 upstream 14.190911

| baseVar | allZero | dispGeneEst | dispGeneIter | dispFit | dispersion | dispIter |
|---------|---------|-------------|--------------|---------|------------|----------|
| 6       | FALSE   | 0.00000001  | 1            | 0.05945454 | 0.05479368 | 6        |
| 9       | FALSE   | 0.00000001  | 29           | 0.28971133 | 0.26925526 | 5        |
| 10      | FALSE   | 0.00000001  | 2            | 0.14773241 | 0.14105146 | 5        |
| 4       | FALSE   | 0.10743298  | 16           | 0.13535952 | 0.13347989 | 7        |
| 11      | FALSE   | 0.38357459  | 61           | 0.38357459 | 0.38398939 | 4        |
| 2       | FALSE   | 1.48293947  | 11           | 1.48274693 | 1.48301566 | 3        |
| 3       | FALSE   | 0.52213118  | 3            | 0.52129057 | 0.52152354 | 4        |
| 7       | FALSE   | 0.07911394  | 9            | 0.07911394 | 0.07913127 | 3        |
| 1       | FALSE   | 1.82664245  | 8            | 1.75547312 | 1.82664245 | 6        |
| 8       | FALSE   | 0.04042483  | 100          | 0.07313249 | 0.07804989 | 9        |
| 5       | FALSE   | 0.00000001  | 1            | 0.08302684 | 0.07265074 | 6        |

| dispOutlier | dispMAP | Intercept | condition_cyt_vs_ctrl | SE_Intercept |
|-------------|---------|-----------|------------------------|--------------|
| 6           | FALSE   | 0.05479368 | 3.7448040              | 2.07807832   | 0.3466634 |
| 9           | FALSE   | 0.26925526 | 3.0719727              | -5.42164502  | 0.6145912 |
| 10          | FALSE   | 0.14105146 | 2.6568294              | -3.56439438  | 0.5248662 |
| 4           | FALSE   | 0.13347898 | 4.3952911              | -1.30665668  | 0.4218429 |
| 11          | FALSE   | 0.38398939 | 0.4948203              | 2.50932010   | 0.9859333 |
| 2           | FALSE   | 1.48301566 | 1.6844718              | 1.99759539   | 1.3416063 |
| 3           | FALSE   | 0.52152354 | 3.2665405              | 1.29023289   | 0.7927548 |
| 7           | FALSE   | 0.07913127 | 5.4277780              | -0.45631357  | 0.3185374 |
| 1           | TRUE    | 1.76799770 | 2.1532894              | 1.58561324   | 1.4445922 |
| 8           | FALSE   | 0.07004989 | 4.8551578              | -0.36740801  | 0.3181217 |
| 5           | FALSE   | 0.07265074 | 3.8084033              | 0.03980409   | 0.3655521 |

| SE_condition_cyt_vs_ctrl | WaldStatistic | WaldPvalue | Intercept |
|--------------------------|---------------|-----------|-----------|
| 6                        | 0.440512      | 10.928492 | 0.928492  |
| 9                        | 1.6550790     | 4.9983996 | 0.9983996 |
| 10                       | 1.5785471     | 5.0619756 | 5.0619756 |
| 4                        | 0.6688360     | 10.419260 | 10.419260 |
| 11                       | 1.2288324     | 0.5018802 | 0.5018802 |
| 2                        | 1.8519517     | 1.2555635 | 1.2555635 |
| 3                        | 1.1037954     | 4.1204926 | 4.1204926 |
| 7                        | 0.4681855     | 17.039603 | 17.039603 |
| 1                        | 2.0176436     | 1.4905997 | 1.4905997 |
| 8                        | 0.4728372     | 15.261952 | 15.261952 |
| 5                        | 0.5350755     | 10.418224 | 10.418224 |

| WaldStatistic_condition_cyt_vs_ctrl | WaldPvalue | Intercept |
|-------------------------------------|-----------|-----------|
| 6                                   | 4.7174028 | 8.423617e-28 |
| 9                                   | -3.2756211 | 5.788080e-07 |
| 10                                  | -2.25802214 | 4.149342e-07 |
| 4                                   | -1.95362792 | 2.025227e-25 |
| 11                                  | 2.04203616 | 6.157518e-01 |
| 2                                   | 1.07864337 | 2.092743e-01 |
## WaldPvalue
| condition_cyt_vs_ctrl | betaConv | betaIter | deviance | maxCooks |
|-----------------------|----------|----------|----------|----------|
| 6                     | TRUE     | 3        | 25.90544 | NA       |
| 9                     | TRUE     | 2        | 11.99684 | NA       |
| 10                    | TRUE     | 5        | 13.13145 | NA       |
| 4                     | TRUE     | 3        | 24.63113 | NA       |
| 11                    | TRUE     | 6        | 17.92462 | NA       |
| 2                     | TRUE     | 6        | 24.82048 | NA       |
| 3                     | TRUE     | 4        | 29.13969 | NA       |
| 7                     | TRUE     | 3        | 29.9677  | NA       |
| 1                     | TRUE     | 7        | 26.41241 | NA       |
| 8                     | TRUE     | 5        | 25.53186 | NA       |
| 5                     | TRUE     | 2        | 1.41393  | NA       |

## ctrl_h24 CIITA ctrl_h32 CIITA cyt_h24 CIITA cyt_h32 CIITA
| pvalue |
|--------|
| 6      | 14.58799 12.46577 47.435794 65.831493 2.388669e-06 |
| 9      | 8.752795 8.102756 0.000000 0.000000 1.053773e-03 |
| 10     | 4.82664 7.479467 1.054129 0.000000 2.39428e-02 |
| 4      | 24.313321 18.075379 12.649545 4.316819 5.07424e-02 |
| 11     | 0.000000 2.493156 5.270644 10.792048 4.119749e-02 |
| 2      | 0.000000 6.232889 9.487159 16.188072 2.807467e-01 |
| 3      | 16.533058 3.116445 16.866060 30.217734 2.42416e-01 |
| 7      | 43.763977 42.383647 42.165151 20.504891 3.297357e-01 |
| 1      | 0.000000 8.726045 11.595416 15.108867 4.319414e-01 |
| 8      | 24.313321 33.034313 16.866060 28.053225 4.371240e-01 |
| 5      | 13.615460 14.335645 13.703674 15.108867 9.407003e-01 |

## log2FoldChange p.adj sign
| 6 | 2.0780732 2.627535e-05 TRUE |
| 9 | -5.42164502 5.79570e-03 TRUE |
| 10 | -3.56439438 8.779569e-02 FALSE |
| 4 | -1.3066568 1.163955e-01 FALSE |
| 11 | 2.50932010 1.163955e-01 FALSE |
| 2 | 1.99759599 4.41734e-01 FALSE |
| 3 | 1.29023289 4.41734e-01 FALSE |
| 7 | -0.45631357 4.53891e-01 FALSE |
| 1 | 1.58561324 4.808562e-01 FALSE |
| 8 | -0.36748081 4.808562e-01 FALSE |
| 5 | 0.03980489 9.407003e-01 FALSE |

The parameter `counts` indicates whether raw counts used for the test should be outputted. In Fisher’s Exact Test, `umis_ref` corresponds to the number of raw UMIs from the sample/group used as reference (accessible through `metadata(umi_dif)$ref_umi4c`).

### 4.7 Visualizing UMI-4C contact data

Once the `UMI4C` object is created the chromatin interactions can be visualized using the `plotUMI4C()` function.
The gene annotations will be extracted from the TxDb.Hsapiens.UCSC.hg19.knownGene package by default. Make sure that the annotations coincide with your reference genome. You can check the package GenomicFeatures for more information on available TxDb objects. The domainogram plotting is controlled by the dgram_plot argument. If you set it to FALSE, the domainogram will not be plotted.

In case you are interested in plotting the profiles of the different samples contained in your experiment, you just need to set the grouping argument to NULL, which will disable sample grouping (Supplementary Figure 4).

```r
plotUMI4C(umi, grouping = NULL, TxDb = TxDb.Hsapiens.UCSC.hg19.knownGene::TxDb.Hsapiens.UCSC.hg19.knownGene, dgram_plot = FALSE, xlim = c(10.75e6, 11.25e6))
```

Supplementary Figure 4: CIITA locus plot produced by the UMI4Cats package.

If the UMI4C object contains information on the differential contacts, this data will be shown in the plot as well (Supplementary Figure 5). The grouping argument uses the grouped trends and domainograms stored in groupSU4C(). If you want to add a new grouping, you can use the addGrouping() function.

```r
plotUMI4C(umi_wald, grouping = "condition",
```

---

**Supplementary Figure 4:** CIITA locus plot produced by the UMI4Cats package.
Supplementary Figure 5: CIITA locus plot containing differential contacts visualization produced by the UMI4Cats package.
4.8 Session Information

```r
sessionInfo()
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## **Matrix products:** default
## **BLAS:** /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## **LAPACK:** /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## **locale:**
## [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
## [3] LC_TIME=es_ES.UTF-8 LC_COLLATE=C
## [5] LC_MONETARY=es_ES.UTF-8 LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=es_ES.UTF-8 LC_NAME=C
## [9] LC_ADDRESS=C LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## **attached base packages:**
## [1] parallel stats4 stats graphics grDevices utils datasets
## [8] methods base
##
## **other attached packages:**
## [1] org.Hs.eg.db _3.12.0 AnnotationDbi _1.52.0
## [3] tidyr _1.1.3 BSgenome.Hsapiens.UCSC.hg19 _1.4.3
## [5] Biostrings _2.58.0 rtracklayer _1.50.0
## [7] Biocstrings _2.58.0 XVector _0.30.0
## [9] UMI4Cats _1.1.10 SummarizedExperiment _1.20.0
## [11] Biobase _2.50.0 GenomicRanges _1.42.0
## [13] GenomeInfoDb _1.26.7 IRanges _2.24.1
## [15] S4Vectors _0.28.1 BiocGenerics _0.36.0
## [17] BiocStyle _2.18.1
##
## **loaded via a namespace (and not attached):**
## [1] colorspace _2.0-0
## [2] ellipsis _0.3.1
## [3] mclust _5.4.7
## [4] farver _2.1.0
## [5] bit64 _4.0.5
## [6] fansi _0.4.2
## [7] mvtnorm _1.1-1
## [8] xml2 _1.3.2
## [9] splines _4.0.2
## [10] cachem _1.0.4
## [11] geneplotter _1.68.0
## [12] knitr _1.31
## [13] Rsamtools _2.6.0
## [14] annotate _1.68.0
## [15] cluster _2.1.1
```
## 

```r
## [16] dbplyr_2.1.1
## [17] BiocManager_1.30.12
## [18] compiler_4.0.2
## [19] httr_1.4.2
## [20] rainbow_3.6
## [21] assertthat_0.2.1
## [22] Matrix_1.3-2
## [23] fastmap_1.1.0
## [24] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
## [25] htmltools_0.5.1.1
## [26] prettyunits_1.1.1
## [27] tools_4.0.2
## [28] gtable_0.3.0
## [29] glue_1.4.2
## [30] GenomeInfoDbData_1.2.4
## [31] reshape2_1.4.4
## [32] dplyr_1.0.5
## [33] rappdirs_0.3.3
## [34] Rcpp_1.0.6
## [35] vctrs_0.3.7
## [36] xfun_0.22
## [37] stringr_1.4.0
## [38] lifecycle_1.0.0
## [39] XML_3.99-0.6
## [40] zlibbioc_1.36.0
## [41] MASS_7.3-53.1
## [42] zoo_1.8-9
## [43] scales_1.1.1
## [44] hms_1.0.0
## [45] RColorBrewer_1.1-2
## [46] yaml_2.2.1
## [47] curl_4.3
## [48] memoise_2.0.0
## [49] ggplot2_3.3.3
## [50] biomaRt_2.47.5
## [51] stringi_1.5.3
## [52] RSQLite_2.2.7
## [53] genefilter_1.72.1
## [54] pcaPP_1.9-73
## [55] GenomicFeatures_1.42.3
## [56] BiocParallel_1.24.1
## [57] hdrcde_3.4
## [58] rlang_0.4.10
## [59] pkgconfig_2.0.3
## [60] bitops_1.0-6
## [61] evaluate_0.14
## [62] fda_5.1.9
## [63] lattice_0.20-41
## [64] purrr_0.3.4
## [65] GenomicAlignments_1.26.0
## [66] ks_1.12.0
```
UMI4Cats: an R package to analyze chromatin contact profiles obtained by UMI-4C

```r
## [67] labeling_0.4.2
## [68] cowplot_1.1.1
## [69] bit_4.0.4
## [70] tidyselect_1.1.0
## [71] plyr_1.8.6
## [72] magrittr_2.0.1
## [73] bookdown_0.21
## [74] DESeq2_1.30.1
## [75] R6_2.5.0
## [76] magick_2.7.1
## [77] generics_0.1.0
## [78] DelayedArray_0.16.3
## [79] DBI_1.1.1
## [80] pillar_1.6.0
## [81] survival_3.2-10
## [82] Rcurl_1.98-1.3
## [83] tibble_3.1.0
## [84] crayon_1.4.1
## [85] KernSmooth_2.23-18
## [86] utf8_1.2.1
## [87] BioCFileCache_1.14.0
## [88] rmarkdown_2.7
## [89] fds_1.8
## [90] progress_1.2.2
## [91] locfit_1.5-9.4
## [92] grid_4.0.2
## [93] blob_1.2.1
## [94] digest_0.6.27
## [95] xtable_1.8-4
## [96] regioneR_1.22.0
## [97] openssl_1.4.3
## [98] munsell_0.5.0
## [99] askpass_1.1
```
References

Klein, F.A. et al. (2015) FourCSeq: Analysis of 4C sequencing data. *Bioinformatics*, **31**, 3085–3091.

Langmead, B. and Salzberg, S.L. (2012) Fast gapped-read alignment with bowtie 2. *Nature Methods*, **9**, 357–359.

Love, M.I. et al. (2014) Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, **15**, 550.

Ramos-Rodríguez, M. et al. (2019) The impact of proinflammatory cytokines on the β-cell regulatory landscape provides insights into the genetics of type 1 diabetes. *Nature Genetics*, **51**, 1588–1595.

Ramsay, J. and Silverman, B.W. (2005) Functional data analysis 2nd ed. Springer-Verlag, New York.

Schwartzman, O. et al. (2016) UMI-4C for quantitative and targeted chromosomal contact profiling. *Nature Methods*, **13**, 685–691.