SUMO: Simulation Utilities for Multi-Omics Data (SUMO.v1.2.3)

Bernard Isekah Osang'ir, Surya Gupta, Ziv Shkedy, and Jürgen Claesen (2025)

1 Introduction

The SUMO R package provides a powerful and flexible simulation framework tailored for the generation of synthetic multi-omics datasets. These datasets are vital for evaluating and benchmarking integrative bioinformatics tools, specifically for factor analysis based methods, however, it can be used for other methods such as clustering, and dimensionality reduction methods. SUMO allows users to simulate multiple omics datasets with predefined signal structures, making it ideal for method development, and reproducible benchmarking.

This vignette accompanies the SUMO R package and serves as both a tutorial and reference for researchers and developers interested in synthetic data for multi-omics.

1.1 Applications of SUMO

- Benchmarking multi-omics integration methods (e.g., MOFA2, FABIA)
- Testing robustness of statistical and machine learning models under controlled noise conditions
- Teaching and demonstration of signal vs. noise detection in high-dimensional datasets

1.2 Key Features

- Simulation of two or more omics datasets with latent factor structures
- Flexible configuration of signal-to-noise ratio, latent factor count, and distribution type
- Intuitive visualization tools (2D, 3D, loadings, scores)
- MOFA2-based end-to-end analysis pipeline with exportable reports
- Modular utility functions to support simulation design

1.3 Installation

You can install SUMO directly from CRAN using the following command:

▼ Hide

```
install.packages("SUMO")
```

1.4 Need help

You can get help in SUMO directly by using the following command:

▼ Hide

```
help("SUMO") # or

?SUMO
```

You will receive a quick description of SUMO, key features, main functions, and the responsible contact person who is the author and the maintainer. SUMO main functions include:

1.4.1 SUMO functions

Below is the current set of SUMO's primary functions with brief roles.

Main functions:

- `simulateMultiOmics()` — simulate ≥2 omics with predefined latent factors, non-overlapping sample/feature blocks, and optional real-data noise statistics. :contentReferenceoaicite:0
- `simulate_twoOmicsData()` — simulate exactly two omics with configurable factor structures (shared/unique/mixed), SNR, and signal distributions. :contentReferenceoaicite:1
- `as_multiomics()` — convert legacy outputs to the standardized multi-omics schema (`omics`, `list_alphas`, `list_betas`, `signal_annotation`, `factor_map`). :contentReferenceoaicite:2
- `plot_simData()` — quick heatmap view of merged or per-omic matrices (with optional permutations for sanity checks). :contentReferenceoaicite:3
- `plot_factor()` — visualize ground-truth sample factor scores (scatter or histogram). :contentReferenceoaicite:4
- `plot_weights()` — visualize feature loadings per omic/factor (scatter or histogram). :contentReferenceoaicite:5
- `demo_multiomics_analysis()` — end-to-end MOFA2 workflow on SUMO or CLL data; prefers MOFA2's basilisk backend, can fall back to a user `reticulate` env, supports **pretrained models**, and can export a multi-slide **PowerPoint** report. :contentReferenceoaicite:6
- `compute_means_vars()` — compute overall, row-wise, and column-wise means/SDs for each dataset; useful for real-data–aware noise modeling. :contentReferenceoaicite:7

MOFA utilities (python backend):

- `sumo_setup_mofa()` — interactive fallback to create/use a `reticulate`/conda env with `mofapy2` when basilisk isn't available. :contentReferenceoaicite:8
- `sumo_load_pretrained_mofa()` — load a bundled pretrained MOFA model (no Python needed). :contentReferenceoaicite:9
- `sumo_pretrained_mofa_available()` — list names of included pretrained models. :contentReferenceoaicite:10
- `sumo_pretrained_mofa_path()` — get filesystem path to a bundled pretrained model. :contentReferenceoaicite:11

Supporting helpers (for simulation/block generation):

- `divide_samples()`, `divide_vector()` — split/allocate sample indices for factor blocks. :contentReferenceoaicite:12
- `divide_features_one()`, `divide_features_two()` — allocate feature indices per factor for omic 1/2. :contentReferenceoaicite:13
- `feature_selection_one()`, `feature_selection_two()` — select signal-carrying feature blocks for omic 1/2. :contentReferenceoaicite:14

1.5 Dependencies

You can get help in SUMO directly by using the following command:

2 Simulating Multi-Omics Datasets

SUMO originally introduced the `simulate_twoOmicsData()` function to support synthetic data generation for **two omics layers**, such as transcriptomics and proteomics. This function provided control over:

- The number of features per omic
- Shared or unique latent factor structures
- Signal-to-noise ratios (SNR)
- Customized signal distributions across samples and features

This function creates datasets with user-defined dimensions, signal regions, and latent factor relationships.

However, modern integrative analysis often involves two omics and/or more datasets. To address this, SUMO introduces an extended and general-purpose function: `simulateMultiOmics()`.

2.1 Core Function for ≥ 2 Omics (`simulateMultiOmics()`)

The function `simulateMultiOmics()` generalizes and replaces the two-layer simulator, allowing users to define **two or more omics layers** with distinct or shared signal structures. It is now the recommended core simulator for all multi-omics benchmarking and demo applications.

2.1.1 Key Advantages Over `simulate_twoOmicsData()`

- **Flexible omic count**: Simulate 2, 3, or more omics with a single call.
- **Custom latent factor design**: Define how each factor is distributed—shared by all omics, unique to some omics, or mixed (shares and unique factors) across omics.
- **Independent signal control**: Specify feature- and sample-level signal characteristics per omic.
- **Modular outputs**: Returned object is structured for downstream use in factor analysis based pipeline such as MOFA pipeline.

2.1.2 Example

▼ Hide

```
library(SUMO)

sim_object1 <- simulateMultiOmics(
  vector_features = c(3000, 2500, 2000),  # Features in omic1, omic2, omic3
  n_samples = 100,                        # Shared samples
  n_factors = 3,                          # Number of latent factors
  snr = 3,                                # Signal-to-noise ratio
  signal.samples = c(5, 1),
  signal.features = list(c(3, 0.3), c(2.5, 0.25), c(2, 0.2)),
  factor_structure = "mixed",            # Shared + specific factors
  num.factor = "multiple",
  seed = 123
)
```

2.1.3 Input Parameters

The table below explains each parameter used in the `simulateMultiOmics()` example:

Parameter	Meaning
`vector_features = c(3000, 2500, 2000)`	Specifies the number of features in each omics layer. Omic 1 has 3000, Omic 2 has 2500, and Omic 3 has 2000 features.
`n_samples = 100`	Number of shared samples across all omics. These samples will be simulated identically across the 3 layers.
`n_factors = 3`	The number of latent factors (biological processes, patterns, or signals) to simulate.
`snr = 3`	Signal-to-noise ratio. A higher value means simulated signal is more distinguishable from noise.
`signal.samples = c(5, 1)`	Mean and standard deviation of the signal strength **across samples**.
`signal.features = list(c(3, 0.3), c(2.5, 0.25), c(2, 0.2))`	Mean and standard deviation of signal **across features**, defined separately for each omic.
`factor_structure = "mixed"`	Specifies that some latent factors are **shared** across all omics, while others are **omic-specific**.
`num.factor = "multiple"`	Indicates that multiple factors are being simulated (not just one global factor).
`seed = 123`	Ensures reproducibility. The same random seed yields the same output when rerun.

2.1.4 Output Object

▼ Hide

```
names(sim_object1)
```

- `omic.list`: A list of 3 numeric matrices, each representing a simulated omics dataset with dimensions (features × samples). These are the core data matrices used for downstream analysis or benchmarking.
- `signal_annotation`: A structured list containing information about where signal was injected in both samples and features. This is essential for validating method accuracy (e.g., sensitivity, specificity).
- `list_alphas`: A matrix of sample-level latent factor scores with dimensions (samples × latent factors). Each row corresponds to a sample and each column to a latent factor.
- `list_betas`: A list of feature-level loading matrices for each omic. Each matrix has dimensions (features × latent factors) and captures how strongly each feature loads onto each latent factor, specific to its omics layer.

2.2 Simulating Two Omics Datasets

Use either `simulate_twoOmicsData()` or `simulateMultiOmics()` for two-layer simulation. This allows more fine-tuned control over individual omic signal distributions and latent factor type (shared, unique, mixed). The example below demonstrated using the former function to simulate two-layer dataset. When using `simulate_twoOmicsData()` always convert the output to the current standards as for `simulateMultiOmics()` so that other functionality of SUMO can be used without running into modelling issues.

▼ Hide

```
set.seed(123)
sim_object2 <- simulate_twoOmicsData(
  vector_features = c(4000, 3000),
  n_samples = 100,
  n_factors = 2,
  snr = 2.5,
  num.factor = "multiple",
  advanced_dist = "mixed"
)

sim_std <- as_multiomics(sim_object2)  # ← now looks like simulateMultiOmics()
```

3 Simulation Parameter Setup

The `simulateMultiOmics()` function in SUMO supports three flexible approaches for setting up simulation parameters, depending on the user's goals and level of expertise. This is designed to ensure that both beginners and advanced users can effectively generate high-quality, synthetic data.

3.1 Parameter Configuration Options

There are three main ways to configure simulation parameters:

- **Default parameters**: Suitable for first-time users or exploratory testing. Provides balanced settings for typical benchmarking scenarios.
- **User-defined parameters**: Allows full customization of the simulation settings, giving users precise control over sample size, signal structure, and omics complexity.
- **Real-world informed parameters**: Enables biologically realistic simulations by extracting feature-level statistics (means and variances) from real datasets using the `compute_means_vars()` function and use them in the simulation.

3.2 Parameter Overview

Below is a detailed description of each parameter accepted by `simulateMultiOmics()`:

Parameter	Required	Default	Description
`vector_features`	✅	—	Specifies the number of features in each omics layer. E.g., `c(3000, 2500, 2000)` simulates 3 omics with 3000, 2500, and 2000 features.
`n_samples`	✅	—	Number of shared samples across all omics. These samples will be simulated identically across layers.
`n_factors`	✅	—	Number of latent factors (biological patterns or signals) to simulate.
`snr`	❌	`2`	Signal-to-noise ratio. Higher values simulate clearer signal. E.g., `snr = 3` makes signal more distinct from background noise.
`signal.samples`	❌	`c(5, 1)`	Mean and standard deviation of signal intensity across samples for each factor.
`signal.features`	❌	`NULL`	A list like `list(c(3, 0.3), c(2.5, 0.25), c(2, 0.2))` defines signal strength (mean, SD) across features per omic.
`factor_structure`	❌	`"mixed"`	Specifies whether factors are shared, unique to each omic, or a mix of both (`"shared"`, `"unique"`, or `"mixed"`).
`num.factor`	❌	`"multiple"`	Indicates how many factors each omic receives: `"single"` for one factor per omic, `"multiple"` for several.
`seed`	❌	`NULL`	Optional seed value for reproducibility. Using the same seed ensures the same output when rerun.

3.3 Example Usage

Here's an example of using default and user-defined parameters:

▼ Hide

```
# Minimal example using default parameters
simulateMultiOmics(
  vector_features = c(3000, 2500, 2000),
  n_samples = 100,
  n_factors = 3
)

# Fully customized example - user-defined
simulateMultiOmics(
  vector_features = c(3000, 2500, 2000),
  n_samples = 100,
  n_factors = 3,
  snr = 3,
  signal.samples = c(10, 2),
  signal.features = list(c(3, 0.3), c(2.5, 0.25), c(2, 0.2)),
  factor_structure = "mixed",
  num.factor = "multiple",
  seed = 123
)
```

3.4 Real-World Informed Simulation

To simulate omics layers with real-world-like distributions, users can leverage the `compute_means_vars()` function. This function calculates the overall, row-wise, and column-wise means and standard deviations of each dataset. These statistics can then guide signal generation in `simulateMultiOmics()`.

This approach is especially useful when benchmarking methods using simulated data that closely mimic real omics datasets (e.g., gene expression and methylation data).

Below is an example using two simple simulated matrices to demonstrate how users can extract empirical means and standard deviations and plug them into `simulateMultiOmics()`:

3.4.1 Step 1: Compute feature-level and sample-level statistics

▼ Hide

```
# Assuming `real_data1`, `real_data2` are matrices of real omics data. Let's simulate
# two real-world-like omics datasets

set.seed(123)
real_data1 <- matrix(rnorm(100, mean = 5, sd = 2), nrow = 10, ncol = 10)  # Omic 1
real_data2 <- matrix(rnorm(100, mean = 10, sd = 3), nrow = 10, ncol = 10) # Omic 2

data_list <- list(real_data1, real_data2)

# Compute summary statistics using SUMO's built-in function

real_stats <- compute_means_vars(data_list)
print(real_stats)

# Use output from `real_stats` to inform custom simulation settings
# (future versions may allow direct plugging into simulateMultiOmics)
```

This outputs values such as:

▼ Hide

```
$overall_mean.one: 5.18        $overall_sd.one: 1.83
$overall_mean.two: 9.68        $overall_sd.two: 2.90
$mean_smp: 7.43                $sd_smp: 2.32
```

3.4.2 Step 2: Use the statistics in `simulateMultiOmics()`

You can now plug these values directly into the `signal.features` and `signal.samples` parameters to simulate data with similar distributions:

▼ Hide

```
# Simulate using real-data-informed parameters
sim_data_real <- simulateMultiOmics(
  vector_features = c(10000, 15000),
  n_samples = 150,
  n_factors = 2,
```

```
  snr = 0.5,
  signal.samples = c(real_stats$mean_smp, real_stats$sd_smp),  # Use sample-level mean/sd
  signal.features = list(
    c(real_stats$overall_mean.one, real_stats$overall_sd.one),  # Omic 1
    c(real_stats$overall_mean.two, real_stats$overall_sd.two)   # Omic 2
  ),
  factor_structure = "mixed",
  num.factor = "multiple",
  seed = 42
)
```

This real-data-informed simulation mode bridges the gap between fully synthetic and real datasets, offering realism while retaining control over ground-truth signal structures.

You can view the resulting structure with:

▼ Hide

```
str(sim_data_real)
```

4 Visualization Tools

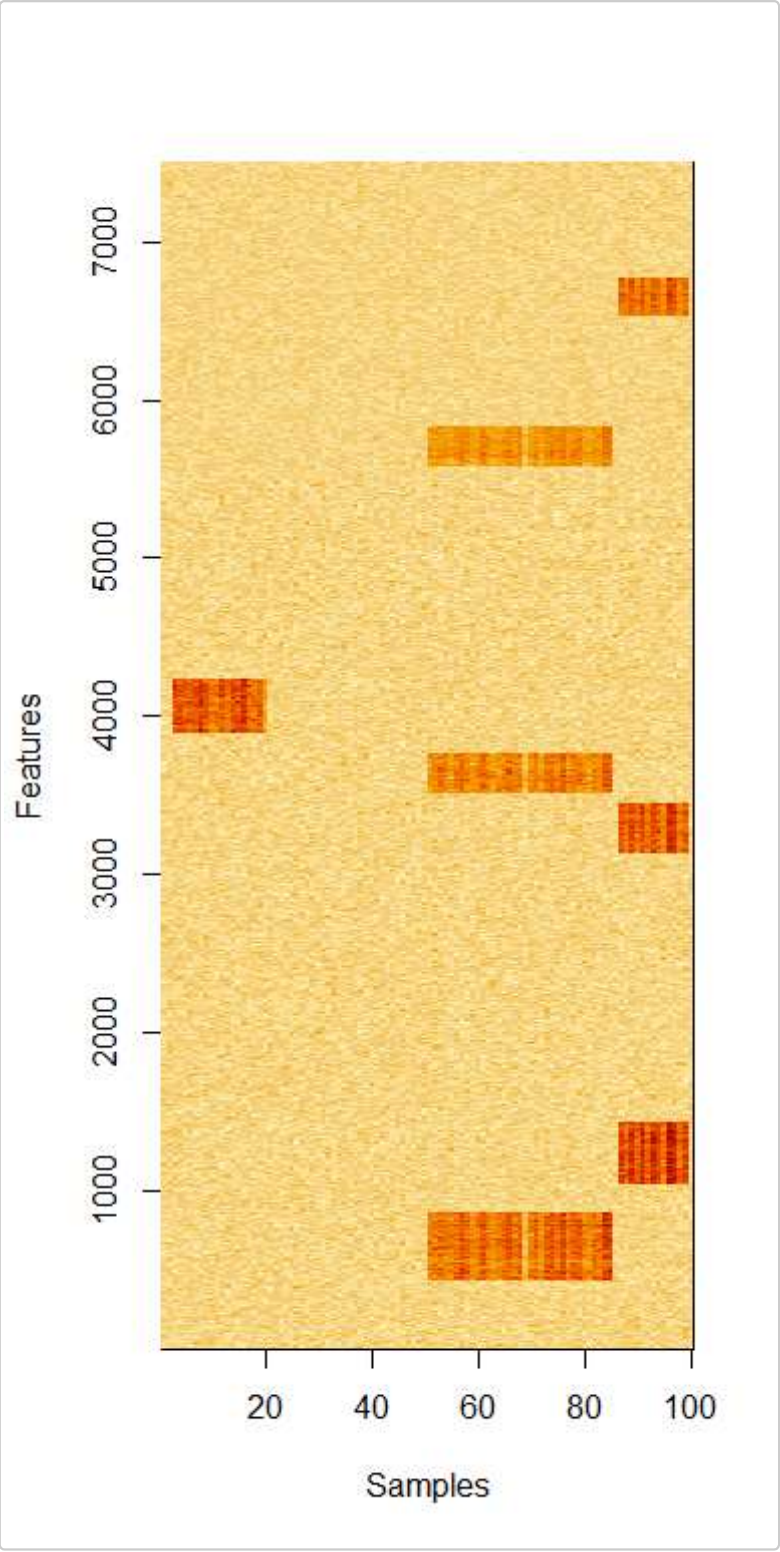
SUMO provides a variety of built-in visualization tools to help users explore simulated multi-omics data. These include heatmaps, 3D surface plots, factor score scatterplots, and feature loading visualizations. All functions are compatible with both merged and individual omics layers.

4.1 Heatmaps and Surface Plots

You can visualize either the merged omics matrix or individual omics layers using 2D heatmaps or 3D surface plots. These are useful for inspecting co-expression patterns, block structure, and noise.

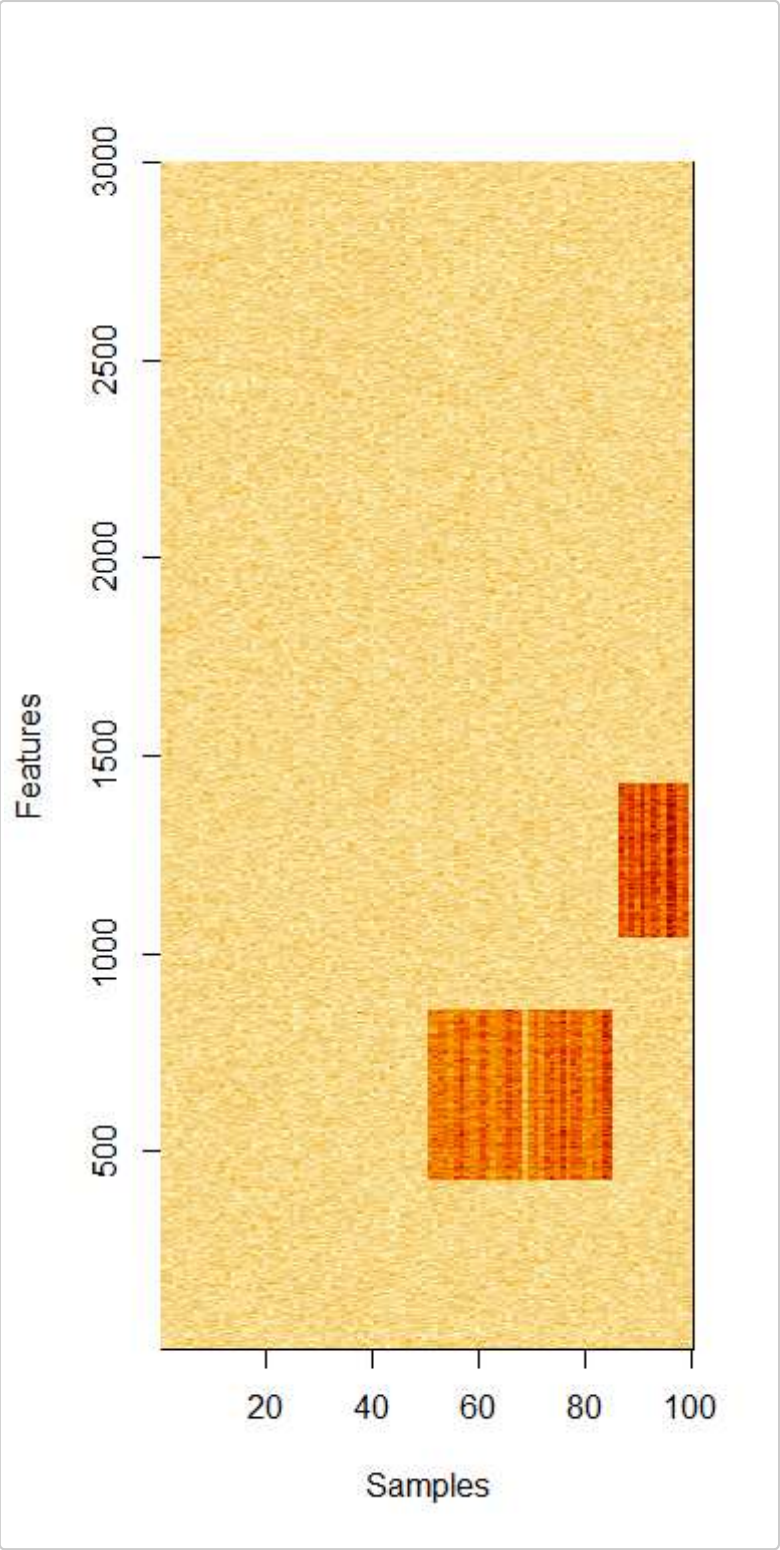
▼ Hide

```
# combined data
plot_simData(sim_object1, data = "merged", type = "heatmap")
```



▼ Hide

```
# # Heatmap of individual omic layer 2: omic1_tbl
plot_simData(sim_object1, data = "omic1", type = "heatmap")
```



▼ Hide

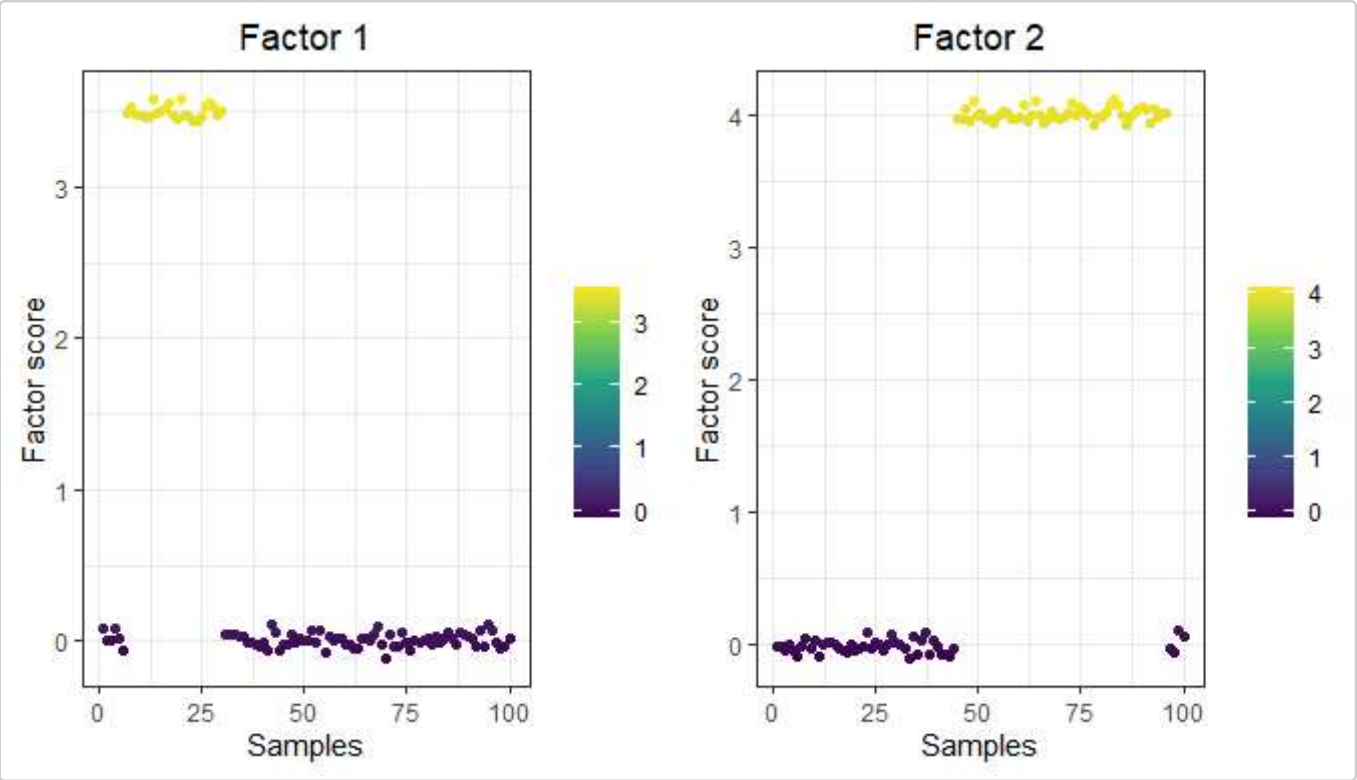
```
# Optional: 3D surface plot of omic 2: omic1_tbl
#plot_simData(sim_object1, data = "merged", type = "3D")
```

4.2 Factor Score Visualization

The `plot_factor()` function shows factor scores across samples. When `factor_num = "all"` is used, multiple scatterplots are displayed, one per factor. You can also specify a specific factor to isolate its effect.

▼ Hide

```
plot_factor(sim_std, factor_num = "all", type = "scatter")
```

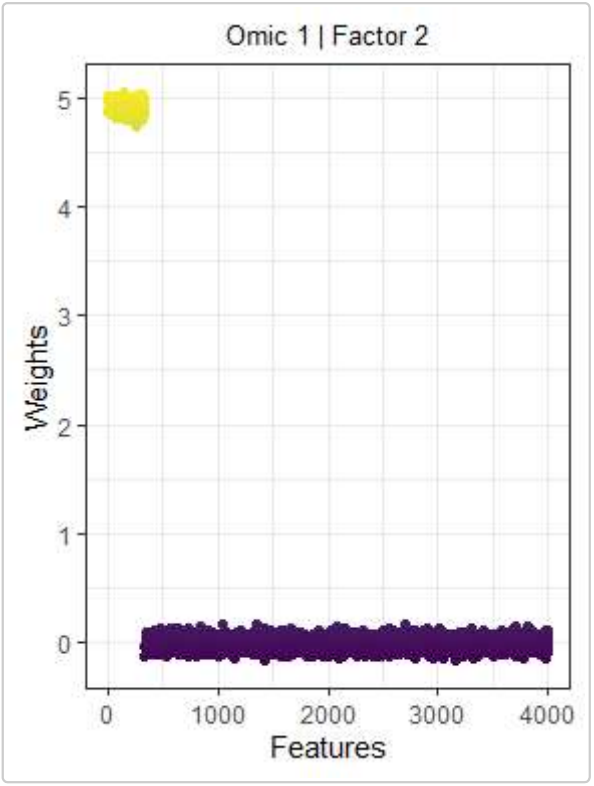


4.3 Feature Loading (Weight) Plots

The `plot_weights()` function visualizes how strongly each feature loads onto a selected latent factor. This helps identify signal-driving features or clusters.

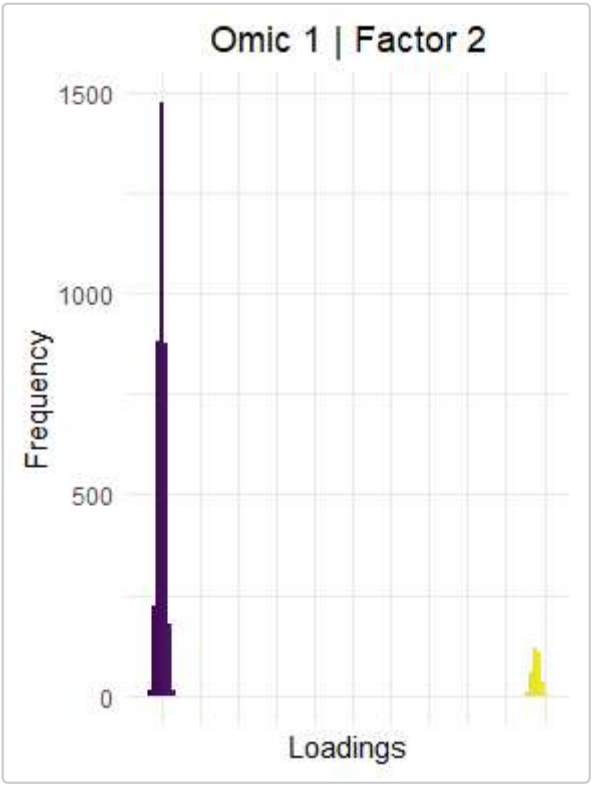
▼ Hide

```
plot_weights(sim_object = sim_std, factor_num = 2, omic = 1,
   type = 'scatter',
   show.legend = FALSE
   )
```



▼ Hide

```
plot_weights(sim_object = sim_std, factor_num = 2, omic = 1,
   type = 'histogram',
   show.legend = FALSE
   )
```



These plots help users verify the location and magnitude of signal across features and samples.

5 Full MOFA2 Analysis Pipeline

5.1 MOFA2 set-up

5.1.1 Installation

To successfully run `demo_multiomics_analysis()` you need to set-up and install `MOFA2`. This is a established multi-omics integration package build based of factor analysis. MOFA2 is build over `Bioconductor`. Follow the instructions below to install `MOFA2` from the `Bioconductor`.

▼ Hide

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

BiocManager::install(c("MOFA2", "MOFAdata"), ask = FALSE) # ask = FALSE avoids interactive prompts
         (useful for scripts/vignettes).
```

To view documentation for the version of this package installed in your system, start R and enter:

▼ Hide

```
browseVignettes("MOFA2")
```

`MOFA2` uses python dependencies and to install this we use `basilisk` package which freezes python dependencies inside `Bioconductor` packages. You will need to install `basilisk` package as below:

▼ Hide

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

BiocManager::install("basilisk")
```

To view documentation for the version of this package installed in your system, start R and enter:

▼ Hide

```
browseVignettes("basilisk")
```

Note that upon installation of MOFA2, you can usually run the demo without installing `basilisk`, because SUMO ships a python backend that already embeds `basilisk` and `reticulate` packages. This ensures execution of mofapy2 behinds the scenes (see details below).

5.1.2 Packages required for `demo_multiomics_analysis()` reporting

The following packages are necessary to run the `demo_multiomics_analyssis()` function.

▼ Hide

```
install.packages(c("flextable", "rvg"))
```

`flextable`: make nicely formatted tables for Word/PPT. `rvg`: export editable vector graphics (e.g., ggplots) into PowerPoint/Word via `officer`. ***Note***: officer is already listed in your package Imports, so users don't need to install it separately. ## Description of the function (`demo_multiomics_analysis()`) The function

`demo_multiomics_analysis()` provides a comprehensive demo using either `SUM0`-generated data (option `data_type = "SUMO"`) or real-world CLL data (option `data_type = "real_world"`). Here, we run a complete `MOFA2`-based analysis pipeline using these two data types.

This function includes preprocessing, MOFA model training, variance decomposition visualization, and optional PowerPoint report generation, when `export_pptx = TRUE`. Future development, will include .HTML,.docx, .PDF reports.

To avoid having issues with `mofapy2`, we have added a pretrained model that can be loaded if python_envr cannot be invoked. The option `use_pretrained` One of `"auto"`, `"always"`, `"never"`. `"auto"`: train if a backend is available, otherwise load a pretrained model. `"always"`: always load a pretrained model and skip training.`"never"`: always train (requires a working Python backend for `MOFA2`).

This means `SUMO` can seamlessly be integrated as a plugin to any factor analysis multi-omics analysis pipeline for testing and validate the methods.

▼ Hide

```
# 1. Demo - SUMO
demo_sumo <- demo_multiomics_analysis(
  data_type = "SUMO",
  export_pptx = FALSE,
  verbose = TRUE,
  use_pretrained = "auto")

demo_sumo <- demo_multiomics_analysis(
  data_type = "SUMO",
  export_pptx = FALSE,
  verbose = TRUE,
  use_pretrained = "always")

# Demo - SUMO
demo_real <- demo_multiomics_analysis(
  data_type = "real_world",
  export_pptx = FALSE,
  verbose = TRUE,
  use_pretrained = "never")
```

5.2 Example of Analysis Demo with SUMO-generated dataset

When saving the results to .pptx, this will be saved automatically to your working directory local computer.

▼ Hide

```
# 1. Demo - SUMO
demo_sumo <- demo_multiomics_analysis(
  data_type = "SUMO",
  export_pptx = FALSE,
  verbose = TRUE,
  use_pretrained = "never")

demo_sumo
```

5.3 Example of Analysis Demo with SUMO-generated dataset

When saving the results to .pptx, this will be saved automatically to your working directory local computer.

▼ Hide

```
# 1. Demo - SUMO
demo_real <- demo_multiomics_analysis(
  data_type = "real_world",
  export_pptx = FALSE,
  verbose = TRUE,
  use_pretrained = "auto")

demo_real
```

This includes model fitting, variance decomposition, and PowerPoint export.

6 Supporting Utility Functions

The following functions are essential for simulation logic and block generation:

- `divide_samples()` — partitions sample indices into contiguous blocks per latent factor (with min-size constraints); used to simulate non-overlapping sample-level signal. :contentReferenceoaicite:0
- `divide_vector()` — helper to randomly split `n_samples` into `num` segments (respecting `min_size`), used for sampling/bootstrapping of score blocks. :contentReferenceoaicite:1
- `feature_selection_one()` / `feature_selection_two()` — assign non-overlapping feature blocks carrying signal to omic 1 / omic 2 across factors. :contentReferenceoaicite:2
- `divide_features_one()` / `divide_features_two()` — low-level helpers to allocate per-factor feature indices in omic 1 / omic 2 (single vs multiple factor modes). :contentReferenceoaicite:3
- `compute_means_vars()` — computes per-omic means/SDs; enables realistic noise modeling when `real_stats = TRUE` in `simulateMultiOmics()`. :contentReferenceoaicite:4
- `as_multiomics()` — normalizes legacy simulation outputs to the current schema (`omics`, `list_alphas`, `list_betas`, `signal_annotation`, `factor_map`) so downstream utilities work consistently. :contentReferenceoaicite:5

7 Recommended Use Cases

- Researchers creating benchmark datasets for integrative omics methods
- Educators teaching signal detection, latent structure discovery, and dimensionality reduction
- Method developers validating robustness to noise and signal sparsity

8 Conclusion

SUMO is a foundational tool for synthetic multi-omics simulation. Its rigorously tested structure, customizability, and compatibility with modern pipelines make it a valuable asset to bioinformatics methodologists, educators, and practitioners.

For further information and updates, please refer to the accompanying publication in *Bioinformatics Advances* and the official GitHub repository.