Simulation for Microbiome Analysis

Kris Sankaran, Saritha Kodikara, and Kim-Anh Lê Cao

2024-06-26

Contents

Chapter 1

Introduction

This website accompanies the review Leveraging Simulation in Microbiome Data Analysis. All examples discussed in the case studies there can be reproduced by running the code below, and we have also included additional introductory material about some packages that simplify our analysis.

1.1 Setup

The `MIGsim` package below provides wrapper functions that we will use throughout these book. You can install it by running:

```
devtools::install_github("krisrs1128/intro-to-simulation/MIGsim")
```

This block loads all packages that will be used in this book.

```
suppressPackageStartupMessages({
  library(CovTools)
  library(SpiecEasi)
  library(SummarizedExperiment)
  library(dplyr)
  library(forcats)
  library(gamboostLSS)
  library(ggdist)
  library(ggplot2)
  library(glue)
  library(mixOmics)
  library(patchwork)
  library(scico)
  library(splines)
  library(tibble)
  library(tidyr)
  library(MIGsim)
  library(purrr)
  library(scDesigner)
})
theme_set(theme_classic())
```

1.2 Using `SummarizedExperiment`

`SummarizedExperiment` data structures simplify manipulation of sequencing experiments, and we'll be using them throughout these tutorials. For example, they distinguish between molecule counts, which are stored in the `assay` slot, and sample descriptors, which are stored in `colData`. At the same time, these separate components are nicely synchronized. For example, subsetting samples from one of these tables automatically subsets the other.

The line below loads a small subset of genera from the Atlas experiment, which profiled the gut microbiomes from 1006 healthy adults in Western Europe.

```
data(atlas)
table(rowData(atlas)$Phylum)
```

```
##
## Actinobacteria  Bacteroidetes     Firmicutes
##              1              2             21
```

```
mean(atlas$age)
```

```
## [1] 45.15629
```

```
ix <- colData(atlas)$bmi_group == "obese"
abundances <- assay(atlas)
rowMeans(abundances[, ix])
```

```
##                   Allistipes et rel.         Anaerostipes caccae et rel.          Ba
##                            289.85263                           123.84211
##       Butyrivibrio crossotus et rel.       Clostridium cellulosi et rel.
##                            188.54035                           437.13684
##       Clostridium sphenoides et rel.       Clostridium symbiosum et rel.          Co
##                            139.35439                           331.90175
##     Oscillospira guillermondii et rel. Outgrouping clostridium cluster XIVa         P
##                           1560.27368                            91.93684
##       Sporobacter termitidis et rel.     Subdoligranulum variable at rel.
##                            423.71579                           442.34737
```

Exercise: To practice working with `SummarizedExperiment` objects, try answering:

- How many genera are available in this experiment object?

- What was the most common phylum in this dataset?
- What was the average participant age?
- What was the average abundance of `Allistipes et rel.` among people in the `obese` BMI group?

Hint: The most important functions are `assay()`, `rowData()`, and `colData()`.

Solution

```
nrow(atlas)
```

```
## [1] 24
```

```
table(rowData(atlas)$Phylum)
```

```
##
## Actinobacteria  Bacteroidetes     Firmicutes
##              1              2             21
```

```
mean(atlas$age)
```

```
## [1] 45.15629
```

```
atlas[, atlas$bmi_group == "obese"] |>
  assay() |>
  rowMeans()
```

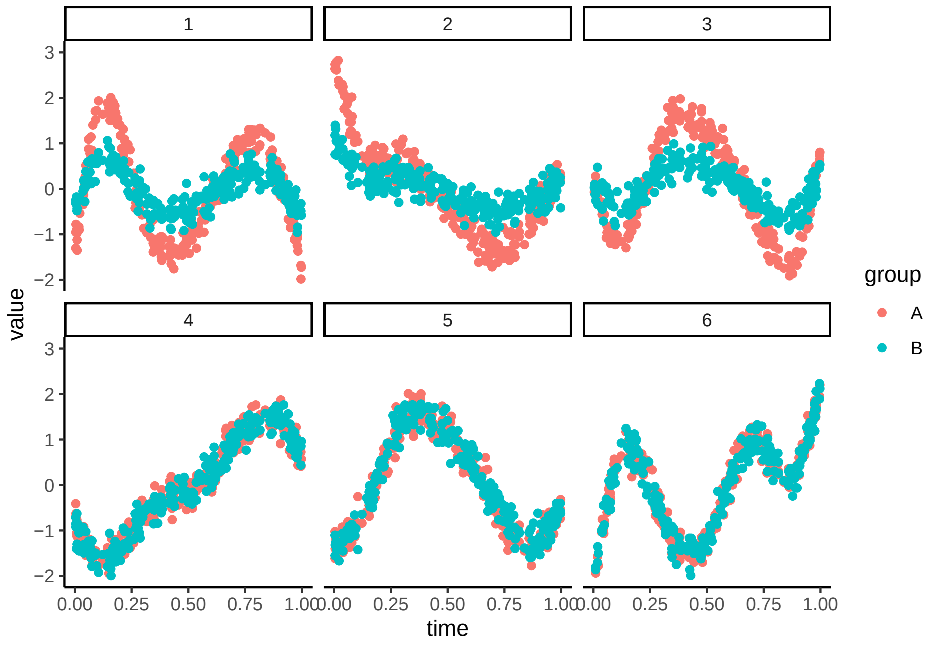
```
##                   Allistipes et rel.         Anaerostipes caccae et rel.         Bacteroides
##                            289.85263                           123.84211
##       Butyrivibrio crossotus et rel.        Clostridium cellulosi et rel.          Clostridiu
##                            188.54035                           437.13684
##       Clostridium sphenoides et rel.        Clostridium symbiosum et rel.         Coprococcus
##                            139.35439                           331.90175
##    Oscillospira guillermondii et rel. Outgrouping clostridium cluster XIVa          Ruminococcu
##                           1560.27368                            91.93684
##       Sporobacter termitidis et rel.     Subdoligranulum variable at rel.          Uncultured
##                            423.71579                           442.34737
```

1.3 Warm-up: A Gaussian Example

Here's a toy dataset to illustrate the main idea of GAMLSS. Each panel in the plot below represents a different feature (e.g., taxon, gene, metabolite…). The abundance varies smoothly over time, and in the first three panels, the trends differ by group assignment.

```
data(exper_ts)
exper_lineplot(exper_ts)
```

```
## Joining with `by = join_by(newdata_index)`
## Warning: Removed 12 rows containing missing values or values outside the scale rang
```



We can try to approximate these data with a new simulator. The `setup_simulator` command takes the template `SummarizedExperiment` object as its first argument. The second gives an R formula syntax-style specification of GAMLSS parameters (mean and SD, in this case) dependence on sample properties. The last argument gives the type of model to fit, in this case, a Gaussian location-shape-scale model.

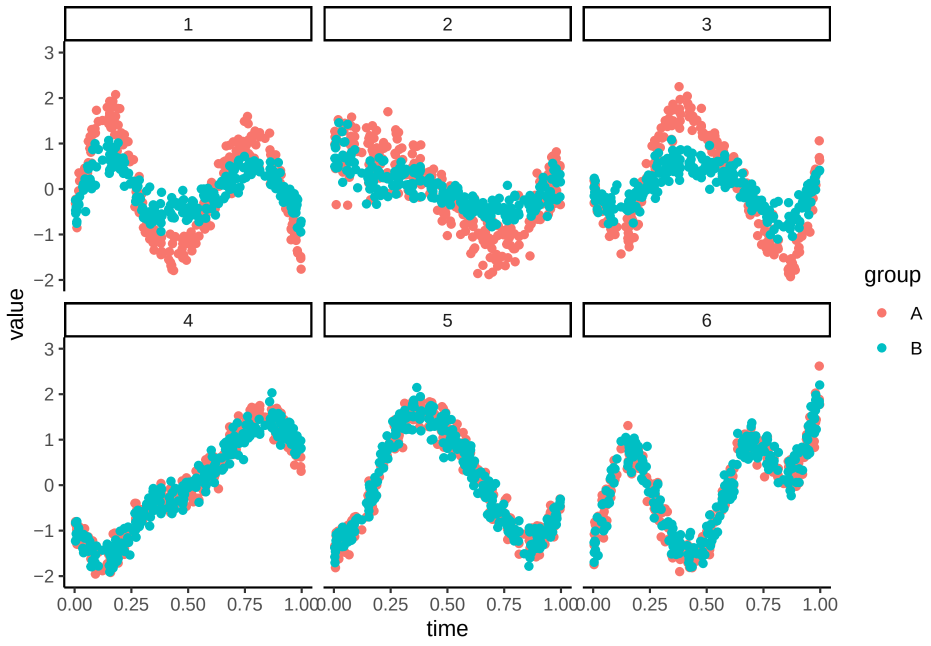
```r
sim <- setup_simulator(exper_ts, ~ ns(time, df = 7) * group, ~ GaussianLSS()) |>
  estimate(nu = 0.01, mstop = 1000)
```

```
## [=============================================================================
## 0s[============================================================================
## 0s[============================================================================
## 0s[============================================================================
```

```r
sample(sim) |>
  exper_lineplot()
```

```
## Joining with `by = join_by(newdata_index)`
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
```



Exercise: Right now, each panel allows for an interaction between the trend and group type. Can you define a simulator where the groups have no effect on the trends for the first two panels? This is the basis for defining synthetic negative controls.

```r
sim <- sim |>
  scDesigner::mutate(
    1:2,
     link = ~ ns(time, df = 7)
  ) |>
  estimate(nu = 0.01, mstop = 1000)

sample(sim) |>
  exper_lineplot()
```

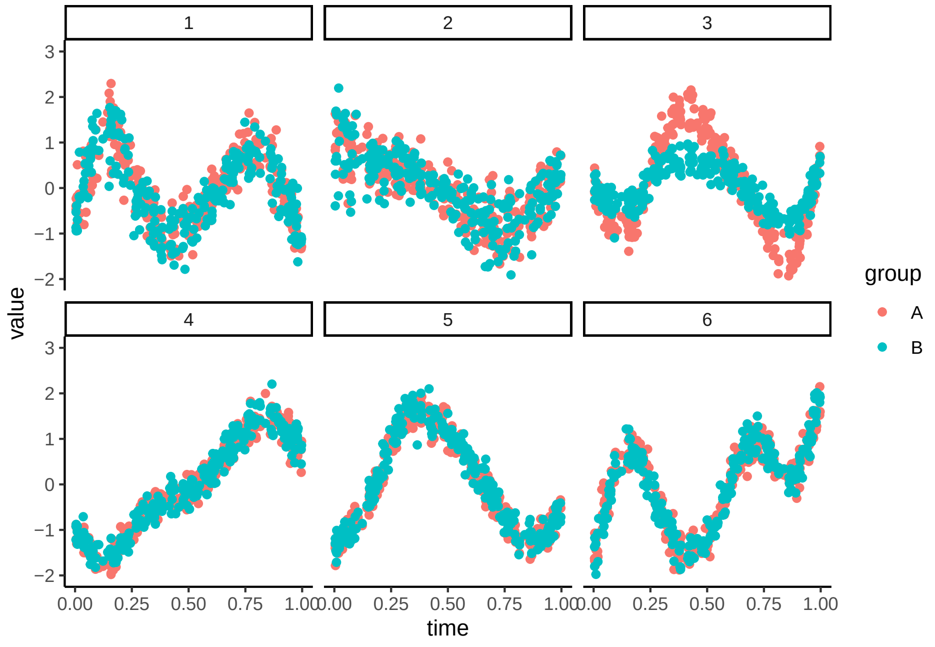
Solution: We can modify the formula so that it no longer has an interaction with group. We just need to remove the `* group` from the original formula in our updated link function. To ensure that this only applies to the first two panels, we use 1:2 in the first argument of `mutate`. This first argument specifies which features to apply the new formula to.

```r
sim <- sim |>
  mutate(1:2, link = ~ ns(time, df = 7)) |>
  estimate(nu = 0.01, mstop = 1000)

sample(sim) |>
```

```r
exper_lineplot()
```

```
## Joining with `by = join_by(newdata_index)`
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
```



```r
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Australia/Melbourne
## tzcode source: internal
##
## attached base packages:
##  [1] splines   parallel  stats4    stats     graphics  grDevices utils     datasets
##
## other attached packages:
```

```
##  [1] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1               XVector_0.44.0
##  [7] MIGsim_0.0.0.9000               tidyr_1.3.1                     tibble_3.2.1
## [13] lattice_0.22-6                  MASS_7.3-60.2                   glue_1.7.0
## [19] mboost_2.9-10                   stabs_0.6-4                     forcats_1.0.0
## [25] GenomicRanges_1.56.0            GenomeInfoDb_1.40.0             IRanges_2.38.0
## [31] matrixStats_1.3.0               SpiecEasi_1.1.3                 CovTools_0.5.4
##
## loaded via a namespace (and not attached):
##   [1] minpack.lm_1.2-4            XML_3.99-0.16.1             rpart_4.1.23                life
##   [8] MultiAssayExperiment_1.30.2 insight_0.20.1              magrittr_2.0.3              limm
##  [15] RColorBrewer_1.1-3          ADGofTest_0.3               abind_1.4-5                 zlib
##  [22] kde1d_1.0.7                 rgl_1.3.1                   yulab.utils_0.1.4           prac
##  [29] tidytree_0.4.6              genefilter_1.86.0           ellipse_0.5.0               RSpe
##  [36] shapes_1.2.7                tidyselect_1.2.1            shape_1.4.6.1               UCSC
##  [43] jsonlite_1.8.8              Formula_1.2-5               survival_3.7-0              iter
##  [50] progress_1.2.3              treeio_1.28.0               ragg_1.3.2                  Rcpp
##  [57] SparseArray_1.4.8           mgcv_1.9-1                  xfun_0.44                   dist
##  [64] fansi_1.0.6                 digest_0.6.35               R6_2.5.1                    text
##  [71] copula_1.1-3                flare_1.7.0.1               utf8_1.2.4                  gene
##  [78] httr_1.4.7                  htmlwidgets_1.6.4           S4Arrays_1.4.1              scat
##  [85] gtable_0.3.5                blob_1.2.4                  pcaPP_2.0-4                 html
##  [92] SHT_0.1.8                   png_0.1-8                   knitr_1.47                  rstu
##  [99] stringr_1.5.1               libcoin_1.0-10              AnnotationDbi_1.66.0        pill
## [106] huge_1.3.5                  xtable_1.8-4                gamlss.dist_6.1-1           eval
## [113] locfit_1.5-9.9              compiler_4.4.0              rlang_1.1.4                 cray
## [120] stringi_1.8.4               BiocParallel_1.38.0         nnls_1.5                    asse
## [127] lazyeval_0.2.2              glmnet_4.1-8                Matrix_1.7-0                hms_
## [134] statmod_1.5.0               highr_0.11                  rbibutils_2.2.16            part
## [141] ape_5.8
```

Chapter 2

Simulating Differential Abundance

Before we consider simulating entire microbial communities, with their complex correlation structures, let's learn simulators for individual taxa. This is already enough to analyze taxon-level differential abundance approaches. For example, at the end of this session, we'll apply a simulator to study the power and false discovery rate of limma-voom when applied to microbiome data (as opposed to the bulk RNA-seq data for which it was originally proposed). Also, marginal modeling is a first step towards multivariate (community-wide) modeling, which we'll explore in the next session.

Let's load the necessary packages. Instructions for `scDesigner` and `MIGsim` can be found in the pre-workshop announcement. `SummarizedExperiment` is on Bioconductor, and all other packages are on CRAN.

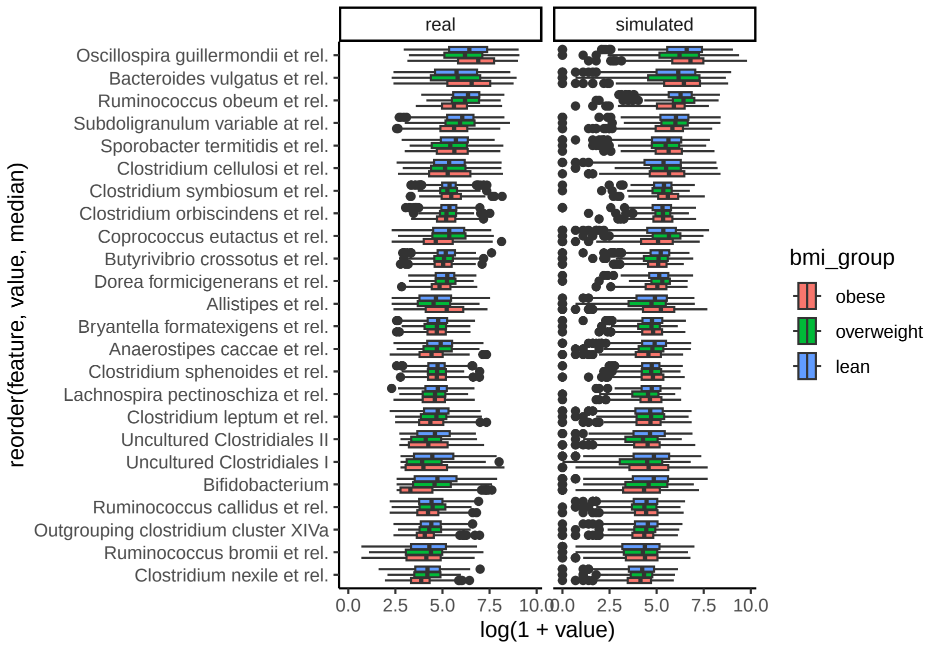
Let's train a simulator to fit the Atlas dataset. We'll use two covariates. `bmi_group` is the main covariate of interest – we want to see how microbiome composition varies among people with different BMI. The `log_depth` term is used to adjust for differential sequencing depths. We found it helpful to fixed zero inflation across the population (`nu`), so we have set `nu = ~1`. Finally, since we want to eventually evaluate testing methods that are designed for count data, we have used the (Z)ero (I)nflated (N)egative (B)inomial location-shape-scale model.

```r
data(atlas)

fmla <- list(
  mu = ~ bmi_group + log_depth,
  sigma = ~ bmi_group + log_depth,
  nu = ~1
)
sim <- setup_simulator(atlas, fmla, ~ ZINBLSS()) |>
  estimate(nu = 0.01, mstop = 1000)
```

2.1 Critique

Exercise: The block below combines the real and simulated experiments and visualizes their difference. With your neighbors, discuss how well the simulator approximates the original template.

```
combined <- bind_rows(
  real = pivot_experiment(atlas), # real data
  simulated = pivot_experiment(sample(sim)), # simulated
  .id = "source"
)

ggplot(combined) +
  geom_boxplot(
    aes(log(1 + value), reorder(feature, value, median), fill = bmi_group)
  ) +
  facet_grid(. ~ source)
```



Solution: The clearest difference is that, for these more abundant taxa, there there are not many low or zero counts. In contrast, the simulated data tend to have a long left tail (the many outlier circles on the left side of the boxplots), reflecting the fact that samples from the negative binomial distribution usually have support for all counts ≥ 0. Nonetheless, the ordering of abundances between the groups typically agrees between the real and simulated data. The interquartile ranges for each taxon also seem to roughly match.

2.2 Power Analysis Loop

To run a power analysis, we need to define datasets that have known ground truth. Then, we can run any differential abundance methods we want and see how many of the true associations are recovered (and how many nulls are falsely rejected). To this end, we'll remove associations from 16 of the original 24 genera, just like we removed group interactions in our spline fits above. We'll choose to remove the 16 that have the weakest associations in the original data. This is helpful because, even if we use `bmi_group` in our formula, if in reality there is no (or very weak) effect, then even if our simulator considers it as a true signal, the difference may be hard to detect. Eventually, our package will include functions for modifying these effects directly; at this point, though, we can only indirectly modify parameters by re-estimating them with new formulas.

```
nulls <- differential_analysis(atlas, "LIMMA_voom") |>
  rownames() |>
  tail(16)

null_fmla <- list(mu = ~log_depth, sigma = ~log_depth, nu = ~1)
sim <- sim |>
  mutate(any_of(nulls), link = null_fmla) |>
  estimate(nu = 0.01, mstop = 1000)
```

Now that we have ground truth associations, we'll evaluate LIMMA-voom for differential analysis. We consider sample sizes ranging from 50 to 1200, and we simulate 10 datasets for each sample size.

```
config <- expand.grid(
  sample_size = floor(seq(50, 1200, length.out = 5)),
  n_rep = 1:10
) |>
  mutate(run = as.character(row_number()))

results <- list()
for (i in seq_len(nrow(config))) {
  atlas_ <- sample_n(sim, config$sample_size[i])
  results[[i]] <- differential_analysis(atlas_, "LIMMA_voom") |>
    da_metrics(nulls, level = 0.3)
  print(glue("{i}/{nrow(config)}"))
}
```

Exercise: Visualize the results. How would you interpret the results of the power analysis? Based on your earlier critique of the simulator, do you think the estimated power here is conservative, liberal, or about right?

Solution: We'll use the `stat_pointinterval` function from the `ggdist` package to visualize the range of empirical power estimates across sample sizes. We can see that the average false discovery proportion is always controlled below 0.3, though the variance in this proportion can be quite high. We can also see that we would have quite good power with $n \geq 625$ samples, but the worst case scenarios can be quite poor for anything with fewer samples.

```
bind_rows(results, .id = "run") |>
  left_join(config) |>
  ggplot() +
  stat_pointinterval(aes(factor(sample_size), value)) +
  facet_wrap(~metric, scales = "free")
```

We expect that this result is somewhat conservative. This is because the original data have more symmetric distributions than our simulation, so limma's transformation to normality is likely easier to accomplish than in our more highly skewed data.

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Australia/Melbourne
## tzcode source: internal
##
## attached base packages:
##  [1] splines   parallel  stats4    stats     graphics  grDevices utils     datasets
##
## other attached packages:
##  [1] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1               XVector_0.44.0
##  [7] MIGsim_0.0.0.9000               tidyr_1.3.1                     tibble_3.2.1
## [13] lattice_0.22-6                  MASS_7.3-60.2                   glue_1.7.0
## [19] mboost_2.9-10                   stabs_0.6-4                     forcats_1.0.0
## [25] GenomicRanges_1.56.0            GenomeInfoDb_1.40.0             IRanges_2.38.0
## [31] matrixStats_1.3.0               SpiecEasi_1.1.3                 CovTools_0.5.4
```

```
##
## loaded via a namespace (and not attached):
##   [1] libcoin_1.0-10          RColorBrewer_1.1-3      rstudioapi_0.16.0       jsonlite_1.8.8
##   [9] fs_1.6.4                zlibbioc_1.50.0         vctrs_0.6.5             memoise_2.0.1
##  [17] S4Arrays_1.4.1          distributional_0.4.0    SparseArray_1.4.8       Formula_1.2-5
##  [25] copula_1.1-3            igraph_2.0.3            lifecycle_1.0.4         minpack.lm_1.2-4
##  [33] fastmap_1.2.0           GenomeInfoDbData_1.2.12 rbibutils_2.2.16        numDeriv_2016.8-
##  [41] kde1d_1.0.7             ellipse_0.5.0           labeling_0.4.3          pspline_1.0-20
##  [49] compiler_4.4.0          withr_3.0.0             doParallel_1.0.17       gsl_2.1-8
##  [57] scatterplot3d_0.3-44    tools_4.4.0             ape_5.8                 quadprog_1.5-8
##  [65] reshape2_1.4.4          generics_0.1.3          gtable_0.3.5            flare_1.7.0.1
##  [73] foreach_1.5.2           pillar_1.9.0            stringr_1.5.1           yulab.utils_0.1.
##  [81] tidyselect_1.2.1        locfit_1.5-9.9          ADGofTest_0.3           knitr_1.47
##  [89] expm_0.999-9            statmod_1.5.0           geigen_2.3              stringi_1.8.4
##  [97] evaluate_0.23           codetools_0.2-20        SHT_0.1.8               cli_3.6.2
## [105] Rcpp_1.0.12             rngWELL_0.10-9          randtoolbox_2.0.4       assertthat_0.2.1
## [113] tidytree_0.4.6          mvtnorm_1.2-5           scales_1.3.0            pcaPP_2.0-4
## [121] rvinecopulib_0.6.3.1.1
```

Chapter 3

Multivariate Power Analysis

How can we choose sample sizes in more complex bioinformatic workflows, where we simultaneously analyze many features (taxa, genes, metabolites) in concert? While traditional, analytical power analysis often breaks down, simulation can still be effective. We'll look at a concrete case study where we try to choose a good sample size for a sparse partial least squares discriminant analysis (sPLS-DA) of the Type I Diabetes (T1D) gut microbiome.

First, we'll load the required packages. Relative to our first session, the only additional package that we need is `mixOmics`. This can be installed using `BiocManager::install('mixOmics')`.

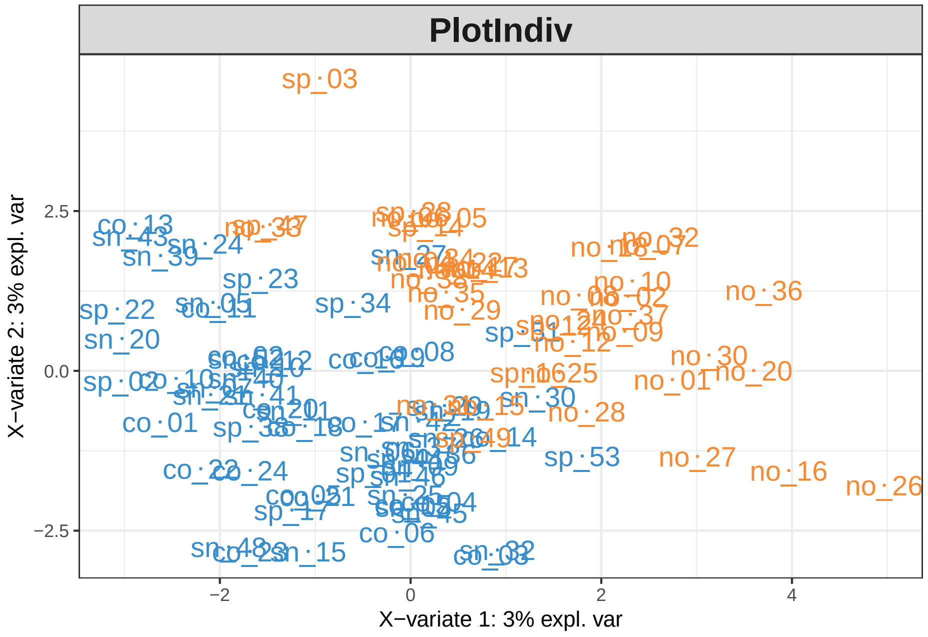
3.1 Interpreting PLS-DA

The T1D dataset below describes 427 metabolites from the gut microbiomes of 40 T1D patients and 61 healthy controls. These data were originally gathered by Gavin et al. (2018), who were motivated by the relationship between the pancreas and intestinal issues often experienced by T1D patients. They were especially curious about whether microbime-associated proteins might be related to increased risk for T1D development.

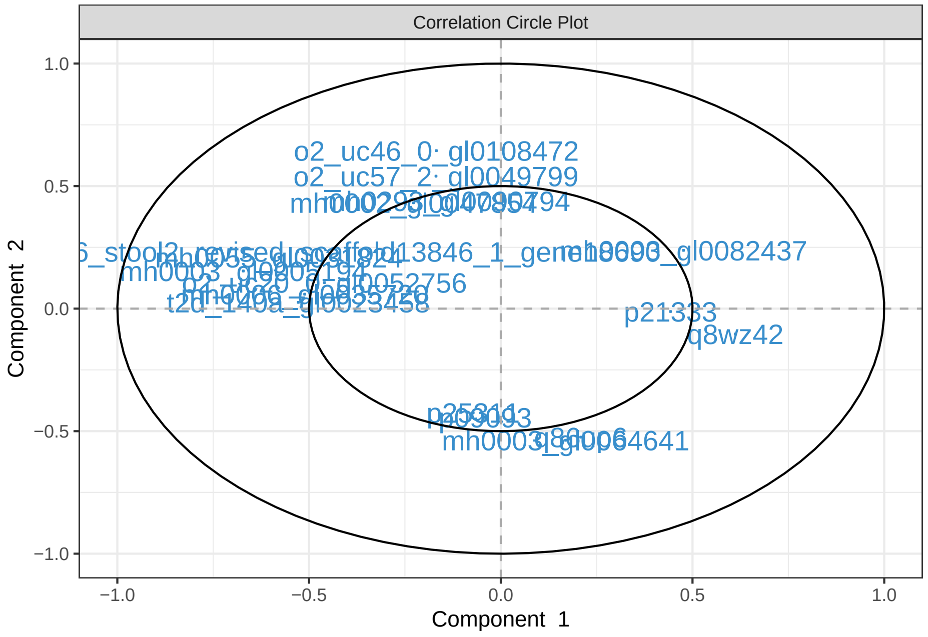
```
data(t1d)
```

The output that we care about are the PLD-DA scores and loadings. The wrapper below directly gives us this output without our having to explicitly set hyerparameters, though you can look here to see how the function was defined.

```
result <- splsda_fit(t1d)
plotIndiv(result$fit, cex = 5)
```



```
plotVar(result$fit, cutoff = 0.4, cex = 5)
```



Exercise: Discuss the output of `plotIndiv`. How does `plotVar` shape your interpretation?

Solution: The first two dimensions of the sPLS-DA pick on distinctions between the T1D (orange) and control (blue) groups. The variance by these first two dimensions is relatively small – variation in protein signatures may not be easiliy captured in two dimensions or may not generally be correlated with the disease response. Since most of the differences between groups are captured by the first dimension, we can look for features that are highly correlated with this dimension to see which proteins might be driving the difference. For example, we can conclude that T1D samples tend to have higher levels of p21333 and q8wz42 compared to the controls.

3.2 Estimation

Let's estimate a simulator where every protein is allowed ot vary across T1D type. Since the data have already been centered log-ratio transformed, it's okay to treat these as Gaussian.

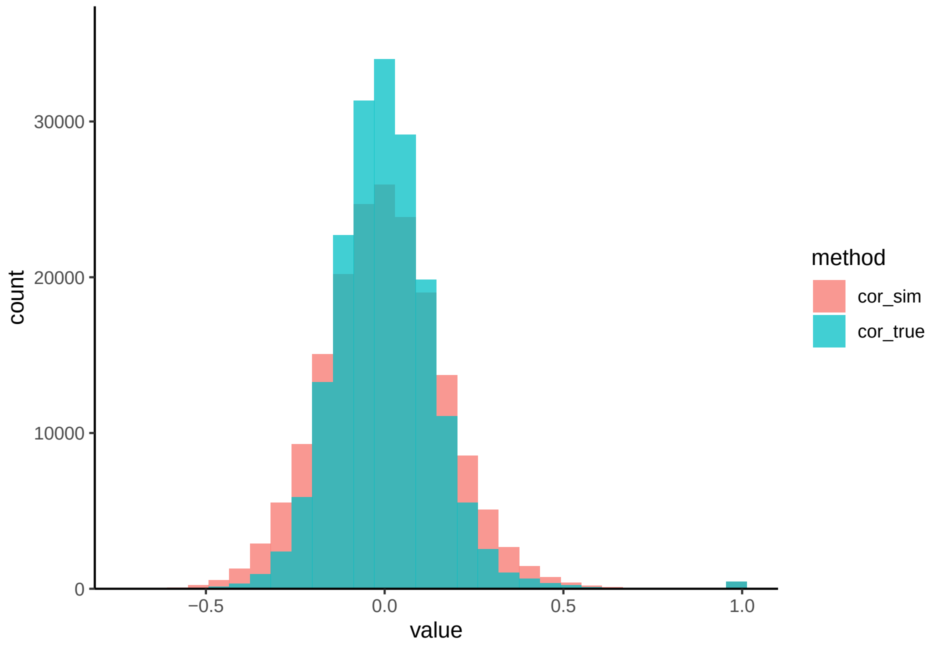
```
simulator <- setup_simulator(t1d, link_formula = ~outcome2, family = ~ GaussianLSS()) |>
  estimate(mstop = 100)
```

3.3 Evaluation

Last time, we saw how we could visualize marginal simulator quality. How can we tell whether a joint simulator is working, though? One simple check is to analyze the pairwise correlations. Since the copula model is designed to capture second-order moments, it should at the very least effectively capture the correlations.

We've written a small helper that visualizes the pairwise protein-protein correlations from both the real and the simulated datasets. We seem to be often overestimating the correlation strength. This is likely a consequence of the high-dimensionality of the problem.

```
sim_exper <- sample(simulator)
correlation_hist(t1d, sim_exper)
```

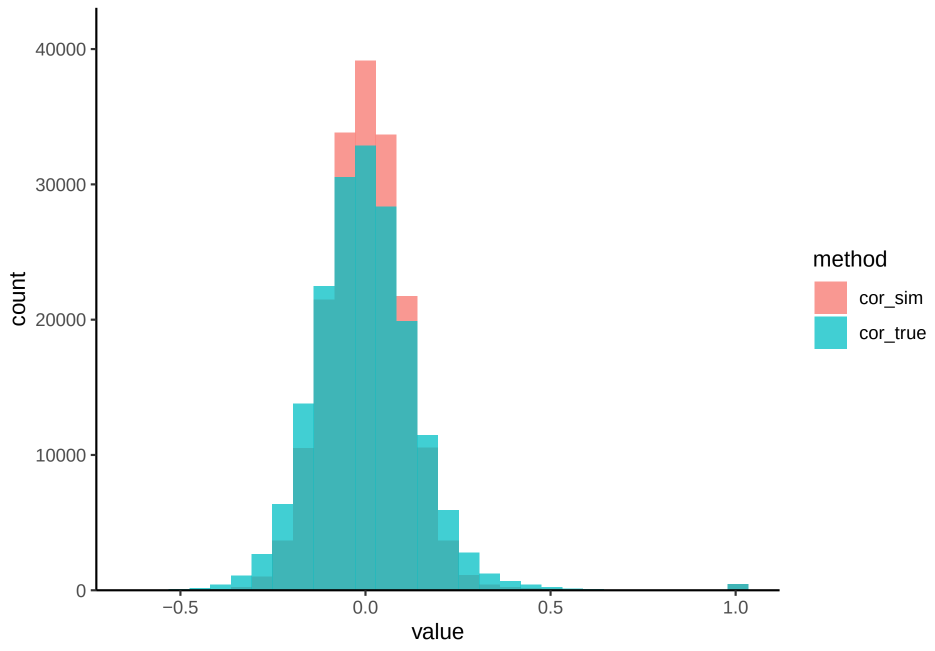


Exercise: To address this, let's try modifying the `copula_def` argument of `setup_simulator` to use a more suitable simulator. Generate new correlation histograms and comment on the changes you observe. You only need to modify the commented lines (`#`) lines in the block below.

```r
simulator <- setup_simulator(
  t1d,
  link_formula = ~outcome2,
  family = ~ GaussianLSS(),
  copula_def = # fill this code in
  ) |>
  estimate(mstop = 100)

sim_exper <- sample(simulator) # and then run these two lines
correlation_hist(t1d, sim_exper) #
```

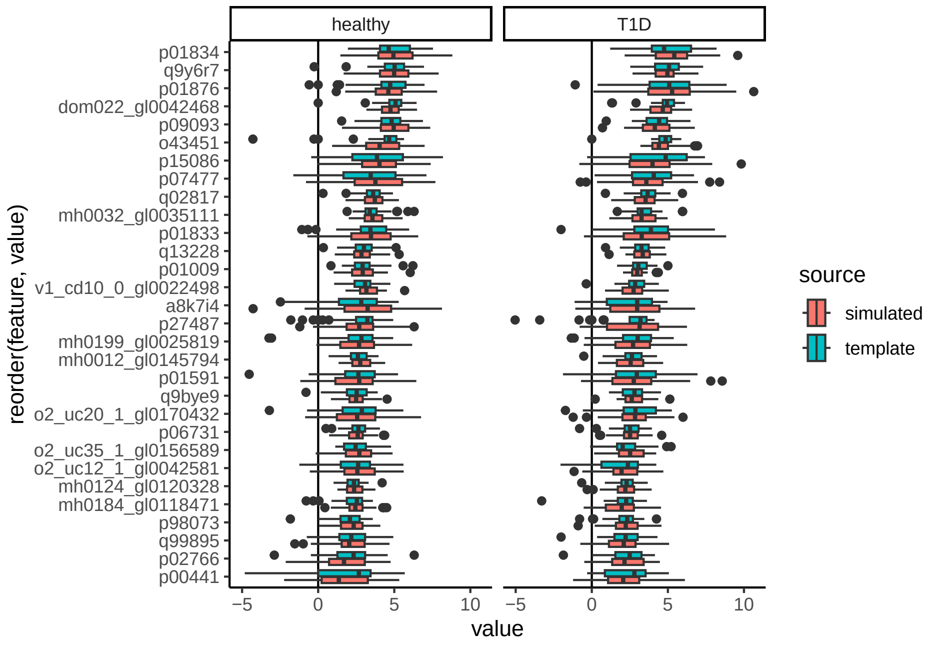
Solution: For our coupla, we can use a covariance estimator of Cai and Liu (2011), that is suited for high dimensions. Larger values of `thr` will increase the stability of our estimates, but at the cost of potentially missing or weakening true correlations. In line with this point, our new simulated correlations are more concentrated.

```r
simulator <- setup_simulator(
  t1d,
  link_formula = ~outcome2,
  family = ~ GaussianLSS(),
  copula_def = copula_adaptive(thr = 0.4)
) |>
  estimate(mstop = 100)

sim_exper <- sample(simulator)
correlation_hist(t1d, sim_exper)
```



:::

```
contrast_boxplot(t1d, sim_exper, ~ . ~ reorder(outcome2, value))
```



3.4 PLS-DA Power Analysis

Now that we have a simulator, we can run a power analysis. In theory, we could look at how any summary from the PLS-DA output varies as the sample size increases. The most natural one, though, is simply to see how classifier performance improves as we gather more samples. Specifically, we'll measure the holdout Area Under the Curve (auc), a measure of how well the trains PLS-DA classifier balance precision and recall on new samples.

Moreover, we'll study the effect of sparsity – what happens when many features have no relationship at all with the response? We'll also simulate three hypothetical datasets for each sample size and sparsity level. All configurations of interest are stored in the `config` matrix below.

```r
config <- expand.grid(
  sample_size = floor(seq(15, 150, length.out = 5)),
  n_rep = 1:3,
  n_null = floor(seq(317, 417, length.out = 4)),
  metrics = NA
)

data(t1d_order)
```

Exercise: Finally, we're in a position to generate synthetic data and evaluate PLS-DA performance. Fill in the block below to update the simulator for each `i`.

Remember that the original `simulator` defined above assumes that all proteins are associated with T1D. You can use `t1d_order` to prioritize the proteins with the strongest effects in the original data. As before, you should only need to modify the line marked with the comments (`#`).

```r
for (i in seq_len(nrow(config))) {
  simulator <- simulator |>
    mutate(
      # fill this in
    ) |>
    estimate(mstop = 100)

  config$metrics[i] <- (sample_n(simulator, config$sample_size[i]) |>
    splsda_fit())[["auc"]]
  print(glue("run {i}/{nrow(config)}"))
}
```

Solution: To define nulls, we mutate the weakest proteins so that there is no longer any association with T1D: `link = ~ 1` instead of `link = ~ outcome2`. To speed up the computation, we organized the `mutate` calls so that we don't need to re-estimate proteins whose effects were removed in a previous iteration.

```r
for (i in seq_len(nrow(config))) {
  if (i == 1 || config$n_null[i] != config$n_null[i - 1]) {
    simulator <- simulator |>
      mutate(any_of(rev(t1d_order)[1:config$n_null[i]]), link = ~1) |>
      estimate(mstop = 100)
  }

  config$metrics[i] <- (sample_n(simulator, config$sample_size[i]) |>
    splsda_fit())[["auc"]]
  print(glue("run {i}/{nrow(config)}"))
}
```

:::

We can visualize variation in performance.

```r
ggplot(config, aes(sample_size, metrics, col = factor(n_null))) +
  geom_point() +
  facet_wrap(~n_null)
```

Discussion: Interpret the visualization above. How do you think analysis like this could help you justify making some experimental investments over others?

Solution: Reading across facets from top left to bottom right, power decreases when the number of null proteins increases. It seems that sPLS-DA can benefit from having many weakly associated features. While power is sometimes high in low sample sizes, the variance can be quite large. In all settings, there is a noticeable decrease in variance in power as we go from 15 to 48 samples. If we can assume that a moderate fraction (> 15%) of measured proteins are associated with T1D, then we may already achieve good power with ~ 100 samples. However, if we imagine our effect might be sparser in our future experiment, then this figure would give us good justification for arguing for a larger number of samples, in order to ensure we can discover a disease-associated proteomics signature.

```r
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Australia/Melbourne
## tzcode source: internal
##
## attached base packages:
##  [1] splines   parallel  stats4    stats     graphics  grDevices utils     datasets
##
## other attached packages:
##  [1] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1               XVector_0.44.0
##  [7] MIGsim_0.0.0.9000               tidyr_1.3.1                     tibble_3.2.1
## [13] lattice_0.22-6                  MASS_7.3-60.2                   glue_1.7.0
## [19] mboost_2.9-10                   stabs_0.6-4                     forcats_1.0.0
## [25] GenomicRanges_1.56.0            GenomeInfoDb_1.40.0             IRanges_2.38.0
## [31] matrixStats_1.3.0               SpiecEasi_1.1.3                 CovTools_0.5.4
##
## loaded via a namespace (and not attached):
##   [1] minpack.lm_1.2-4            XML_3.99-0.16.1             rpart_4.1.23
##   [8] MultiAssayExperiment_1.30.2 insight_0.20.1              magrittr_2.0.3
##  [15] RColorBrewer_1.1-3          ADGofTest_0.3               abind_1.4-5
##  [22] kde1d_1.0.7                 rgl_1.3.1                   yulab.utils_0.1.4
##  [29] tidytree_0.4.6              genefilter_1.86.0           ellipse_0.5.0
##  [36] shapes_1.2.7                tidyselect_1.2.1            shape_1.4.6.1
##  [43] jsonlite_1.8.8              Formula_1.2-5               survival_3.7-0
##  [50] progress_1.2.3              treeio_1.28.0               ragg_1.3.2
##  [57] SparseArray_1.4.8           mgcv_1.9-1                  xfun_0.44
##  [64] fansi_1.0.6                 digest_0.6.35               R6_2.5.1
##  [71] copula_1.1-3                flare_1.7.0.1               utf8_1.2.4
##  [78] httr_1.4.7                  htmlwidgets_1.6.4           S4Arrays_1.4.1              scat
##  [85] gtable_0.3.5                blob_1.2.4                  pcaPP_2.0-4                 htm]
##  [92] SHT_0.1.8                   png_0.1-8                   knitr_1.47                  rstu
##  [99] stringr_1.5.1               libcoin_1.0-10              AnnotationDbi_1.66.0        pil]
## [106] huge_1.3.5                  xtable_1.8-4                gamlss.dist_6.1-1           eval
## [113] locfit_1.5-9.9              compiler_4.4.0              rlang_1.1.4                 cray
## [120] stringi_1.8.4               BiocParallel_1.38.0         nnls_1.5                    asse
## [127] lazyeval_0.2.2              glmnet_4.1-8                Matrix_1.7-0                hms_
## [134] statmod_1.5.0               highr_0.11                  rbibutils_2.2.16            part
## [141] ape_5.8
```

Chapter 4

Microbiome Networks

Unlike human social networks, there is no simple way to observe microbe-microbe interactions – we have to make do with indirect evidence. One approach uses population profiles as a proxy for ecological interaction. Taxa that often co-occur are understood to have cooperative ecological interactions, while those that don't are thought to compete for the same niche.

Many algorithms have been designed around this intuition, all trying to go beyond simple co-occurrence and instead capture more complex types of dependence. A challenge in practice is that it's hard to know which method to use when, since the problem is unsupervised. Even when thorough simulation benchmarking studies are available, it's often not obvious how well those simulation setups match our problems of interest.

4.1 Estimation

Let's use simulation to benchmark network estimation methods using data from rounds 1 and 2 of the American Gut Project. We will simulate data with known correlation structure and taxa-level marginals estimated from the study data. The block below reads in the data.

```r
data(amgut)
amgut
```

```
## class: SummarizedExperiment
## dim: 45 261
## metadata(0):
## assays(1): counts
## rownames(45): 326792 181016 ... 364563 301645
## rowData names(0):
## colnames(261): 000001879.1076223 000002437.1076448 ... 000002656.1076186 000001582.1076447
## colData names(167): X.SampleID BarcodeSequence ... Description sequencing_depth
```

We've estimated a zero-inflated negative binomial location-shape-scale (`ZINBLSS`) models for each taxon, using a gaussian copula to capture dependence. We have used the regression formula `~log(sequencing_depth) + BMI`. The data structure below captures all the simulator components, and we can swap pieces in and out to modify the form of the simulator. For example, if we wanted, we could `mutate` the family and link function associated with particular features.

```
sim <- setup_simulator(
  amgut,
  ~ log(sequencing_depth) + BMI,
  ~ ZINBLSS()
) |>
  estimate(mstop = 100)
sim
```

```
## [Marginals]
## Plan:
## # A tibble: 6 x 3
##     feature              family                             link
##   <gene_id>             <distn>                           <link>
## 1    326792 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
## 2    181016 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
## 3    191687 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
## 4    326977 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
## 5    194648 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
## 6    541301 ZINBI [mu,sigma,nu] ~log(sequencing_depth) + BMI
##
## Estimates:
## # A tibble: 3 x 2
##   feature fit
##   <chr>   <list>
## 1 326792  <glmbsLSS>
## 2 181016  <glmbsLSS>
## 3 191687  <glmbsLSS>
## ... and 42 additional features.
##
## [Dependence]
## 1 normalCopula with 45 features
##
## [Template Data]
## class: SummarizedExperiment
## dim: 45 261
## metadata(0):
## assays(1): counts
## rownames(45): 326792 181016 ... 364563 301645
## rowData names(0):
## colnames(261): 000001879.1076223 000002437.1076448 ... 000002656.1076186 000001582.1076447
## colData names(167): X.SampleID BarcodeSequence ... Description sequencing_depth
```

The simulated data is always a `SummarizedExperiment`. This means that any workflow that applied to the original data can be applied to the simulated one without any changes. Notice also that `sample` defaults to drawing samples from the same design as the original input experiment (we'll modify this using the `new_data` argument in a minute).

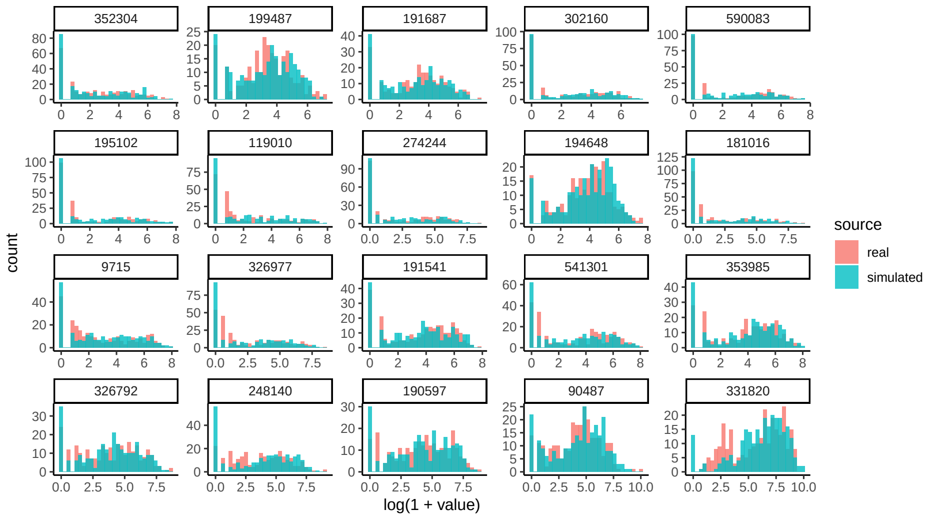
```r
simulated <- sample(sim)
simulated
```

```
## class: SummarizedExperiment
## dim: 45 261
## metadata(0):
## assays(1): counts_1
## rownames(45): 326792 181016 ... 364563 301645
## rowData names(0):
## colnames(261): 000001879.1076223 000002437.1076448 ... 000002656.1076186 000001582.1076447
## colData names(167): X.SampleID BarcodeSequence ... Description sequencing_depth
```

4.2 Evaluation

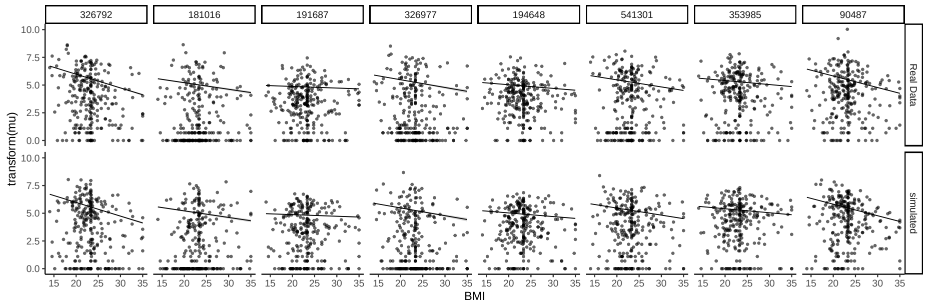
Let's compare the marginal count distributions for the real and simulated data. We'll need the data in "long" format to be able to make the ggplot2 figure. The `pivot_experiment` helper can transform the original `SummarizedExperiment` objects in this way. Notice that the simulated data tends to overestimate the number of zeros in the high-abundance taxa. To refine the simulator, we should probably replace the zero-inflated negative binomial with ordinary negative binomials for these poorly fitted taxa.

```r
bind_rows(
  real = pivot_experiment(amgut),
  simulated = pivot_experiment(simulated),
  .id = "source"
) |>
  filter(feature %in% rownames(simulated)[1:20]) |>
  ggplot() +
  geom_histogram(
    aes(log(1 + value), fill = source),
    position = "identity", alpha = 0.8
  ) +
  facet_wrap(~ reorder(feature, value), scales = "free")
```



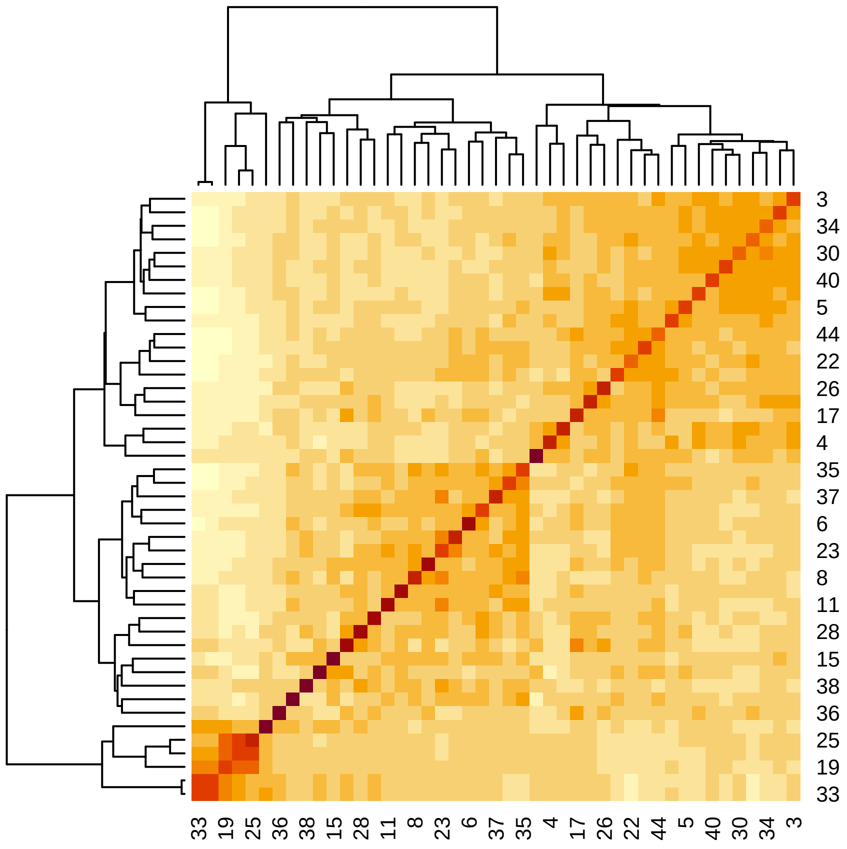
Are the learned relationships with BMI plausible? We can compare scatterplots of the real and simulated data against this variable. Note that, by default, the ribbons will be evaluated along all variables, which makes for the jagged ribbons (neighboring values for BMI might have different sequencing depth, potentially leading to quite different predictions). To remove this artifact, we can assume that all samples had exactly the same sequencing depth.

```r
new_data <- colData(amgut) |>
  as_tibble() |>
  mutate(sequencing_depth = 2e4)
plot(sim, "BMI", sample(sim, new_data = new_data), new_data = new_data)
```



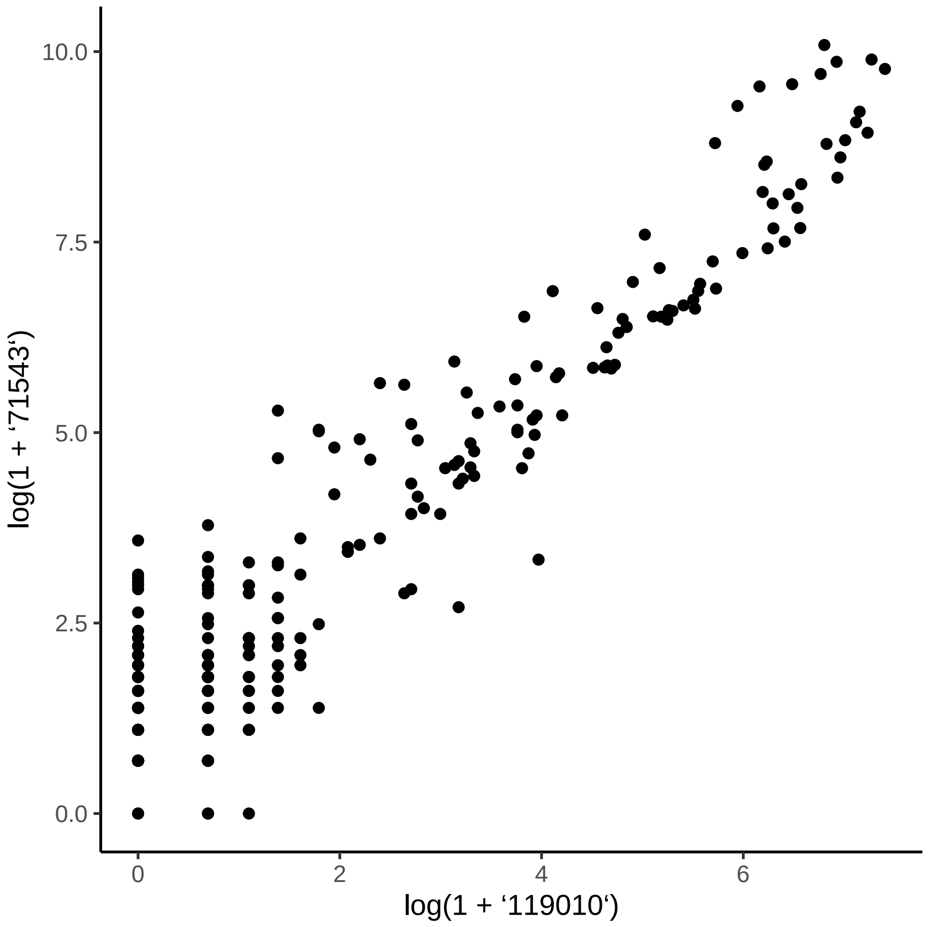
We next visualize the correlation matrix estimated by the simulator's copula model. There are a few pairs of taxa that are very highly correlated, and there are also a few taxa that seem to have higher correlation across a large number of taxa (e.g., the taxon in row 34). There is no obvious banding or block structure in this real data, though.

```r
rho <- copula_parameters(sim)
heatmap(rho)
```



The pair below is one of those with high positive correlation. You can replace the selection with the commented out line to see what one of the anticorrelated pairs of taxa looks like.

```
#taxa <- rownames(amgut)[c(33, 43)]
taxa <- rownames(amgut)[c(14, 25)]
pivot_experiment(amgut) |>
  filter(feature %in% taxa) |>
  pivot_wider(names_from = feature) |>
  ggplot() +
  geom_point(aes(log(1 + .data[[taxa[1]]]), log(1 + .data[[taxa[2]]])))
```



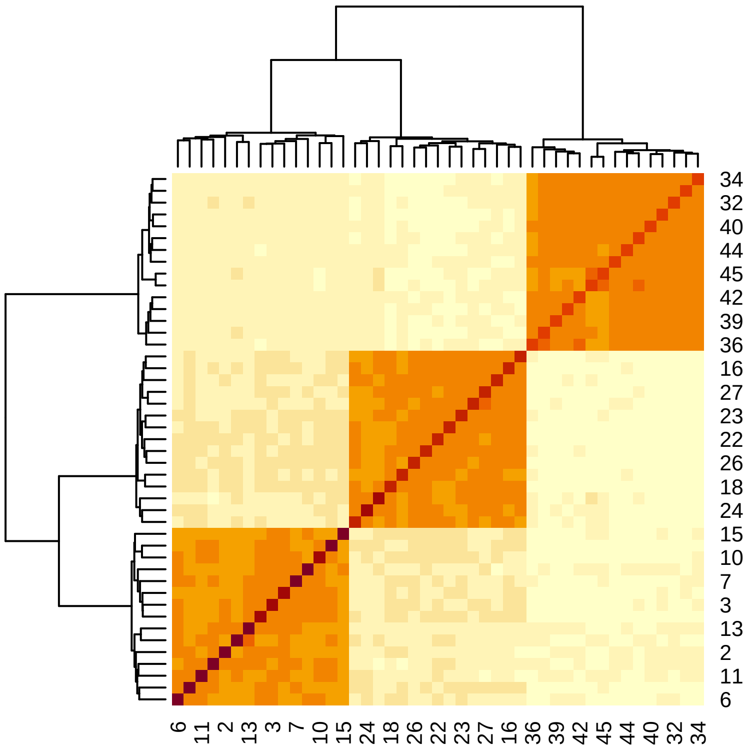
4.3 Block Covariance

Let’s replace the current copula correlation structure with one from a block diagonal matrix. In this example, the off-diagonal correlations are 0.6. We can use `mutate_correlation` to swap this new correlation matrix into our earlier simulator.

```
rho <- c(0.4, .6, 0.8) |>
  map(~ matrix(., nrow = 15, ncol = 15)) |>
  Matrix::bdiag() |>
  as.matrix()
diag(rho) <- 1

simulated <- sim |>
  mutate_correlation(rho) |>
  sample()

x <- t(assay(simulated))
```

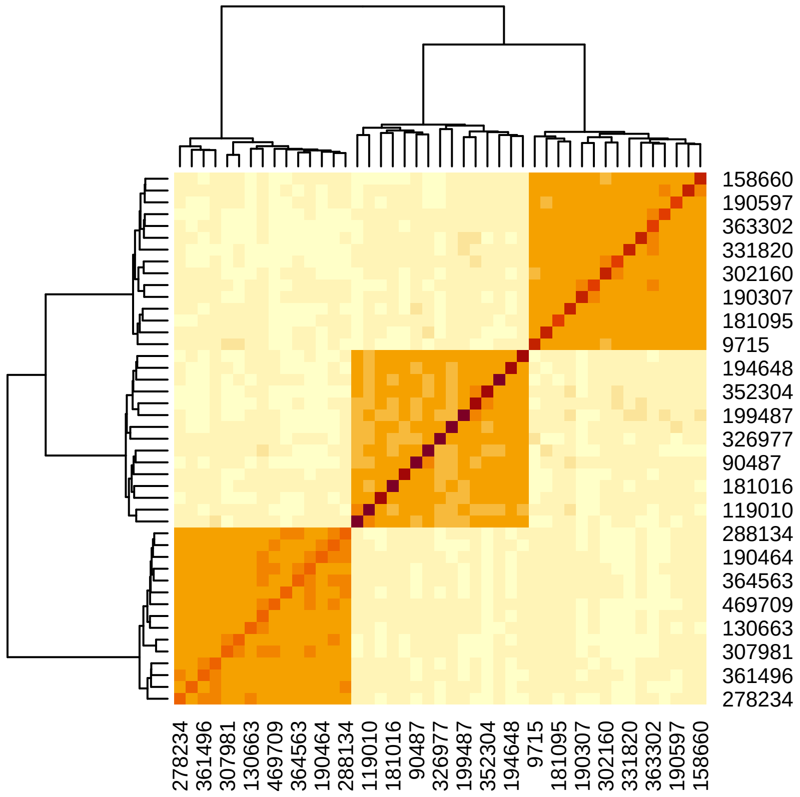
Let's first look at the `SpiecEasi` covariance estimate. This is a variant of the graphical lasso that is designed to be well-adapted to microbiome data. The good news is that it does warn that the default choices of λ are too large, which is correct in this case. Unfortunately, it took a while to get this answer, and we had already been quite generous in allowing it to fit 10 choices of λ.

```
rho_se <- spiec.easi(x, nlambda = 10, pulsar.params = list(rep.num = 1)) |>
  getOptCov() |>
  as.matrix() |>
  cov2cor()
heatmap(rho_se)
```



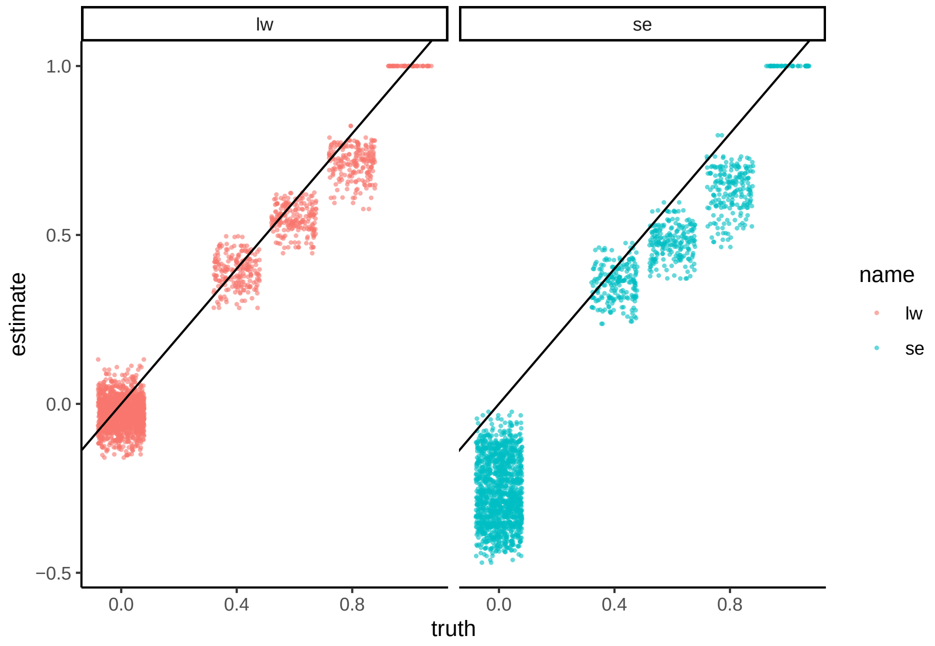
Let's instead use the Ledoit-Wolf estimator on the log-transformed data. The results make much more sense.

```
rho_lw <- CovEst.2003LW(log(1 + x))$S |>
  cov2cor()
heatmap(rho_lw)
```



Since color comparisons are difficult to evaluate precisely, we can also make a scatterplot comparing the different covariance estimators.

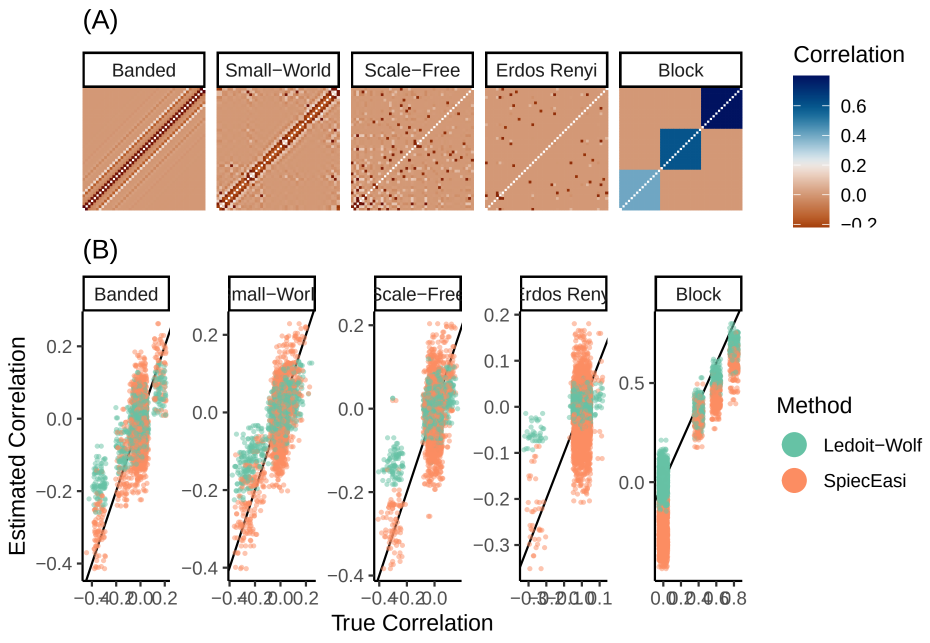
```r
data.frame(truth = c(rho), se = c(rho_se), lw = c(rho_lw)) |>
  pivot_longer(-truth, values_to = "estimate") |>
  ggplot() +
  geom_jitter(aes(truth, estimate, col = name), alpha = 0.6, size = 0.4) +
  geom_abline(slope = 1) +
  facet_wrap(~name)
```



4.4 Generalization

What about other network structures? We can use the same logic to evaluate several network regimes. For example, the block below defines correlation matrices associated with scale-free and banded structures.

```r
data(example_rho)
rho_hat <- list()
for (r in seq_along(example_rho)) {
  x <- sim |>
    mutate_correlation(example_rho[[r]]) |>
    scDesigner::sample() |>
    assay()

  rho_se <- spiec.easi(t(x), nlambda = 10, pulsar.params = list(rep.num = 1)) |>
    getOptCov() |>
    as.matrix() |>
    cov2cor()
  rho_lw <- CovEst.2003LW(log(1 + t(x)))$S |>
    cov2cor()
  rho_hat[[names(example_rho)[r]]] <- list(se = rho_se, lw = rho_lw)
}
```



This example shows that, when we start with real template data, it's not too hard to setup a benchmarking experiment. It's generally easier to reconfigure the components of an existing simulator than it is to specify all the simulation steps from scratch. There is the secondary bonus that the data tend to look close to real data of interest, at least up to the deliberate transformations needed to establish ground truth.

We could imagine extending this example to include different data properties (sample sizes, variable block sizes and correlations, more general correlation structure) and estimation strategies (alternative transformations or estimators). Design changes could be implemented using `expand_colData`, changes in the signal can be specified as above with `mutate_correlation`, and any workflow can be used as long as it applies to a `SummarizedExperiment` object.

Chapter 5

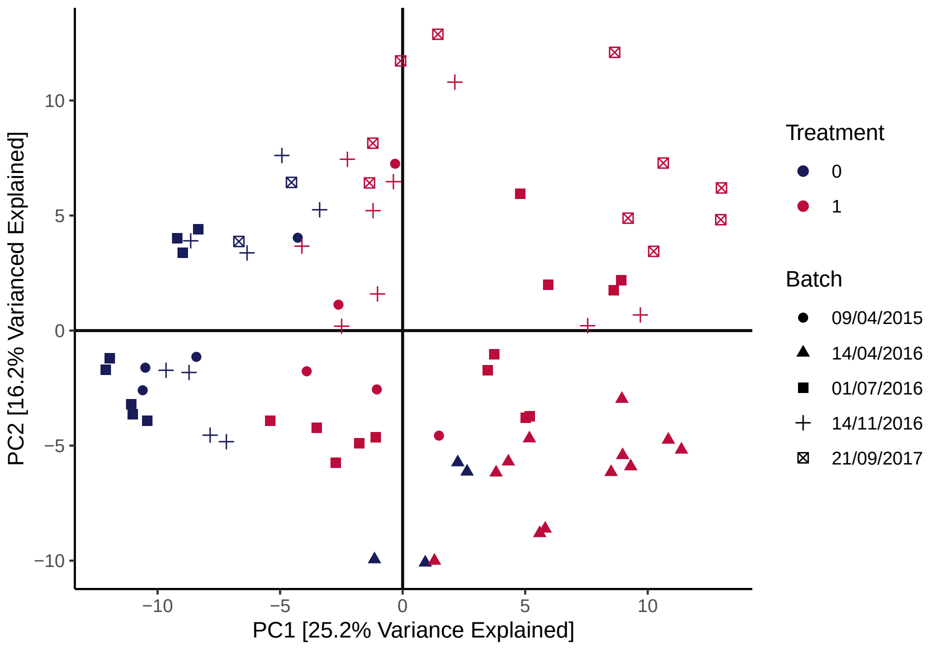
Batch Effect Correction

Integration can be a subtle exercise. We need to balance our interest in seeing similarities between datasets with the risk of making things seem more similar than they really are. Simulation can help navigate this subtlety by letting us see how integration methods would behave in situations where we know exactly how the different datasets are related. This note will illustrate this perspective by showing how simulation can help with both horizontal (across batches) and vertical (across assays) integration. We'll have a brief interlude on the `map` function in the `purrr`, which is helpful for concisely writing code that would otherwise need for loops (e.g., over batches or assays).

5.1 Anaerobic Digestion Data

This example is about simultaneously analyzing several batches in a dataset about the efficiency of anaerobic digestion (AD) of organic matter. The essential problem is that, in this study, the samples could not be collected simultaneously. Small differences across separate runs could lead to systematic differences in the resulting data, which can obfuscate the more interesting between-group variation that the experiment was intended to uncover. For example, in the AD dataset, the date of the sequencing run has a global effect on measured community composition, which we can see right away from a principal components plot:

```r
data(anaerobic)
pca_batch(anaerobic, facet = FALSE) +
  scale_color_manual(values = c("#191C59", "#bc0c3c")) +
  labs(
    col = "Treatment",
    shape = "Batch",
    x = "PC1 [25.2% Variance Explained]",
    y = "PC2 [16.2% Varianced Explained]"
  )
```



You can learn more about the general microbiome batch effect integration problem in (Wang and Le Cao, 2020), which is where this dataset example and the batch effect correction code below comes from. The article also reviews mechanisms that could lead to batch effects in microbiome data, together with methods for removing these effects and the situations within which they are most appropriate.

In batch effect correction, it's important to remove as much of the batch variation as possible without accidentally also removing the real biological variation that would have been present even if all the samples had been sequenced together. This is sometimes called "overintegration,' ' and this is an especially high risk if some of the real biological variation is quite subtle, e.g., a rare cell type or one that is very similar to a more prominent one. Simulation can help us gauge the extent to which different methods may or may not overintegrate. Since we get to control the between-batch and and between-biological-condition differences, we can see the extent to which integration methods can remove the former while preserving the latter.

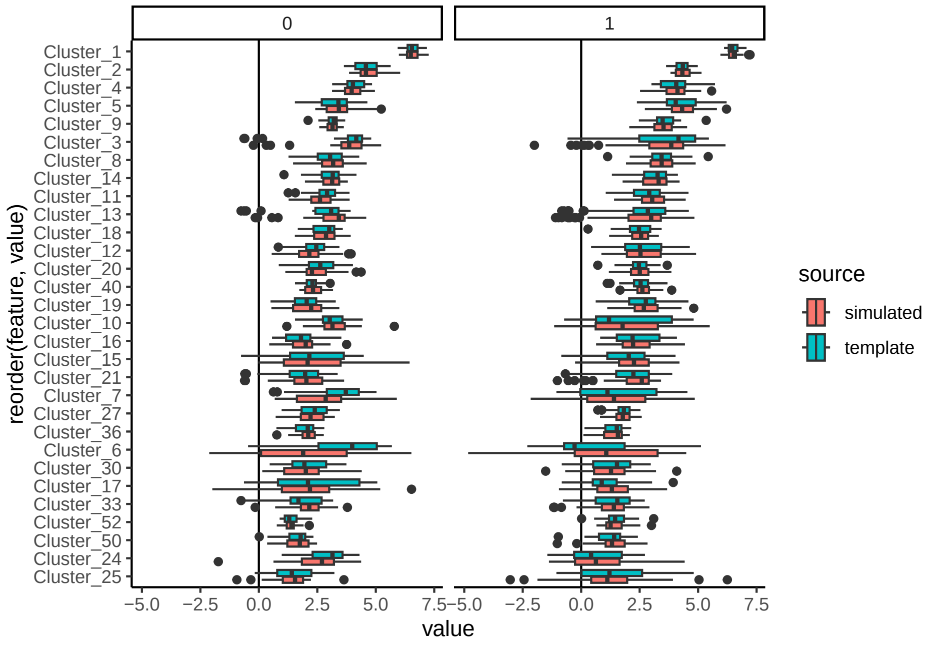
5.2 Simulation Design

The block below estimates a candidate simulator. By using the formula `~ batch + treatment`, we're allowing for taxon-wise differences due to batch and treatment. Note that in principle, we could estimate an interaction between batch and treatment (the treatment could appear stronger in some batches than others). I encourage you to try estimating that model; however, visually analyzing the output suggests that this full model has a tendancy to overfit. Since the data have already been centered log-ratio transformed, we can try out a Gaussian marginal model. The AD dataset has relatively few samples compared to the number of features, so we'll use a copula that's designed for this setting.

```
simulator <- setup_simulator(
  anaerobic,
  ~ batch + treatment,
  ~ GaussianLSS(),
  copula = copula_adaptive(thr = .1)
) |>
  estimate(nu = 0.05, mstop = 100) # lower nu -> stable training
```

We can simulate from the fitted model and evaluate the quality of our fit using `contrast_boxplot`. This is a light wrapper of the ggplot2 code we used to compare experiments from our first session, and you can read its definition here.

```
anaerobic_sim <- sample(simulator)
contrast_boxplot(anaerobic, anaerobic_sim)
```



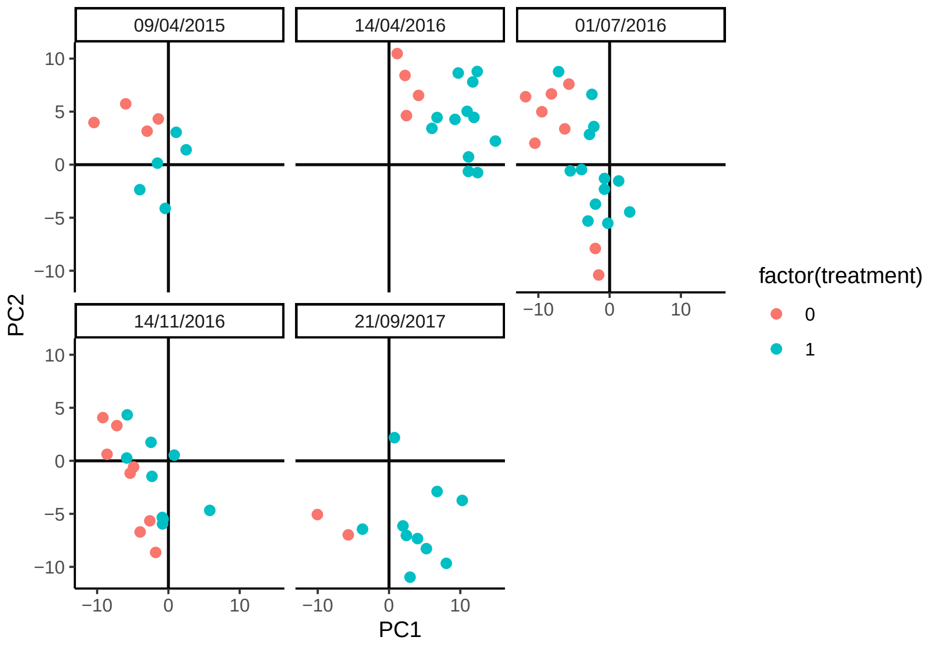
Exercise Propose and create at least one other visualization that can be used to compare and contrast the simulator with real data. What conclusions can you draw?

Solution: There are many possible answers:

- Boxplots across taxa with different overall abundance levels.
- Analogous histograms or CDF plots, to show the entire distributions, rather than just summarized quantiles.
- Pair scatterplots, to see how well the bivariate relationships between taxa are preserved.
- Dimensionality reduction on the simulated data, to see how well it matches global structure in the original data.

We'll implement the last idea using PCA. This should be contrasted with the PCA plot on the original data above. It's okay if the plot seems rotated relative to the oiginal plot – PCA is only unique up to rotation. The main characteristic we're looking for is that the relative sizes of the batch and treatment effects seem reasonaly well-preserved, since these will be the types of effects that our later batch effect integration methods must be able to distinguish.

```
pca_batch(anaerobic_sim)
```



:::

To study the risk for overintegration, let's imagine that there were a third treatment group with relatively fewer samples. This is the type of group that a correction method might accidentally blend in with the rest, if it's too aggressive. We've defined the imaginary experiment using the data.frame below. The `treatment` level `1.8` is the new one. We've supposed there are between 1 - 3 technical replicates (`extraction`) for each biological sample (`sample`), and the batch dates are the same as before.

```r
data(imaginary_design)
summary(imaginary_design)
```

```
##    extraction        batch      treatment           rep             sample
##  Min.   :1    09/04/2015:36   Min.   :0.0000   Min.   :1.00   1      :  2
##  1st Qu.:1    14/04/2016:36   1st Qu.:0.0000   1st Qu.:1.75   2      :  2
##  Median :2    01/07/2016:36   Median :1.0000   Median :2.50   3      :  2
##  Mean   :2    14/11/2016:36   Mean   :0.7167   Mean   :2.75   4      :  2
##  3rd Qu.:3    21/09/2017:36   3rd Qu.:1.0000   3rd Qu.:4.00   5      :  2
##  Max.   :3                    Max.   :1.8000   Max.   :5.00   6      :  2
##                                                                (Other):168
```

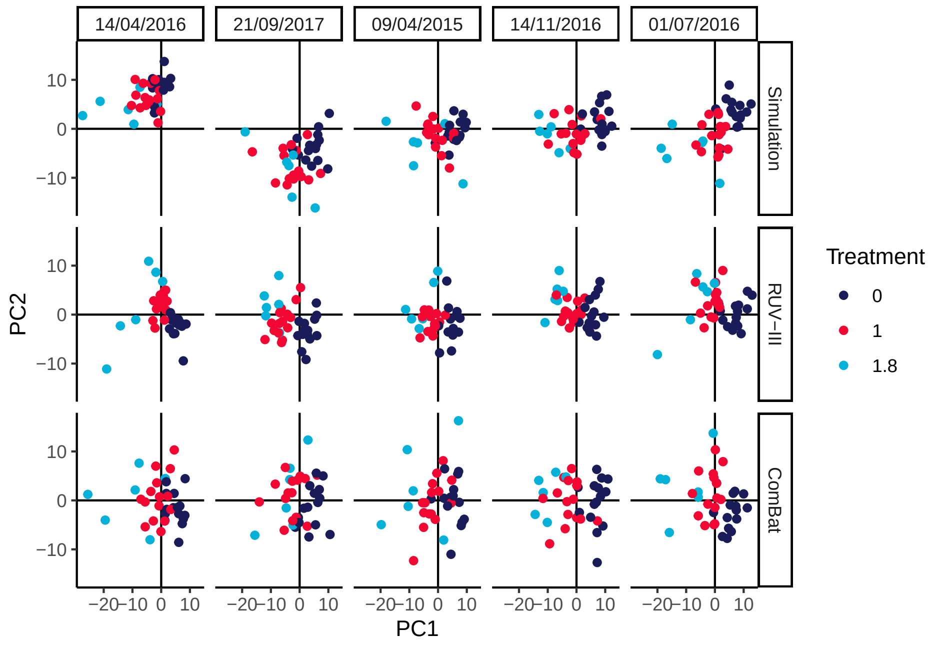
We can simulate from the new design and look at how different this new treatment group seems from the others. It's a subtle effect, definitely smaller than the batch effect, but also separate enough that we should be able to preserve it.

```r
p <- list()
anaerobic_sim <- sample(simulator, new_data = imaginary_design)
p[["sim"]] <- pca_batch(anaerobic_sim)
```

5.3 Benchmarking

We've defined a `batch_correct` wrapper function that implements either the RUV-III or ComBat batch effect correction methods. Their outputs are contrasted in the PCAs below. It looks like ComBat might be somewhat too aggressive, causing the `1` and `1.8` treatment groups to substantially overlap, while RUV is a bit more conservative, keeping the treatment groups nicely separate. As an aside, we note that this conclusion can depend on the number of replicates and total number of samples available. We've included the code for generating the `imaginary_design` data.frame in a vignette for the `MIGsim` package. Can you find settings that lead either method astray?

```r
p[["ruv"]] <- pca_batch(batch_correct(anaerobic_sim, "ruv"))
p[["combat"]] <- pca_batch(batch_correct(anaerobic_sim, "combat"))
```



```r
prediction_errors <- function(df) {
  y_hat <- predict(lda(treatment ~ PC1 + PC2, data = df), type = "response")$class
  table(df$treatment, y_hat)
}

map(p, ~ prediction_errors(.$data))
```

```
## $sim
##      y_hat
##        0  1 1.8
##   0   62 13   0
##   1   15 56   4
##   1.8  1 14  15
##
## $ruv
##      y_hat
##        0  1 1.8
##   0   70  5   0
##   1    2 67   6
##   1.8  0  8  22
##
## $combat
##      y_hat
##        0  1 1.8
##   0   65 10   0
##   1   12 59   4
##   1.8  1 12  17
```

```r
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.4
## 
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib 
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LA
## 
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
## 
## time zone: Australia/Melbourne
## tzcode source: internal
## 
## attached base packages:
##  [1] splines   parallel  stats4    stats     graphics  grDevices utils     datasets  methods
## 
## other attached packages:
##  [1] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1               XVector_0.44.0
##  [7] MIGsim_0.0.0.9000               tidyr_1.3.1                     tibble_3.2.1
## [13] lattice_0.22-6                  MASS_7.3-60.2                   glue_1.7.0
## [19] mboost_2.9-10                   stabs_0.6-4                     forcats_1.0.0
## [25] GenomicRanges_1.56.0            GenomeInfoDb_1.40.0             IRanges_2.38.0
## [31] matrixStats_1.3.0               SpiecEasi_1.1.3                 CovTools_0.5.4
## 
## loaded via a namespace (and not attached):
##   [1] minpack.lm_1.2-4           XML_3.99-0.16.1            rpart_4.1.23               life
##   [8] MultiAssayExperiment_1.30.2 insight_0.20.1            magrittr_2.0.3             limm
##  [15] RColorBrewer_1.1-3         ADGofTest_0.3              abind_1.4-5                zlib
##  [22] kde1d_1.0.7                rgl_1.3.1                  yulab.utils_0.1.4          prac
##  [29] tidytree_0.4.6             genefilter_1.86.0          ellipse_0.5.0              RSpe
##  [36] shapes_1.2.7               tidyselect_1.2.1           shape_1.4.6.1              UCSC
##  [43] jsonlite_1.8.8             Formula_1.2-5              survival_3.7-0             iter
##  [50] progress_1.2.3             treeio_1.28.0              ragg_1.3.2                 Rcpp
##  [57] SparseArray_1.4.8          mgcv_1.9-1                 xfun_0.44                  dist
##  [64] fansi_1.0.6                digest_0.6.35              R6_2.5.1                   text
##  [71] copula_1.1-3               flare_1.7.0.1              utf8_1.2.4                 gene
##  [78] httr_1.4.7                 htmlwidgets_1.6.4          S4Arrays_1.4.1             scat
##  [85] gtable_0.3.5               blob_1.2.4                 pcaPP_2.0-4                html
##  [92] SHT_0.1.8                  png_0.1-8                  knitr_1.47                 rstu
##  [99] stringr_1.5.1              libcoin_1.0-10             AnnotationDbi_1.66.0       pill
## [106] huge_1.3.5                   xtable_1.8-4                gamlss.dist_6.1-1
## [113] locfit_1.5-9.9               compiler_4.4.0              rlang_1.1.4
## [120] stringi_1.8.4                BiocParallel_1.38.0         nnls_1.5
## [127] lazyeval_0.2.2               glmnet_4.1-8                Matrix_1.7-0
## [134] statmod_1.5.0                highr_0.11                  rbibutils_2.2.16
## [141] ape_5.8
```

Chapter 6

Vertical Integration

In horizontal integration, we have many datasets, all with the same features. They only differ because they were gathered at different times. In contrast, for vertical integration, we instead have many datasets all with the same *samples*. They differ because they measure different aspects of those samples. Our goal in this situation is not to remove differences across datasets, like it was in horizontal integration, but instead to clarify the relationships across sources.

One important question that often arises in vertical integration is – are the data even alignable? That is, in our effort to look for relationships across datasets, we might accidentally miss out on interesting variation that exists within the individual assays. If the technologies are measuring very different things, we might be better off simply analyzing the data separately. To help us gauge which setting we might be in, we can simulate data where we know that we shouldn't align the sources. If our integration methods are giving similar outputs as they give on this simulated data, then we should be more cautious.

There are a few ways in which a dataset might not be "alignable.' ' The most general reason is that there may be no latent sources of variation in common between the sources. A simpler reason is that something that influenced one assay substantially (e.g., disease state) might not influence the other by much. Let's see how an integration method might work in this setting.

6.1 ICU Dataset

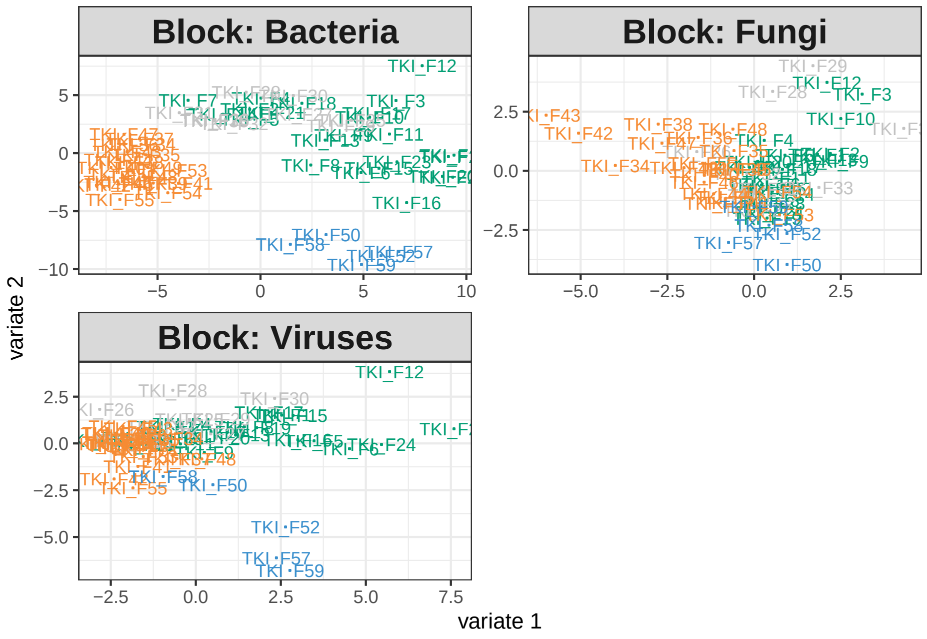
We'll work with the ICU sepsis dataset previously studied by Haak et al. (2021) and documented within a vignette for the MOFA package. The three datasets here are 16S bacterial, ITS fungal, and Virome assays, all applied to different healthy and sepsis patient populations. Moreover, some participants were on a course of antibiotics while others were not. The question is how either sepsis, antibiotics, or their interaction affects the microbiome viewed through these three assays. The data are printed below, they have already been filtered and CLR transformed following the MOFA vignette.

```
data(icu)
icu
```

```
## $Bacteria
## class: SummarizedExperiment
## dim: 180 57
## metadata(0):
## assays(1): clr
## rownames(180): Acidaminococcus Actinomyces ... Veillonellaceae Weissella
## rowData names(0):
## colnames(57): TKI_F1 TKI_F10 ... TKI_F8 TKI_F9
## colData names(16): Age Sexs ... Propionate Butyrate
##
## $Fungi
## class: SummarizedExperiment
## dim: 18 57
## metadata(0):
## assays(1): clr
## rownames(18): Agaricus Aspergillus ... Sclerotiniaceae Vishniacozyma
## rowData names(0):
## colnames(57): TKI_F1 TKI_F10 ... TKI_F8 TKI_F9
## colData names(16): Age Sexs ... Propionate Butyrate
##
## $Viruses
## class: SummarizedExperiment
## dim: 42 57
## metadata(0):
## assays(1): clr
## rownames(42): Acidovorax phage Acinetobacter phage ... Streptomyces phage Vibrio pha
## rowData names(0):
## colnames(57): TKI_F1 TKI_F10 ... TKI_F8 TKI_F9
## colData names(16): Age Sexs ... Propionate Butyrate
```

We can simultaneously analyze these data sources using block sPLS-DA. This is the multi-assay version of the analysis that we saw in the previous session. `exper_splsda` is a very light wrapper of a mixOmics function call, which you can read here. The output plot below shows that each assay differs across groups, and this is quantitatively summarized by the high estimated weights between each category and the estimated PLS directions.

```
fit <- exper_splsda(icu)
plotIndiv(fit)
```



```r
fit$weights
```

```
##               comp1     comp2
## Bacteria 0.8580402 0.8191458
## Fungi    0.6513668 0.4845247
## Viruses  0.6118091 0.8047067
```

6.2 Interlude: Using map

In the examples below, we'll find it helpful to use the function `map` in the purrr package. This function gives a one-line replacement for simple for-loops; it is analogous to list comprehensions in python. It can be useful many places besides the topic of this tutorial. For example, if we want to convert the vector `c(1, 2, 3)` into `c(1, 4, 9)`, we can use this map:

```r
map(1:3, ~ .^2)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
```

The ~ notation is shorthand for defining a function, and the . represents the current vector element. More generally, we can apply map to lists. This line will update the list so that 1 is added to each element.

```
map(list(a = 1, b = 2, c = 3), ~ . + 1)
```

```
## $a
## [1] 2
##
## $b
## [1] 3
##
## $c
## [1] 4
```

Exercise: To test your understanding, can you write a map that computes the *mean for each vector in the list `x` below? What about the mean of the 10 smallest elements?

```
x <- list(a = rnorm(100), b = rnorm(100, 1))
```

Show solution

```
map(x, mean)
```

```
## $a
## [1] 0.1177328
##
## $b
## [1] 1.128995
```

```
map(x, ~ mean(sort(.)[1:10]))
```

```
## $a
## [1] -1.920247
##
## $b
## [1] -0.8639795
```

6.3 Simulation Question

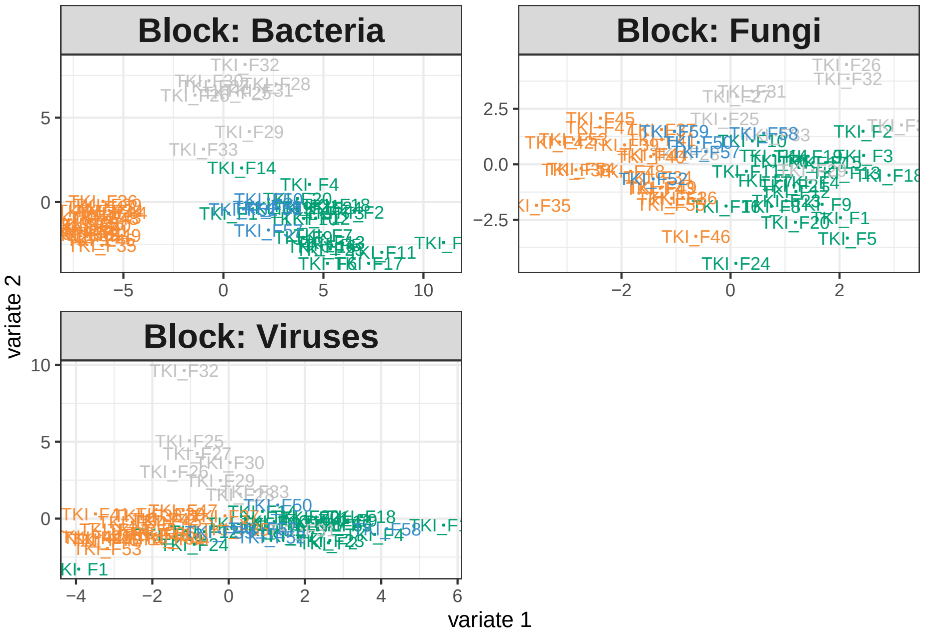
How would the output have looked if 16S community composition had not been related to disease or antibiotics groups? Since integrative analysis prioritizes similarities across sources, we expect this to mask some of the real differences in the fungal and virus data as well. We can use simulation to gauge the extent of this masking.

Our first step is to train a simulator. We're just learning four different setes of parameters for each of the four observed groups. This is not as nuanced as learning separate effects for sepsis and antibiotics, but it will be enough for illustration. We have used `map` to estimate a simulator for each assay in the `icu` list.

```
simulator <- map(
  icu,
  ~ setup_simulator(., ~Category, ~ GaussianLSS()) |>
    estimate(nu = 0.05)
)
```

So far, we haven't tried removing any relationships present in the 16S assay, and indeed our integrative analysis output on the simulated data looks comparable to that from the original study.

```
icu_sim <- join_copula(simulator, copula_adaptive()) |>
  sample() |>
  split_assays()

fit_sim <- exper_splsda(icu_sim)
plotIndiv(fit_sim)
```



```
fit$weights
```

```
##               comp1     comp2
## Bacteria 0.8580402 0.8191458
## Fungi    0.6513668 0.4845247
## Viruses  0.6118091 0.8047067
```

Exercise: Modify the simulator above so that the 16S group no longer depends on disease cateogry. This will allow us to study how the integrative analysis output changes when the data are not alignable.

```
null_simulator <- simulator
# fill this in
# null_simulator[[1]] <- ???
```

Solution: We need to define a new link that no longer depends on `Category`. One solution is to modify the existing simulator in place using `mutate`.

```
null_simulator[[1]] <- simulator[[1]] |>
  mutate(1:180, link = ~1) |>
  estimate(nu = 0.05)
```

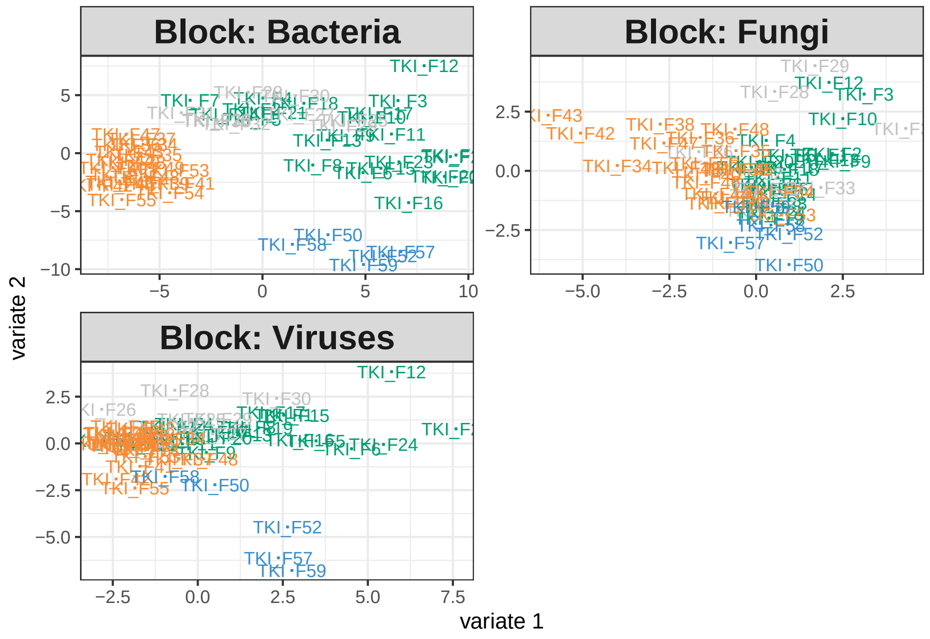
Since we are modifying all taxa, a simpler solution is to just define a new simulator from scratch.

```
null_simulator[[1]] <- setup_simulator(icu[[1]], ~1, ~ GaussianLSS()) |>
  estimate(nu = 0.05)
```

6.4 Simulation Results

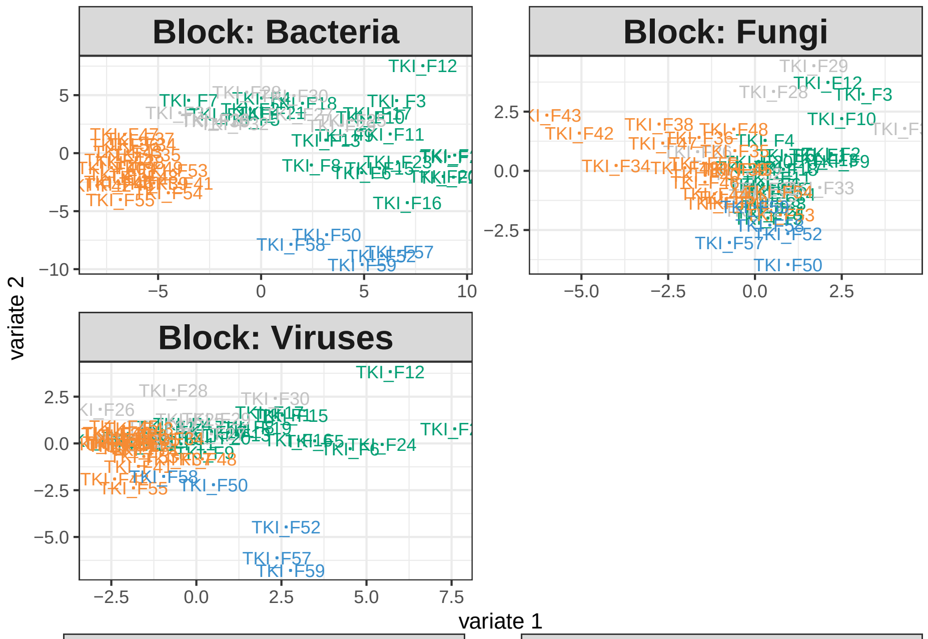
We can rerun the integrative analysis using the modified simulator. Somewhat surprisingly, the disease association in the bacteria group hasn't been erased. This is an artifact of the integration. The other assays have associations with disease group, and since the method encourages outputs across tables to be consistent with one another, we have artificially introduced some structure into the bacteria visualization (even if it is quite weak.) Nonetheless, we still observe a large dropoff in weight for the bacterial table. Further, there seems to be a minor deterioration in the group separations for the fungal and virus communities, and the component weights are higher when we work with only the fungal and virus assays. Altogether, this suggests that we may want to check table-level associations with the response variable, especially if any of the integration outputs are ambiguous. In this case, we might be able to increase power by focusing only on class-associated assays. Nonetheless, the block sPLS-DA also seems relatively robust – considering the dramatic change in the microbiome table, the output for the remaining tables still surfaces interesting relationships.

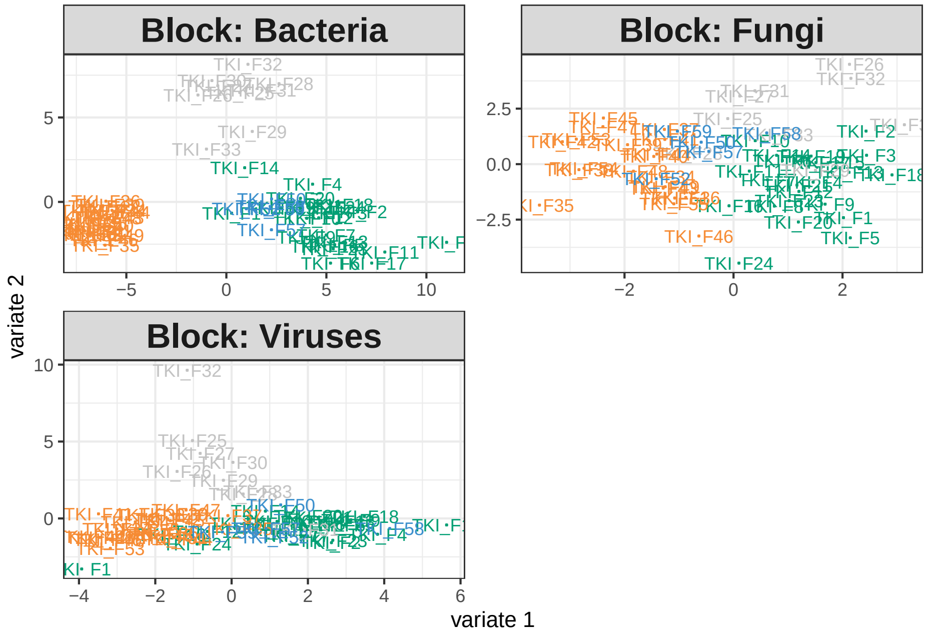
```
icu_sim <- join_copula(null_simulator, copula_adaptive()) |>
  sample() |>
  split_assays()
fit_null <- exper_splsda(icu_sim)
plotIndiv(fit)
```

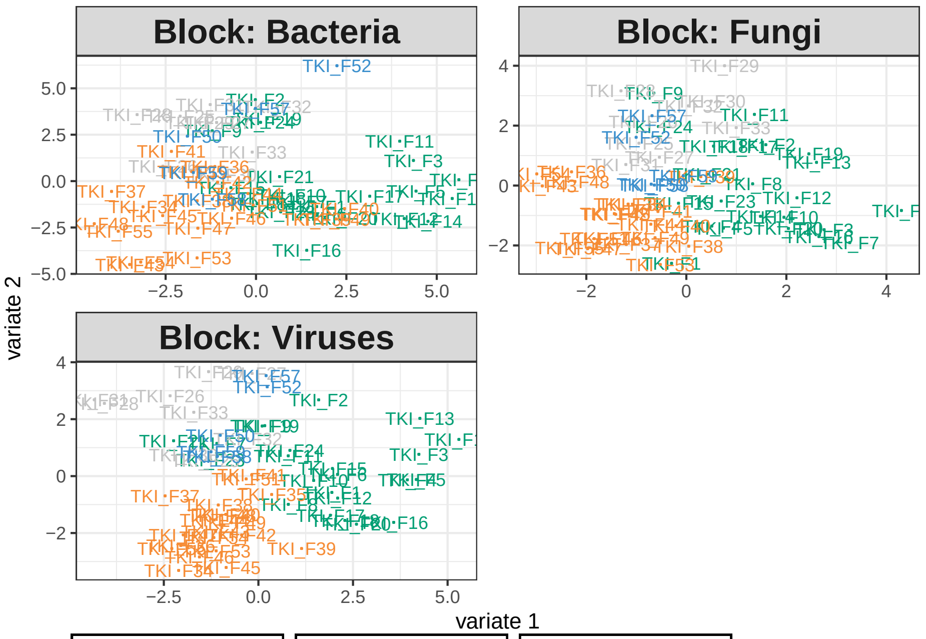


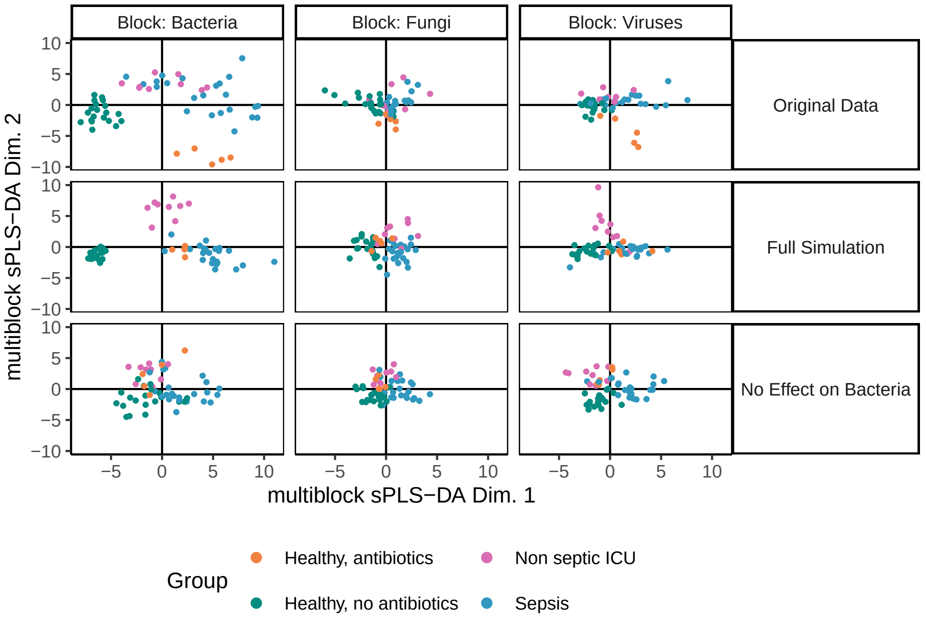
```r
fit_null$weights
```

```
##               comp1     comp2
## Bacteria 0.6658170 0.6571177
## Fungi    0.7479665 0.6797712
## Viruses  0.7015960 0.7929047
```


Block: Bacteria
Block: Fungi
Block: Viruses
variate 1
variate 2


Block: Bacteria
Block: Fungi
Block: Viruses
variate 1
variate 2





6.5 Alignability

How do the canonical correlations compare between the real data and a reference null? First, we can compute the CCA canonical correlations using the real Bacteria and Virus data.

```r
merged <- simulator[c(1, 3)] |>
  join_copula(copula_adaptive(thr = 0.01))

xy <- sample(merged) |>
  assay() |>
  t()
ix <- map(icu, rownames)
cca_result <- rcc(xy[, ix[[1]]], xy[, ix[[3]]], method = "shrinkage")
```

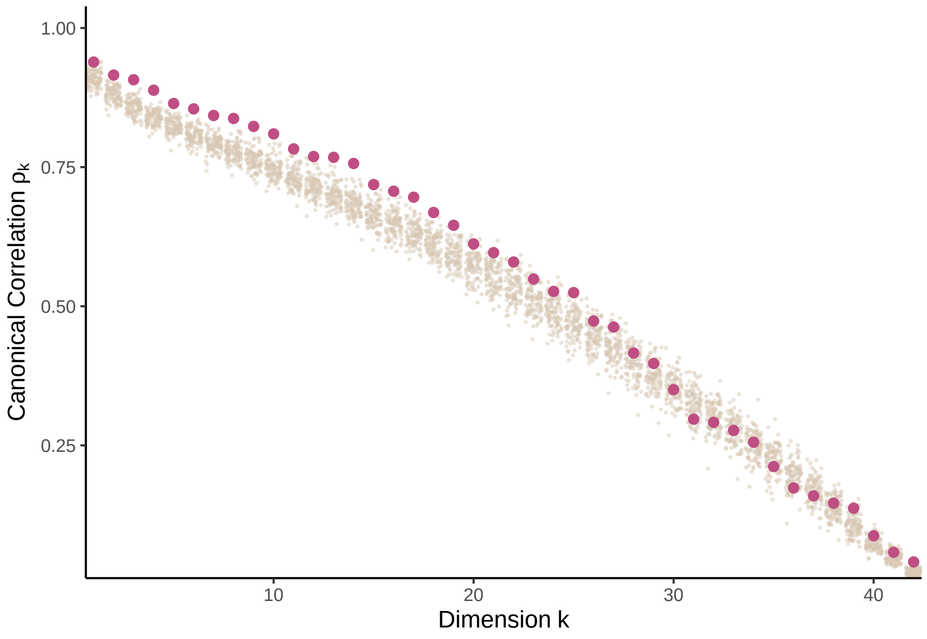
Next, we define a reference null and compute canonical correlations on ten replicates from this null. To define the reference null, we zero out any correlations across tables. We then draw 100 samples from the resulting distribution and compute canonical correlations for each.

```r
rho_null <- copula_parameters(merged)
dimnames(rho_null) <- list(colnames(xy), colnames(xy))
rho_null[ix[[1]], ix[[3]]] <- 0
rho_null[ix[[3]], ix[[1]]] <- 0

null_cancor <- merged |>
  mutate_correlation(rho_null)

B <- 100
cancors <- list()
for (b in seq_len(B)) {
  xy_null <- sample(null_cancor) |>
    assay() |>
    t()

  cancors[[glue("sim_{b}")]] <- rcc(xy_null[, ix[[1]]], xy_null[, ix[[3]]], method = "
}
```

To help with later visualization, we combine both the real and reference canonical correlations into a tidy data.frame. These are visualized in a hidden chunk below – if you want to see the source code, you can review it here.

```r
cancors[["true"]] <- cca_result$cor
contrast_data <- bind_rows(cancors, .id = "rep") |>
  pivot_longer(-rep, names_to = "loading") |>
  mutate(
    source = grepl("true", rep),
    loading = as.integer(loading)
  )
head(contrast_data)
```

```
## # A tibble: 6 x 4
##   rep   loading value source
##   <chr>   <int> <dbl> <lgl>
## 1 sim_1       1 0.913 FALSE
## 2 sim_1       2 0.880 FALSE
## 3 sim_1       3 0.848 FALSE
## 4 sim_1       4 0.826 FALSE
## 5 sim_1       5 0.815 FALSE
## 6 sim_1       6 0.797 FALSE
```



```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Australia/Melbourne
## tzcode source: internal
##
## attached base packages:
##  [1] splines   parallel  stats4    stats     graphics  grDevices utils     datasets  methods
```

```
##
## other attached packages:
##  [1] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1               XVector_0.44.0
##  [7] MIGsim_0.0.0.9000               tidyr_1.3.1                     tibble_3.2.1
## [13] lattice_0.22-6                  MASS_7.3-60.2                   glue_1.7.0
## [19] mboost_2.9-10                   stabs_0.6-4                     forcats_1.0.0
## [25] GenomicRanges_1.56.0            GenomeInfoDb_1.40.0             IRanges_2.38.0
## [31] matrixStats_1.3.0               SpiecEasi_1.1.3                 CovTools_0.5.4
##
## loaded via a namespace (and not attached):
##   [1] minpack.lm_1.2-4            XML_3.99-0.16.1             rpart_4.1.23
##   [8] MultiAssayExperiment_1.30.2 insight_0.20.1              magrittr_2.0.3
##  [15] RColorBrewer_1.1-3          ADGofTest_0.3               abind_1.4-5
##  [22] kde1d_1.0.7                 rgl_1.3.1                   yulab.utils_0.1.4
##  [29] tidytree_0.4.6              genefilter_1.86.0           ellipse_0.5.0
##  [36] shapes_1.2.7                tidyselect_1.2.1            shape_1.4.6.1
##  [43] jsonlite_1.8.8              Formula_1.2-5               survival_3.7-0
##  [50] progress_1.2.3              treeio_1.28.0               ragg_1.3.2
##  [57] SparseArray_1.4.8           mgcv_1.9-1                  xfun_0.44
##  [64] fansi_1.0.6                 digest_0.6.35               R6_2.5.1
##  [71] copula_1.1-3                flare_1.7.0.1               utf8_1.2.4
##  [78] httr_1.4.7                  htmlwidgets_1.6.4           S4Arrays_1.4.1
##  [85] gtable_0.3.5                blob_1.2.4                  pcaPP_2.0-4
##  [92] SHT_0.1.8                   png_0.1-8                   knitr_1.47
##  [99] stringr_1.5.1               libcoin_1.0-10              AnnotationDbi_1.66.0
## [106] huge_1.3.5                  xtable_1.8-4                gamlss.dist_6.1-1
## [113] locfit_1.5-9.9              compiler_4.4.0              rlang_1.1.4
## [120] stringi_1.8.4               BiocParallel_1.38.0         nnls_1.5
## [127] lazyeval_0.2.2              glmnet_4.1-8                Matrix_1.7-0
## [134] statmod_1.5.0               highr_0.11                  rbibutils_2.2.16
## [141] ape_5.8
```