MOSim

Sonia Tarazona, Carlos Martínez

May 19, 2021

Package

MOSim 1.7.0

Contents

1 Introduction

MOSim package simulates multi-omic experiments that mimic regulatory mechanisms within the cell. Gene expression (RNA-seq count data) is the central data type and the rest of available omic data types act as regulators of genes and include ATAC-seq (DNase-seq), ChIP-seq, small RNA-seq and Methyl-seq. In addition, transcription factor (TF) regulation can also be modeled.

MOSim algorithm returns the simulated count data matrices, regulatory connections between genes and omic features and a detailed description of the simulation settings. Thus, these results can be used to test new integration methods, to tune or prepare analysis pipelines, or as example data in users' manuals or for teaching purposes.

MOSim requires a seed count dataset for each omic to be simulated. For regulatory omics and TFs, an association table linking genes to regulators must also be given. For convenience, *MOSim* includes default datasets from STATegra project (mouse data) for all omics, so the package can be still used in absence of custom data.

Due do the potentially great amount of information generated, multiple helper functions are available for both passing the necessary input data and retrieving the generated data.

The outcome of the simulation process depends on two types of input information: the specific parameters to the omics to be simulated and the experimental design.

The experimental design options are flexible. The user can choose the number of experimental groups and the number of replicates. Time series data can also be simulated and, again, the user decided the number of time points.

The process starts by simulating RNA-seq (gene expression) data. To do that, the program takes a sample from the supplied identifiers (row names of the initial count dataset) and labels them as differentially expressed genes (DEG). The percentage of DEG can be configured by the user, as many other settings that will be described in this vignette. A gene is considered to be differentially expressed if the expression of the gene changes (i) between the reference experimental group (group 1) and at least one of the remaining groups in the experimental design or (ii) across time.

When time course data is to be simulated, *MOSim* assigns one of the following profiles to each of the DEGs (Figure 1) in each of the experimental groups:

Continuous induction lineal increase of the activity of the gene with time.

Continuous repression lineal decrease of the activity of the gene with time.

Transitory induction inactive gene at the initial time, with progressive increasing of the activity followed by a decrease.

Transitory repression active gene at the initial time, with progressive decreasing of the activity followed by an increase.

Flat the activity of the gene remains constant along time.

If a DEG is assigned to a flat profile in all groups, the algorithm will model a change in expression for at least one of the experimental groups (up or down regulation).

For other experimental designs not including time series, all DEGs are labeled as flat and the change in expression is modeled as indicated above, that is, for at least one of the experimental groups with regard to the first group.

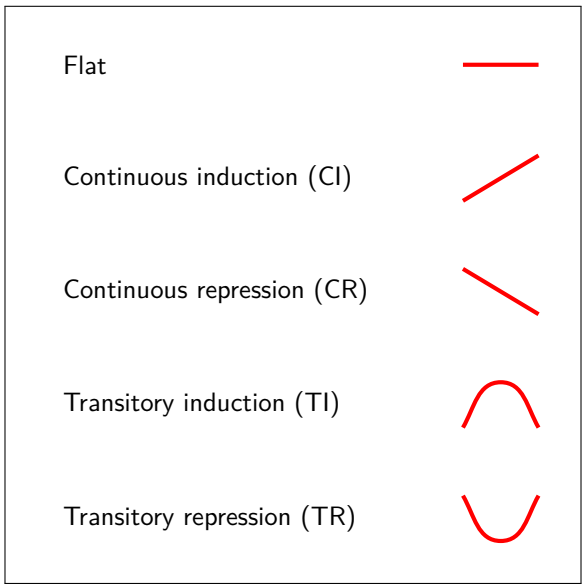


Figure 1: Time profiles representation

Once the gene expression settings have been set the regulatory omics are configured. This is done first by assigning a potential effect (activation, repression or none) to each regulator. In a similar way as the percentage of DEG could be specified for RNA-seq, in regulatory omics the user can indicate the percentage of regulators that should be activators, repressors or with no effect in each omic. However, these initial percentages can be affected because regulatory relationships must be coherent with assigned profiles or selection of DEG. For instance, when a regulator has effect on more than one DEG and these DEGs have opposite profiles, the type of effect must be forced to match the profiles. In fact, and related to this, if a regulator is potentially associated to more than one DEG with different profiles, the system has to decide which profile must be assigned to the regulator. For that, *MOSim* takes the profile class (combination of profiles for all groups, for example CI-FL-CR in groups 1, 2 and 3 respectively) with more DEGs associated to the regulator (majority class) and assigns the corresponding profile to the regulator: depending on the regulator effect, the profile will be the same for activation effect (following the example, CI-FL-CR for groups 1, 2 and 3) or the opposite for repression (CR-FL-CI). The interactions with genes not included in the majoritary class are automatically classified as "activator" if the profiles are the same, "repressor" if the are the totally opposite (CI vs CR, TI vs TR) or "no effect" in any other case.

In brief, *MOSim* usage can be summed up in 3 main steps that will be described in the next sections:

1. Decide the experimental design, omics list and input data to use.
2. Generate a simulation object using the wrapper function `mosim`, in combination with the methods `omicData` and `omicSim`.
3. Extract the results from the simulation object with the helper functions `omicResults` and `omicSettings`.

2 *MOSim* input parameters

For the experimental design of the simulation, there are 3 parameters to be set: the number of experimental groups or conditions, the number of time points (1, if time series are not to be considered) and the number of replicates per condition. The only requirements, so that the algorithm can simulate differential expression, are at least 2 groups with no time points, or 1 group with at least 2 time points.

The list of omics to be simulated must be also provided. At this moment *MOSim* supports the following data types:

- RNA-seq (compulsory)
- DNase-seq
- ChIP-seq
- Methyl-seq
- miRNA-seq

The simulation needs gene expression to be present so RNA-seq will be included always even if it is not specified by the user. The simulation of transcription factors is also supported as a subset of RNA-seq simulated data.

Optionally, the seed samples to start the simulation can also be given. They are used for extracting the feature identifiers from the row names, and as the initial count distribution to generate the rest of simulated samples. The algorithm includes mouse default samples from STATegra project but users may provide seed samples from any other organism or experiment. For each regulatory omic, the association list linking regulator IDs to genes that they potentially regulate is also needed. If provided, extra care must be taken to ensure that the identifiers between RNA-seq and the association lists are correctly matched. Again, for STATegra data, these association files are included in the package. If TF-target gene associations are provided, TF regulation will be also simulated.

The structure of the custom data and how to correctly pass it to *MOSim* will be described in the following sections.

3 Running the simulation: `mosim`

MOSim simulations are stored in a custom class S4 object, which means that the information is contained in slots and can be accessed using the standard way (with the operator `@`). However, this is not recommended and the preferred way to access the information are the accessors or utility function provided by the package, as additional transformations can be applied by these.

The first of these functions is `mosim`. This helper method takes all the options, performs the simulation, and returns the simulation object.

Internally, this helper function creates a series of S4 objects according to the options passed by the user and calls the required methods to simulate the data, gather the results and return them. The user only needs to set the simulation options, as indicated in the following example:

```
mosim(omics, omicsOptions = NULL, diffGenes = .15, numberReps = 3, numberGroups =
2, times = c(0, 2, 4, 12, 24), depth = 74, profileProbs = list( continuous.induction
= .235, continuous.repression = .235, transitory.induction = .235, transitory.repression
= .235, flat = .06 ), minMaxFC = c(3, 8), TFtoGene = NULL)
```

The main arguments accepted by `mosim` function are:

omics Character vector containing the names of the omics to simulate, which can be "RNA-seq", "miRNA-seq", "DNase-seq", "ChIP-seq" or "Methyl-seq" (e.g. c("RNA-seq", "miRNA-seq")). It can also be a list with the omic names as names and their options as values, but we recommend to use the argument omicsOptions to provide the options to simulate each omic.

omicsOptions List containing the options to simulate each omic. We recommend to apply the helper method `omicSim` to create this list in a friendly way, and the function `omicData` to provide custom data (see the related sections for more information). Each omic may have different configuration parameters, but the common ones are:

- **simuData/idToGene** Seed sample and association tables for regulatory omics. The helper function `omicData` should be used to provide this information (see the following section).
- **regulatorEffect** For regulatory omics. List containing the percentage of effect types (repressor, activator or no effect) over the total number of regulators. For example `list('repressor' = 0.05, 'NE' = 0.95)`. Remember that these numbers might be modified by the algorithm as explained in section 1.
- **totalFeatures** Number of features to simulate. By default, the total number of features in the seed dataset.
- **depth** Sequencing depth in millions of reads. If not provided, it takes the global parameter passed to `mosim` function.
- **replicateParams** List with parameters a and b for adjusting the variability in the generation of replicates using the negative binomial. See section 5 for more information.

diffGenes Number of differentially expressed genes to simulate, given in percentage (0 - 1) or in absolute number (> 1).

numberGroups Number of experimental groups or conditions to simulate.

numberReps Number of replicates per experimetal condition (and time point, if time series are to be generated).

times Vector of time points to consider in the experimental design.

depth Sequencing depth in millions of reads.

minMaxFC Vector of minimum and maximum fold-change allowed for differentially expressed features.

TFtoGene A logical value indicating if default transcription factors data should be used (TRUE) or not (FALSE), or a 3 column data frame containing custom associations as explained in 3.1. For transcription factors, the count matrix is not simulated like in the other omics but extracted from RNA-seq simulated data.

The most basic example of `mosim` function usage is calling it with only the list of omics to simulate. In this case, the default values will be used, including the default data samples.

```
library(MOSim)

omic_list <- c("RNA-seq")

rnaseq_simulation <- mosim(omics = omic_list)
```

The `rnaseq_ simulation` object is a *Simulation* class object containing the simulated data, that can be easily accessed with the helper functions `omicResults` and `omicSettings`, as we will see in the corresponding section.

Following with that basic example above, we can modify the experimental design to simulate 2 groups, 1 time point and 4 replicates per group:

Invalid combinations of experimental design arguments will make the algorithm stop with an error message, like this:

```
rnaseq_simulation <- mosim(omics = c("RNA-seq"),
                           times = 0,
                           numberGroups = 1,
                           numberReps = 4)
```

```
## Error in validObject(.Object):  invalid class "MOSimulation" object:  The design
must have a minimum of 2 times or 2 groups.
```

To obtain more than one omic data type, the omics list must be modified. RNA-seq is mandatory, and will be automatically included in the simulation, no matter if it is listed or not. Therefore these two simulations would be equivalent:

As it can be seen, `mosim` function accepts global simulation parameters, but specific settings for a particular omic should be provided through two specially designed functions: `omicData` and `omicSim`.

3.1 Providing custom data: `omicData`

The function `omicData` was designed to help users to provide their own seed data sets, as follows:

```
omicData(omic, data, associationList = NULL)
```

This helper function accepts 3 parameters:

omic The name of the omic data type whose seed sample is to be provided. The omic names must be included in the list of accepted omics.

data Count data. A data frame or ExpressionSet object with the omic identifiers as row names and just one column named `Counts`, containing the counts to be used as seed sample in the simulation for that omic.

associationList Only for regulatory omics. Data frame with 2 columns containing the potential associations between genes and regulators. The first column, called `ID`, must contain the regulator IDs, and the second column, called `Gene`, must contain the gene identifiers.

For illustration purposes, consider as our custom data a subset of the default gene expression dataset. To use it as our seed RNA-seq dataset, we can use this code:

```
# Take a subset of the included dataset for illustration
# purposes. We could also load it from a csv file or RData,
# as long as we transform it to have 1 column named "Counts"
# and the identifiers as row names.
data("sampleData")

custom_rnaseq <- head(sampleData$SimRNAseq$data, 100)

# In this case, 'custom_rnaseq' is a data frame with
# the structure:
head(custom_rnaseq)
##                    Counts
## ENSMUSG00000000001   6572
## ENSMUSG00000000003      0
## ENSMUSG00000000028   4644
## ENSMUSG00000000031      8
## ENSMUSG00000000037      0
## ENSMUSG00000000049      0


# The helper 'omicData' returns an object with our custom data.
rnaseq_customdata <- omicData("RNA-seq", data = custom_rnaseq)

# We use the associative list of 'omics' parameter to pass
# the RNA-seq object.
rnaseq_simulation <- mosim(omics = list("RNA-seq" = rnaseq_customdata))
```

RNA-seq is a special case of omic data type because it does not require an association list to work. For any other omic, such as DNase-seq, we need two different data frames: the seed sample with the structure already mentioned, and the associations between regulator IDs and genes.

```
# Select a subset of the available data as a custom dataset
data("sampleData")

custom_dnaseseq <- head(sampleData$SimDNaseseq$data, 100)

# Retrieve a subset of the default association list.
dnase_genes <- sampleData$SimDNaseseq$idToGene
dnase_genes <- dnase_genes[dnase_genes$ID %in%
                             rownames(custom_dnaseseq), ]

# In this case, 'custom_dnaseseq' is a data frame with
# the structure:
head(custom_dnaseseq)
##                        Counts
## 1_63176480_63177113       513
## 1_125435495_125436168    1058
## 1_128319376_128319506      37
## 1_139067124_139067654     235
## 1_152305595_152305752     105
```

```
## 1_172490322_172490824    290

# The association list 'dnase_genes' is another data frame
# with the structure:
head(dnase_genes)
##                       ID              Gene
## 29195 1_3670777_3670902 ENSMUSG00000051951
## 29196 1_3873195_3873351 ENSMUSG00000089420
## 29197 1_4332428_4332928 ENSMUSG00000025900
## 29198 1_4346315_4346445 ENSMUSG00000025900
## 29199 1_4416827_4416973 ENSMUSG00000025900
## 29200 1_4516660_4516798 ENSMUSG00000096126

dnaseseq_customdata <- omicData("DNase-seq",
                                data = custom_dnaseseq,
                                associationList = dnase_genes)

multi_simulation <- mosim(omics = list(
    "RNA-seq" = rnaseq_customdata,
    "DNase-seq" = dnaseseq_customdata)
    )

## Warning:  ‘distinct_()‘ was deprecated in dplyr 0.7.0.
## Please use ‘distinct()‘ instead.
## See vignette('programming') for more help
```

The two exceptions in this section are transcription factors and methylation.

The simulated transcription factor data is extracted from the generated RNA-seq data but a data frame with 3 columns needs to be provided: `TF` column, with the transcription factor identifiers (any type of identifier can be used); `TFgene` column, with transcription factor identifiers that must coincide with the type of identifier used in RNA-seq; and `LinkedGene` column, with the identifier of the target gene (again, the same used in RNA-seq data). Instead of applying `omicData` function for this purpose, the `TFtoGene` argument in `mosim` function must be used:

```
# Select a subset of the available data as a custom dataset
data("sampleData")

custom_tf <- head(sampleData$SimRNAseq$TFtoGene, 100)
#      TF             TFgene         LinkedGene
# 1  Aebp2 ENSMUSG00000030232 ENSMUSG00000000711
# 2  Aebp2 ENSMUSG00000030232 ENSMUSG00000001157
# 3  Aebp2 ENSMUSG00000030232 ENSMUSG00000001211
# 4  Aebp2 ENSMUSG00000030232 ENSMUSG00000001227
# 5  Aebp2 ENSMUSG00000030232 ENSMUSG00000001305
# 6  Aebp2 ENSMUSG00000030232 ENSMUSG00000001794

multi_simulation <- mosim(omics = list(
    "RNA-seq" = rnaseq_customdata,
    "DNase-seq" = dnaseseq_customdata),
    # The option is passed directly to mosim function instead of
```

```
    # being an element inside "omics" parameter.
    TFtoGene = custom_tf
    )
```

For methylation, a seed count sample does not need to be provided because it will be generated automatically. Methylation simulation just needs the association list containing the CpG sites to be simulated and the associated genes. The chromosomal positions for the CpG sites must be given in the format `<chr>_<start>_<end>`, that is, the chromosome number, start and end positions separated by the char `_`.

```
# Select a subset of the available data as a custom dataset
data("sampleData")

custom_cpgs <- head(sampleData$SimMethylseq$idToGene, 100)

# The ID column will be splitted using the "_" char
# assuming <chr>_<start>_<end>.
#
# These positions will be considered as CpG sites
# and used to generate CpG islands and other elements.
#
# Please refer to MOSim paper for more information.
#
#                    ID               Gene
# 1  11_3101154_3101154 ENSMUSG00000082286
# 2  11_3101170_3101170 ENSMUSG00000082286
# 3  11_3101229_3101229 ENSMUSG00000082286
# 4  11_3101287_3101287 ENSMUSG00000082286
# 5  11_3101329_3101329 ENSMUSG00000082286
# 6  11_3101404_3101404 ENSMUSG00000082286
```

3.2 Changing omic settings: `omicSim`

As commented before when describing `mosim` function, there are two ways of passing omic configuration options: by giving a list in the `omics` parameter or by giving `omics` as a character vector with the omics to simulate and specifying the simulation options in the parameter `omicsOptions`.

The `omicsOptions` parameter accepts a list, but the *MOSim* helper function, `omicSim`, allows to do it in a more straightforward way.

```
omicSim(omics, depth = NULL, totalFeatures = NULL, regulatorEffect = NULL)
```

The description of the parameters was explained in section 3.

Back to the first basic example of RNA-seq simulation using the default dataset, the code to use if we wish to restrict the number of features is:

```
omic_list <- c("RNA-seq")

rnaseq_options <- omicSim("RNA-seq", totalFeatures = 2500)

# The return value is an associative list compatible with
```

```
# 'omicsOptions'
rnaseq_simulation <- mosim(omics = omic_list,
                           omicsOptions = rnaseq_options)
```

When having multiple omics, we concatenate the information like follows:

```
omics_list <- c("RNA-seq", "DNase-seq")

# In R concatenaning two lists creates another one merging
# its elements, we use that for 'omicsOptions' parameter.
omics_options <- c(omicSim("RNA-seq", totalFeatures = 2500),
                   omicSim("DNase-seq",
                           # Limit the number of features to simulate
                           totalFeatures = 1500,
                           # Modify the percentage of regulators with effects.
                           regulatorEffect = list(
                                'activator' = 0.68,
                                'repressor' = 0.3,
                                'NE' = 0.02
                            )))

set.seed(12345)

multi_simulation <- mosim(omics = omics_list,
                          omicsOptions = omics_options)
```

The objects generated by `omicData` and `omicSim` are different and special attention must be paid to combine them. The following code shows an example on how to do it:

```
rnaseq_customdata <- omicData("RNA-seq", data = custom_rnaseq)
rnaseq_options <- omicSim("RNA-seq", totalFeatures = 100)

rnaseq_simulation <- mosim(omics = list("RNA-seq" = rnaseq_customdata),
                           omicsOptions = rnaseq_options)
```

4 Working with simulation results

The information contained in a simulation object can be classified in two categories: the simulation settings used to perform the simulation, and the count data matrices generated by the process.

To access this information, *MOSim* provides two helper functions: `omicSettings` to retrieve the simulation settings and `omicResults` for accessing the count matrices.

4.1 Retrieving the simulation settings: `omicSettings`

The helper function `omicSettings` is used to extract the settings used in the simulation:

```
omicSettings(simulation, omics = NULL, association = FALSE, reverse = FALSE, only.linked
= FALSE, include.lagged = TRUE)
```

The following parameters are accepted by the function:

simulation Simulation object returned by `mosim` function.

omics List with the names of the omic data types whose settings are to be retrieved.

association A logical value. If TRUE, the original association lists used in the simulation are included.

reverse A logical value. If TRUE, it swaps the column order in the association list in case we want to use the output directly and the program requires a different ordering.

only.linked A logical value. If TRUE, it returns only the regulator-gene interactions with effect.

include.lagged A logical value. If TRUE it will return the full settings table including regulator-gene interactions in which the minimum/maximum value for transitory profiles does not perfectly match, otherwise they will be filtered.

Users can choose to recover the setting for all the simulated omics or just for some of them:

```
# This will be a data frame with RNA-seq settings (DE flag, profiles)
rnaseq_settings <- omicSettings(multi_simulation, "RNA-seq")

# This will be a list containing all the simulated omics (RNA-seq
# and DNase-seq in this case)
all_settings <- omicSettings(multi_simulation)
```

For RNA-seq, the settings table has an structure similar to this:

ID	DE	Group1	Group2	Tmax.Group1	Tmax.Group2
ENSMUSG00000017204	TRUE	transitory.induction	continuous.repression	2.57	NA
ENSMUSG00000097082	TRUE	transitory.induction	transitory.induction	1.46	2.23
ENSMUSG00000055493	TRUE	transitory.induction	continuous.repression	2.37	NA
ENSMUSG00000017221	TRUE	transitory.induction	continuous.induction	2.63	NA
ENSMUSG00000020205	TRUE	transitory.induction	continuous.induction	2.83	NA
ENSMUSG00000087802	FALSE	flat	flat	NA	NA

Each column provides different information about the settings used to carry on the simulation:

ID Gene identifier.

DEG A logical value indicating if the gene was selected as differentially expressed (TRUE) or not (FALSE).

GroupX There will be as many group columns as groups defined in the experimental design, each one containing the type of expression profile assigned to the gene in the simulation.

Tmax.GroupX For transitory profiles, the time point with the absolute maximum (or minimum) value.

For regulatory omics, the structure will slightly differ from RNA-seq, adding additional columns. Note that in this example, for reading purposes, the last 4 columns containing the `Tmax.GroupX` and `Lagged.GroupX` have been omitted:

ID	Gene	Effect.Group1	Effect.Group2	Group1	Group2	...
10_111588324_111588448	ENSMUSG00000097082	activator	activator	transitory.induction	transitory.induction	...
10_111588324_111588448	ENSMUSG00000020205	activator	NA	transitory.induction	transitory.induction	...
10_11358301_11358431	ENSMUSG00000055493	activator	activator	transitory.induction	continuous.repression	...
10_11358301_11358431	ENSMUSG00000087802	NA	NA	transitory.induction	continuous.repression	...
11_98682094_98682786	ENSMUSG00000017204	repressor	activator	transitory.repression	continuous.repression	...
11_98682094_98682786	ENSMUSG00000017221	repressor	repressor	transitory.repression	continuous.repression	...

Each row describes a regulator-gene interaction, with the following columns:

ID : Regulator identifier.

Gene : Gene identifier.

Effect.GroupX : There will be as many effect group columns as groups in the experimental design. Each one will contain the effect of the regulator on the gene. As explained in previous sections, if both gene (being a DEG) and regulator share the same profile, the regulator is considered to act as activator; if they have completely opposite profiles, the regulator will be a repressor; for any other case, `NA` value will be set.

GroupX : There will be as many group columns as groups defined in the experimental design, each one containing the type of expression profile assigned to the regulator in the simulation, as described in section 1.

Tmax.GroupX : For transitory profiles, the time point with the absolute maximum (or minimum) value.

Lagged.GroupX : For transitory profiles, a logical value indicating if regulator and gene share the same maximum (or minimum) time point or not. The difference between points could explain potentially low correlation values between the two.

To retrieve the original association lists, the parameter `association` must be set to $TRUE$:

```
# This will be a list with 3 keys: settings, association and regulators
dnase_settings <- omicSettings(multi_simulation, "DNase-seq", association = TRUE)
```

When settings the `association` parameter to $TRUE$, the output object will be a list of lists with the following key names:

association List containing the association table for each omic.

settings List containing the setting data frames for each omic.

regulators Data frame combining the settings from all regulatory omics, adding an additional column $Omic$.

4.2 Accessing the count data matrices: `omicResults`

The last helper function is `omicResults`:

```
omicResults(simulation, omics = NULL)
```

This function can accept 2 parameters: the simulation output object and, optionally, the omics we want to retrieve. As in the previous helper function, retrieving one omic will result in a data frame object, and for more than one a list of data frames will be provided.

```
# multi_simulation is an object returned by mosim function.

# This will be a data frame with RNA-seq counts
rnaseq_simulated <- omicResults(multi_simulation, "RNA-seq")

#                    Group1.Time0.Rep1 Group1.Time0.Rep2 Group1.Time0.Rep3 ...
# ENSMUSG00000073155              4539              5374              5808 ...
# ENSMUSG00000026251                 0                 0                 0 ...
# ENSMUSG00000040472              2742              2714              2912 ...
# ENSMUSG00000021598              5256              4640              5130 ...
```

```
# ENSMUSG00000032348                421                 348                 492 ...
# ENSMUSG00000097226                 16                  14                   9 ...
# ENSMUSG00000027857                  0                   0                   0 ...
# ENSMUSG00000032081                  1                   0                   0 ...
# ENSMUSG00000097164                794                 822                 965 ...
# ENSMUSG00000097871                  0                   0                   0 ...

# This will be a list containing RNA-seq and DNase-seq counts
all_simulated <- omicResults(multi_simulation)
```

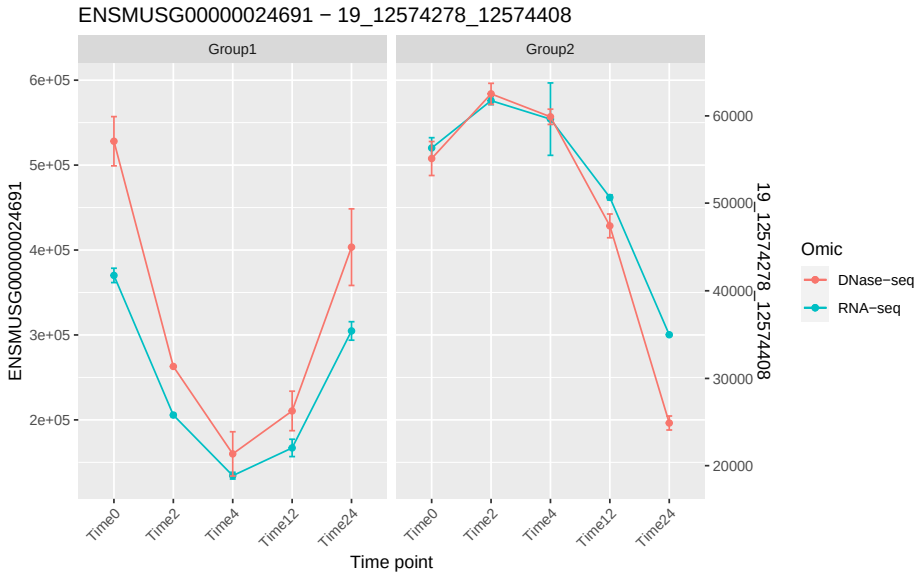
The structure of the final count matrix will have the features as row names, and the conditions as column names following the scheme `<Group>.<Timepoint>.<Replicate>`.

Alternatively a `ExpressionSet` object can be returned by setting the `format` argument to `"ExpressionSet"`.

4.3 Plotting results: `plotProfile`

Graphical plots are useful to check a feature profile or to compare gene & regulator interactions. To generate them, the function `plotProfile` needs the simulation object, the omic or two omics to plot, and one feature for each omic.

```
# The methods returns a ggplot plot, if called directly
# it will be automatically plotted.
plotProfile(multi_simulation,
            omics = c("RNA-seq", "DNase-seq"),
            featureIDS = list(
                "RNA-seq" = "ENSMUSG00000024691",
                "DNase-seq" = "19_12574278_12574408"
            ))
```

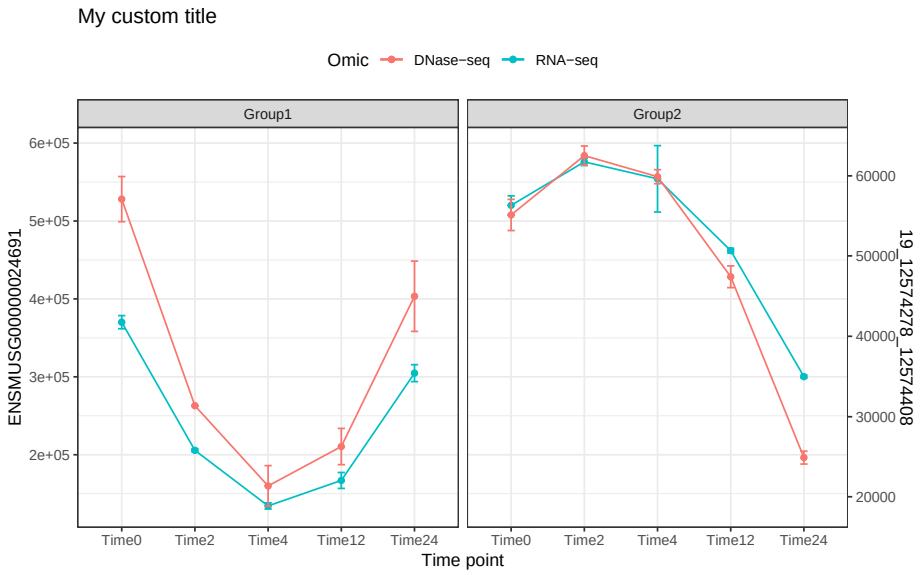


The returned ggplot can be stored in a variable to customize other attributes.

```
library(ggplot2)

# Store the plot in a variable
profile_plot <- plotProfile(multi_simulation,
            omics = c("RNA-seq", "DNase-seq"),
            featureIDS = list(
                "RNA-seq" = "ENSMUSG00000024691",
                "DNase-seq" = "19_12574278_12574408"
            ))

# Modify the title and print
profile_plot +
    ggtitle("My custom title") +
    theme_bw() +
    theme(legend.position="top")
```



5 Advanced use cases

Most of the common users' needs should be covered by the previously showed examples, but this section describes some advanced settings as the variability of the negative binomial distribution that is used to generate replicates.

5.1 Negative binomial variance

In a negative binomial distribution, the variance depends on the mean. In *MOSim*, the generated counts for a given condition (and/or time point) are taken as the mean of the distribution. To model the dependence between the negative binomial mean and variance for each omic, we analyzed several data sets and experiments, and observed a linear relationship between log-transformed count values of means and variances ($R^2 > 0.95$ for all models): $\sigma^2 = 10^a * (\mu + 1)^b - 1$. To assure a minimum variance we really used the maximum of this

value and 0.03. A regression model was applied to estimate coefficients a and b and these estimations were used as default values. The default values for a and b can be changed by users to increase or decrease the default variability in each omic data type.

6 Setup

This vignette was built on:

- R version 4.1.0 beta (2021-05-03 r80259), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 20.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MOSim 1.7.0, ggplot2 3.3.3
- Loaded via a namespace (and not attached): BiocGenerics 0.39.0, BiocManager 1.30.15, BiocStyle 2.21.0, DBI 1.1.1, HiddenMarkov 1.8-13, IRanges 2.27.0, R6 2.5.0, S4Vectors 0.31.0, assertthat 0.2.1, codetools 0.2-18, colorspace 2.0-1, compiler 4.1.0, crayon 1.4.1, digest 0.6.27, dplyr 1.0.6, ellipsis 0.3.2, evaluate 0.14, fansi 0.4.2, farver 2.1.0, formatR 1.9, generics 0.1.0, glue 1.4.2, grid 4.1.0, gtable 0.3.0, highr 0.9, htmltools 0.5.1.1, knitr 1.33, labeling 0.4.2, lattice 0.20-44, lazyeval 0.2.2, lifecycle 1.0.0, magrittr 2.0.1, matrixStats 0.58.0, munsell 0.5.0, parallel 4.1.0, pillar 1.6.1, pkgconfig 2.0.3, purrr 0.3.4, rlang 0.4.11, rmarkdown 2.8, scales 1.1.1, stats4 4.1.0, stringi 1.6.2, stringr 1.4.0, tibble 3.1.2, tidyr 1.1.3, tidyselect 1.1.1, tools 4.1.0, utf8 1.2.1, vctrs 0.3.8, withr 2.4.2, xfun 0.23, yaml 2.2.1, zoo 1.8-9