

Cross species gene expression analysis using OSBF

Amal Thomas

Contents

| | | |
|-----------|---|-----------|
| 1 | Introduction | 1 |
| 2 | Species and organs | 2 |
| 3 | Load gene expression profiles | 2 |
| 3.1 | Compute mean expression profiles | 3 |
| 4 | OSBF | 4 |
| 5 | Project libraries into common space | 7 |
| 6 | Clustering in the common space | 8 |
| 7 | Explore different dimensions | 11 |
| 7.1 | 2D plots in the common space | 11 |
| 7.2 | Expression specificity and eigengene loadings | 16 |
| 7.3 | GO analysis | 18 |
| 8 | Identify tissue-specific genes | 23 |
| 8.1 | Create shuffled counts to generate null | 24 |
| 8.2 | OSBF call for shuffled counts | 25 |
| 8.3 | Identify testis-specific genes | 26 |
| 9 | Exploring dimension 1 of the common space | 28 |
| 9.1 | Identify significant genes | 28 |
| 10 | Session info | 31 |
| 11 | References | 33 |

1 Introduction

Consider a set of k real matrices $D_i \in \mathbb{R}^{m_i \times n}$, each with full column rank. Each D_i represents the mean expression of n tissues (columns) in species i with m_i genes (rows) annotated for that species. The p -th column of each D_i matrix corresponds to the average gene-expression profile of a functionally equivalent tissue representing a similar phenotype across species. The number of rows is different for each D_i matrix, but they all have the same number of columns.

In OSBF, we estimate an orthonormal basis V for the common expression space based on the inter-tissue correlations within each species. Let $X_i \in \mathbb{R}^{m_i \times n}$ be the standardized gene expression matrix for species i . We can write

$$X_i = C_i D_i S_i^{-1},$$

where

$$C_i = I_{m_i} - m_i^{-1} \mathbf{1}_{m_i} \mathbf{1}_{m_i}^T$$

is a centering matrix and $S_i = \text{diag}(s_1, \dots, s_n)$ is a diagonal scaling matrix, where s_j is the standard deviation of j -th column of D_i . Within species i , the correlation between expression profiles for the n tissues is

$$R_i = X_i^T X_i / m_i.$$

Each R_i matrix has a dimension of $n \times n$ independent of the number of genes annotated in the species. Since the corresponding column of each D_i matrix represents a similar phenotype, we can define an expected correlation matrix across the species:

$$E = \mathbb{E}(R) = \frac{1}{k} \sum_{i=1}^k R_i.$$

The shared right basis matrix V represents the common expression space and is determined from the eigenvalue decomposition of E , where $E = V \Lambda V^T$. The right basis matrix V is identical in all the k matrix factorizations, and the different dimensions of V represent the correlation relationship between tissues independent of the fact to which species tissues belong. Once the V is estimated, the species-specific left basis matrix with orthonormal columns U_i (eigengenes) and the Δ_i that minimize the factorization error are computed.

```
# load SBF package
library(SBF)
```

Additional packages required for the workflow

```
# install packages
pkgs <- c("data.table", "dplyr", "matrixStats")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
suppressPackageStartupMessages({
  library(data.table)
  library(dplyr)
  library(matrixStats)
})
```

2 Species and organs

In this workflow, we will work with gene expression profile of six tissues from eight species.

```
species <- c("Homo_sapiens", "Pan_troglodytes", "Macaca_mulatta",
           "Mus_musculus", "Rattus_norvegicus", "Bos_taurus",
           "Sus_scrofa", "Gallus_gallus")
species_short <- sapply(species, getSpeciesShortName)
species_short
#>   Homo_sapiens  Pan_troglodytes  Macaca_mulatta  Mus_musculus
#>   "hsapiens"    "ptroglodytes"    "mmulatta"      "mmusculus"
#>   Rattus_norvegicus      Bos_taurus      Sus_scrofa      Gallus_gallus
#>   "rnorvegicus"    "btaurus"      "sscrofa"      "ggallus"
# common tissues present in all 8 species
tissues <- c("brain", "heart", "kidney", "liver", "lung", "testis")
```

3 Load gene expression profiles

Download the processed RNA-Seq counts file (“counts.tar.gz”) from <https://figshare.com/s/4d8d5cf9c1362bcbe90d> Uncompress the .tar.gz file and add it to the working directory.

```

# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
counts_list <- metadata_list <- list()
require(data.table)
for (sp in species) {
  # read logTPM counts for each species
  counts <- data.table::fread(paste0(path, "counts/", sp, "_logTPM.tsv"),
                               sep = "\t", header = TRUE, data.table = FALSE,
                               nThread = 4)
  row.names(counts) <- counts$V1
  counts$V1 <- NULL
  col_fields <- data.table::tstrsplit(colnames(counts), "_")
  metadata <- data.frame(
    project = col_fields[[1]],
    species = col_fields[[2]],
    tissue = col_fields[[3]],
    gsm = col_fields[[4]],
    name = colnames(counts),
    stringsAsFactors = FALSE)
  metadata_sel <- metadata[metadata$tissue %in% tissues, , drop = FALSE]
  counts_sel <- counts[, colnames(counts) %in% metadata_sel$name, drop = FALSE]
  metadata_sel$ref <- seq_len(nrow(metadata_sel))
  metadata_sel$key <- paste0(metadata_sel$species, "_", metadata_sel$tissue)

  counts_list[[sp]] <- counts_sel
  metadata_list[[sp]] <- metadata_sel
}

```

The dimensions and the number of RNA-Seq profiles for different species.

```

sapply(counts_list, dim)
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus Rattus_norvegicus
#> [1,]      58676          31373        30807       54446        32623
#> [2,]        111            61          124         107          64
#>      Bos_taurus Sus_scrofa Gallus_gallus
#> [1,]     24596        25880        19152
#> [2,]       53           165          117

```

3.1 Compute mean expression profiles

Now, for each species, let us compute the mean expression profile for each tissue. We will use `calcAvgCounts` function from the `SBF` package.

```

avg_counts <- list()
for (sp in species) {
  avg_counts[[sp]] <- calcAvgCounts(counts_list[[sp]],
                                      metadata_list[[sp]])
}

# check tissue columns are matching in each species
c_tissues <- as.data.frame(sapply(avg_counts, function(x) {
  data.table::tstrsplit(colnames(x), "_")[[2]]
}))
if (!all(apply(c_tissues, 1, function(x) all(x == x[1])))) {
  stop("Error! tissues not matching")
}

```

The dimension of mean expression profiles

```
sapply(avg_counts, dim)
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus Rattus_norvegicus
#> [1,]      58676           31373       30807      54446       32623
#> [2,]          6              6          6          6          6
#>      Bos_taurus Sus_scrofa Gallus_gallus
#> [1,]     24596        25880       19152
#> [2,]          6              6          6
```

Remove genes not expressed.

```
# remove empty rows
removeZeros <- function(df) {
  return(df[rowSums(df) > 0, ])
}

avg_counts <- lapply(avg_counts, removeZeros)
sapply(avg_counts, dim)
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus Rattus_norvegicus
#> [1,]      47301           26173       27762      39097       23004
#> [2,]          6              6          6          6          6
#>      Bos_taurus Sus_scrofa Gallus_gallus
#> [1,]     20028        21448       17810
#> [2,]          6              6          6
# update counts_list
counts_list_sub <- list()
for (sp in names(avg_counts)) {
  counts_list_sub[[sp]] <- counts_list[[sp]][row.names(avg_counts[[sp]]), ,
                                             drop = FALSE]
}
```

4 OSBF

We will perform OSBF in two ways.

1. Keeping the initial estimate of V the same while updating U_i and Δ_i to minimize the factorization error. By keeping the V same, the initial V estimated based inter-sample correlation is maintained.
2. Update V , U_i and Δ_i to minimize the factorization error.

```
cat("\nOSBF with optimizing V = FALSE started\n")
#>
#> OSBF with optimizing V = FALSE started
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:06:11 PM 2022
t1 <- proc.time()
# decrease tol to minimize error
osbf_noVupdate <- SBF(avg_counts, transform_matrix = TRUE, orthogonal = TRUE,
                      optimizeV = FALSE, tol = 1e-3)#, tol = 1e-10)
#>
#> OSBF optimizing factorization error
t2 <- proc.time()
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:06:12 PM 2022
cat("OSBF with optimizing V = FALSE finished\n")
#> OSBF with optimizing V = FALSE finished
```

```

cat("Time taken:\n")
#> Time taken:
t2 - t1
#>    user  system elapsed
#> 0.541   0.060   0.600

cat("\nOSBF with optimizing V=TRUE started\n")
#>
#> OSBF with optimizing V=TRUE started
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:06:12 PM 2022
t1 <- proc.time()
# decrease tol to minimize error
osbf <- SBF(avg_counts, transform_matrix = TRUE, orthogonal = TRUE,
            optimizeV = TRUE, tol = 1e-3) #, tol = 1e-10)
#>
#> OSBF optimizing factorization error
t2 <- proc.time()
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:06:27 PM 2022
cat("OSBF with optimizing V=TRUE finished\n")
#> OSBF with optimizing V=TRUE finished
cat("Time taken:\n")
#> Time taken:
t2 - t1
#>    user  system elapsed
#> 15.046   0.053  15.099

```

Note: We use tolerance threshold = 1e-3 for fast computation. Reduce the tolerance threshold (default value 1e-10) to minimize the factorization error further. Lowering the threshold value increases the computing time.

The final factorization error and number of updates taken:

```

cat("\n", sprintf("%-27s:", "Final error [No V update]"), sprintf("%16.2f",
                                                               osbf_noVupdate$error))
#>
#> Final error [No V update] : 219507.66
cat("\n", sprintf("%-27s:", "Final error [With V update]"), sprintf("%16.2f",
                                                               osbf$error))
#>
#> Final error [With V update]: 23483.37
cat("\n", sprintf("%-27s:", "# of update [No V update]"), sprintf("%16d",
                                                               osbf_noVupdate$error_pos))
#>
#> # of update [No V update] : 5
cat("\n", sprintf("%-27s:", "# of update [With V update]"), sprintf("%16d",
                                                               osbf$error_pos))
#>
#> # of update [With V update]: 274

```

Optimization with updating V achieves a lower decomposition error.

```

osbf_noVupdate$error / osbf$error
#> [1] 9.347368

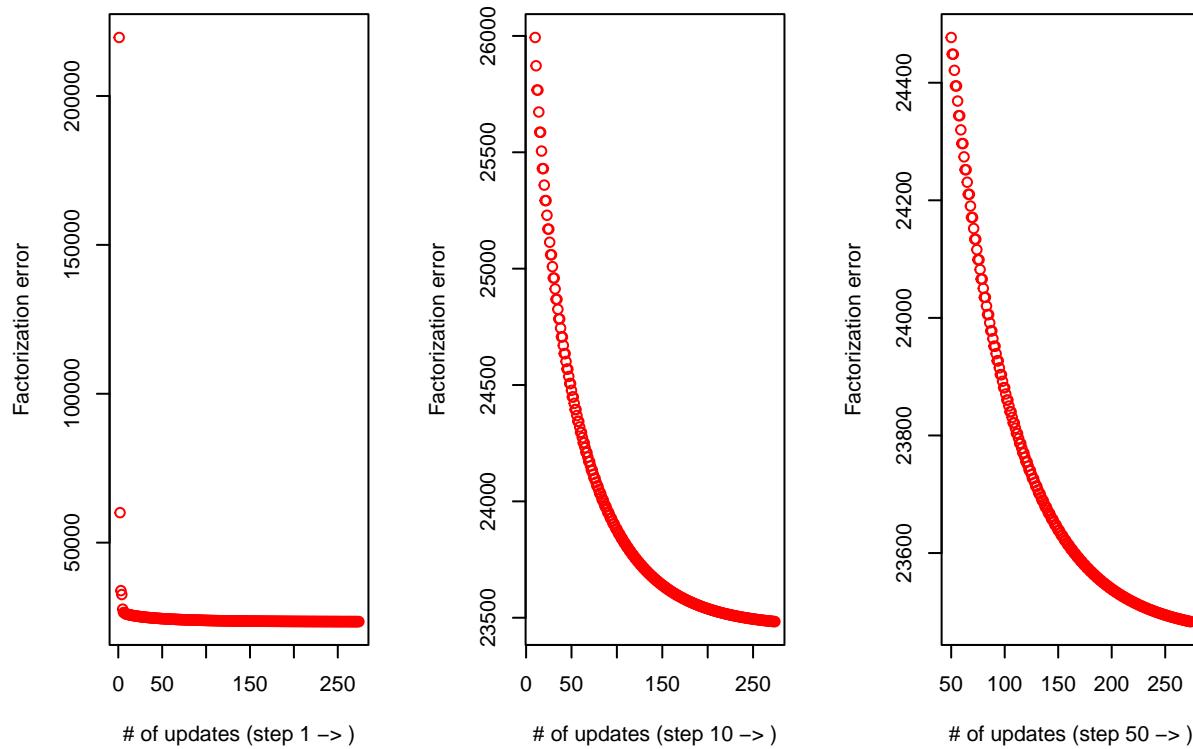
```

Let us plot the decomposition error vs. updates.

```

par(mfrow = c(1, 3))
plot(x = seq_len(length(osbf$error_vec)), y = osbf$error_vec,
      xlab = "# of updates (step 1 -> )",
      ylab = "Factorization error", col = "red")
plot(x = 10:length(osbf$error_vec),
      y = osbf$error_vec[10:length(osbf$error_vec)],
      xlab = "# of updates (step 10 -> )",
      ylab = "Factorization error", col = "red")
plot(x = 50:length(osbf$error_vec),
      y = osbf$error_vec[50:length(osbf$error_vec)],
      xlab = "# of updates (step 50 -> )",
      ylab = "Factorization error", col = "red")

```



Percentage of information (p_{ij}) represented by a common space dimension is defined as $p_{ij} = \delta_{ij}^2 / \sum_{j=1}^6 \delta_{ij}^2 \times 100$, where $\Delta_i = \text{diag}(\delta_{i1}, \dots, \delta_{i6})$

```

cat("\nPercentage for each delta [No V update]:")
#>
#> Percentage for each delta [No V update]:
percentInfo_noVupdate <- calcPercentInfo(osbf_noVupdate)
for (i in names(osbf_noVupdate$delta)) {
  cat("\n", sprintf("%-25s:", i), sprintf("%8.2f", percentInfo_noVupdate[[i]]))
}
#>
#>   Homo_sapiens       :  86.56    5.52    3.01    2.03    1.62    1.26
#>   Pan_troglodytes   :  88.57    4.87    2.61    1.71    1.30    0.95
#>   Macaca_mulatta   :  87.44    5.09    2.99    1.75    1.47    1.26
#>   Mus_musculus      :  81.30    8.11    4.56    2.64    1.73    1.66
#>   Rattus_norvegicus :  83.84    6.53    3.96    2.40    1.77    1.49
#>   Bos_taurus        :  86.99    5.19    2.94    1.85    1.77    1.26

```

```

#> Sus_scrofa      :   85.28    5.38    3.52    2.31    2.00    1.51
#> Gallus_gallus  :   87.70    4.49    2.93    1.90    1.65    1.33
percentInfo <- calcPercentInfo(osbf)
for (i in names(osbf$delta)) {
  cat("\n", sprintf("%-25s:", i), sprintf("%8.2f", percentInfo[[i]]))
}
#>
#> Homo_sapiens    :   87.23    5.05    2.99    2.13    1.41    1.19
#> Pan_troglodytes :   88.86    4.64    2.70    1.60    1.24    0.96
#> Macaca_mulatta :   87.77    4.87    2.98    1.79    1.36    1.22
#> Mus_musculus    :   81.64    7.99    4.43    2.67    1.67    1.59
#> Rattus_norvegicus:   84.10    6.48    3.86    2.51    1.62    1.44
#> Bos_taurus      :   87.37    4.94    2.96    1.78    1.71    1.24
#> Sus_scrofa      :   85.53    5.39    3.38    2.41    1.82    1.47
#> Gallus_gallus  :   88.21    4.15    2.98    1.86    1.56    1.24

```

The percentage of information represented by different dimensions of the two approaches looks very similar.

5 Project libraries into common space

Project individual profiles and average counts to common space by computing $D_i^T U_i \Delta^{-1}$. We will projectCounts function from the SBF package for this.

```

# project profiles using no V update estimates
# we can project both mean expression profiles as well as individual expression
# profiles
df_proj_avg_noVupdate <- projectCounts(avg_counts, osbf_noVupdate)
meta <- data.table::tstrsplit(row.names(df_proj_avg_noVupdate), "_")
df_proj_avg_noVupdate$tissue <- factor(meta[[2]])
df_proj_avg_noVupdate$species <- factor(meta[[1]])
df_proj_avg_noVupdate <- df_proj_avg_noVupdate %>%
  mutate(species = factor(species, levels = species_short))

df_proj_noVupdate <- projectCounts(counts_list_sub, osbf_noVupdate)
meta1 <- data.table::tstrsplit(row.names(df_proj_noVupdate), "_")
df_proj_noVupdate$tissue <- factor(meta1[[3]])
df_proj_noVupdate$species <- factor(meta1[[2]])
df_proj_noVupdate <- df_proj_noVupdate %>% mutate(species = factor(species,
  levels = species_short))

```

Now let us also project profiles with the updated V

```

# project using V update estimates
df_proj_avg <- projectCounts(avg_counts, osbf)
meta <- data.table::tstrsplit(row.names(df_proj_avg), "_")
df_proj_avg$tissue <- factor(meta[[2]])
df_proj_avg$species <- factor(meta[[1]])
df_proj_avg <- df_proj_avg %>% mutate(species = factor(species,
  levels = species_short))

df_proj <- projectCounts(counts_list_sub, osbf)
meta1 <- data.table::tstrsplit(row.names(df_proj), "_")

```

```

df_proj$tissue <- factor(meta1[[3]])
df_proj$species <- factor(meta1[[2]])
df_proj <- df_proj %>% mutate(species = factor(species,
                                                 levels = species_short))

```

6 Clustering in the common space

```

# install packages
pkgs <- c("RColorBrewer")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
pkgs <- c("ComplexHeatmap")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install)) {
  if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
  BiocManager::install("ComplexHeatmap")
}
suppressPackageStartupMessages({
  library(ComplexHeatmap)
  library(RColorBrewer)
})

```

Compute distances between projected profiles and perform clustering.

```

data_noVupdate <- df_proj_noVupdate
data_noVupdate$tissue <- NULL
data_noVupdate$species <- NULL
data_noVupdate <- as.matrix(data_noVupdate)
data_noVupdate_dist <- as.matrix(dist(data_noVupdate, method = "euclidean"))
meta <- data.table:::tstrsplit(colnames(data_noVupdate_dist), "_")
ht <- ComplexHeatmap::HeatmapAnnotation(tissue = meta[[3]], species = meta[[2]],
                                         col = list(tissue = c("brain" = "#1B9E77",
                                                               "heart" = "#D95F02",
                                                               "kidney" = "#7570B3",
                                                               "liver" = "#E7298A",
                                                               "lung" = "#66A61E",
                                                               "testis" = "#E6AB02"),
                                         species = c("hsapiens" = "#66C2A5",
                                                               "ptroglodytes" = "#FC8D62",
                                                               "mmulatta" = "#8DA0CB",
                                                               "mmusculus" = "#E78AC3",
                                                               "rnorvegicus" = "#A6D854",
                                                               "btaurus" = "#FFD92F",
                                                               "sscrofa" = "#E5C494",
                                                               "ggallus" = "#B3B3B3")),
                                         #show_annotation_name = F,
                                         annotation_name_side = "left")
mypalette <- RColorBrewer::brewer.pal(9, "Blues")
morecolors <- colorRampPalette(mypalette)

myheatmap1 <- ComplexHeatmap::Heatmap(as.matrix(data_noVupdate_dist),

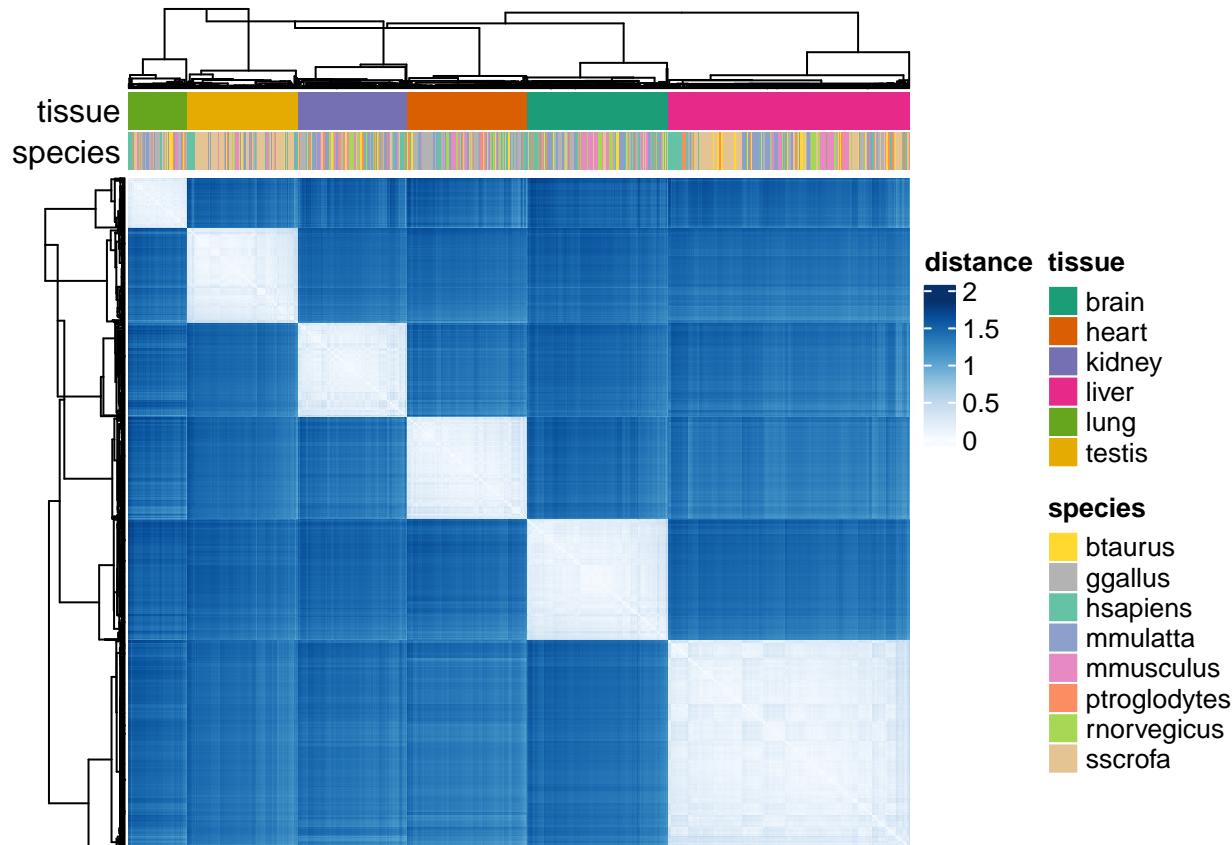
```

```

    cluster_rows = TRUE,
    clustering_method_rows = "centroid",
    cluster_columns = TRUE,
    clustering_method_columns = "centroid",
    top_annotation = ht, col = morecolors(50),
    show_row_names = FALSE,
    show_column_names = FALSE,
    name = "distance")

```

myheatmap1



Compute distances between projected profiles in the common space estimated using optimized V and perform clustering.

```

data <- df_proj
data$tissue <- NULL
data$species <- NULL
data <- as.matrix(data)
data_dist <- as.matrix(dist(data, method = "euclidean"))
meta <- data.table:::strsplit(colnames(data_dist), "_")
ht <- ComplexHeatmap::HeatmapAnnotation(tissue = meta[[3]], species = meta[[2]],
                                         col = list(tissue = c("brain" = "#1B9E77",
                                                               "heart" = "#D95F02",
                                                               "kidney" = "#7570B3",
                                                               "liver" = "#E7298A",
                                                               "lung" = "#66A61E",
                                                               "testis" = "#E6AB02"),
                                         species = c("hsapiens" = "#66C2A5",

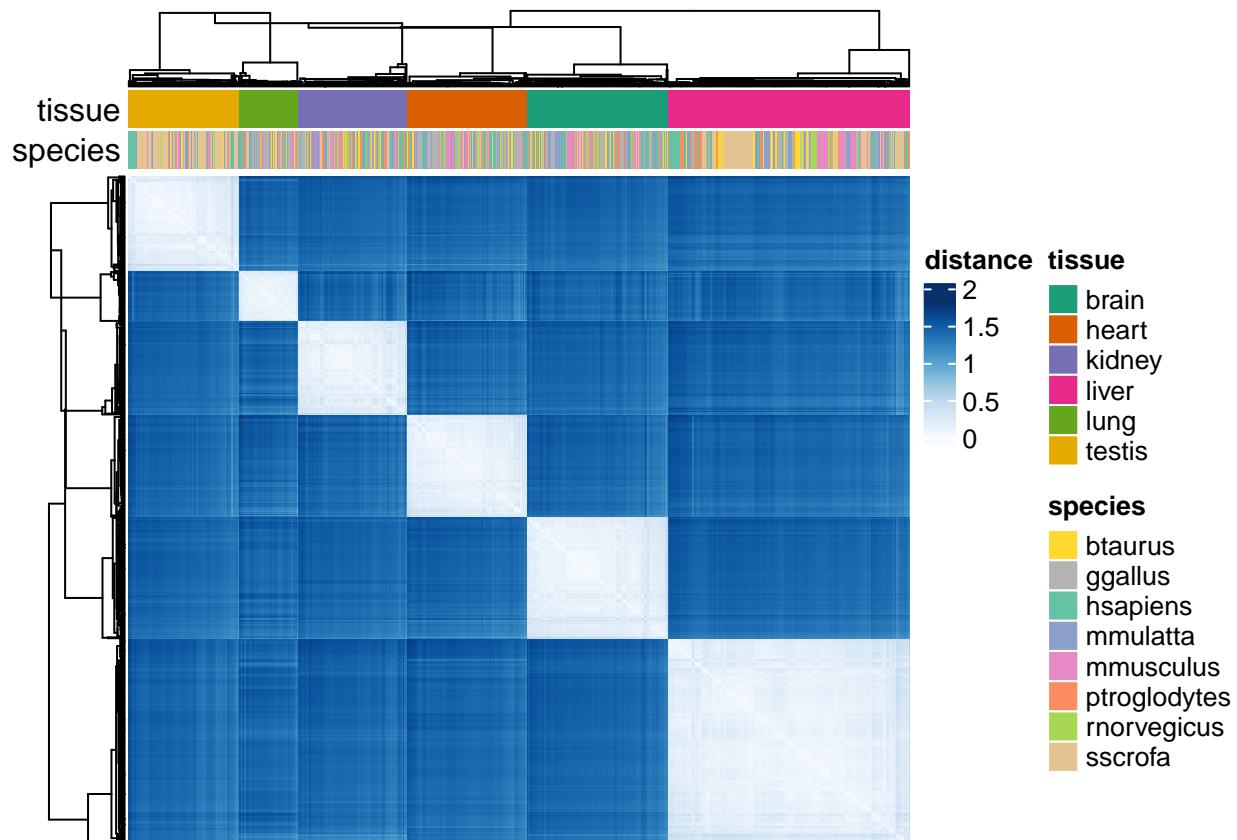
```

```

"ptroglodytes" = "#FC8D62",
"mmulatta" = "#8DA0CB",
"mmusculus" = "#E78AC3",
"rnorvegicus" = "#A6D854",
"btaurus" = "#FFD92F",
"sscrofa" = "#E5C494",
"ggallus" = "#B3B3B3")),
#show_annotation_name = F,
annotation_name_side = "left")
mypalette <- RColorBrewer::brewer.pal(9, "Blues")
morecolors <- colorRampPalette(mypalette)

myheatmap <- ComplexHeatmap::Heatmap(as.matrix(data_dist),
cluster_rows = TRUE,
clustering_method_rows = "centroid",
cluster_columns = TRUE,
clustering_method_columns = "centroid",
top_annotation = ht, col = morecolors(50),
show_row_names = FALSE,
show_column_names = FALSE,
name = "distance")
myheatmap

```



Gene expression profiles cluster by tissue type independent of the species of origin.

7 Explore different dimensions

7.1 2D plots in the common space

Next, we will explore the 2D projection plots in the common space. We will first define a custom theme that we will use for the plots.

```
# install packages
pkgs <- c("grid", "ggthemes", "ggplot2")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
suppressPackageStartupMessages({
  library(grid)
  library(ggthemes)
  library(ggplot2)
})
```

We will use the following custom theme for the ggplots.

```
# custom theme function for ggplot2
customTheme <- function(base_size = 10, base_family = "helvetica") {
  require(grid)
  require(ggthemes)
  (ggthemes::theme.foundation(base_size = base_size)
  + ggplot2::theme(plot.title = element_text(face = "bold",
                                              size = rel(1.2), hjust = 0.5),
                  text = element_text(),
                  panel.background = element_rect(colour = NA),
                  plot.background = element_rect(colour = NA),
                  panel.border = element_rect(colour = NA),
                  axis.title = element_text(size = rel(1)),
                  axis.title.y = element_text(angle = 90, vjust = 2),
                  axis.title.x = element_text(vjust = -0.2),
                  axis.text = element_text(),
                  axis.line = element_line(colour = "black"),
                  axis.ticks = element_line(),
                  panel.grid.major = element_blank(),
                  panel.grid.minor = element_blank(),
                  legend.key = element_rect(colour = NA),
                  legend.position = "top",
                  legend.direction = "horizontal",
                  legend.key.size = unit(0.2, "cm"),
                  legend.spacing = unit(0, "cm"),
                  legend.title = element_text(face = "italic"),
                  plot.margin = unit(c(10, 5, 5, 5), "mm"),
                  strip.background = element_rect(colour = "#f0f0f0", fill = "#f0f0f0"),
                  strip.text = element_text(face = "bold")))
}
```

Let us first check the projected libraries in dimension 1 and 2.

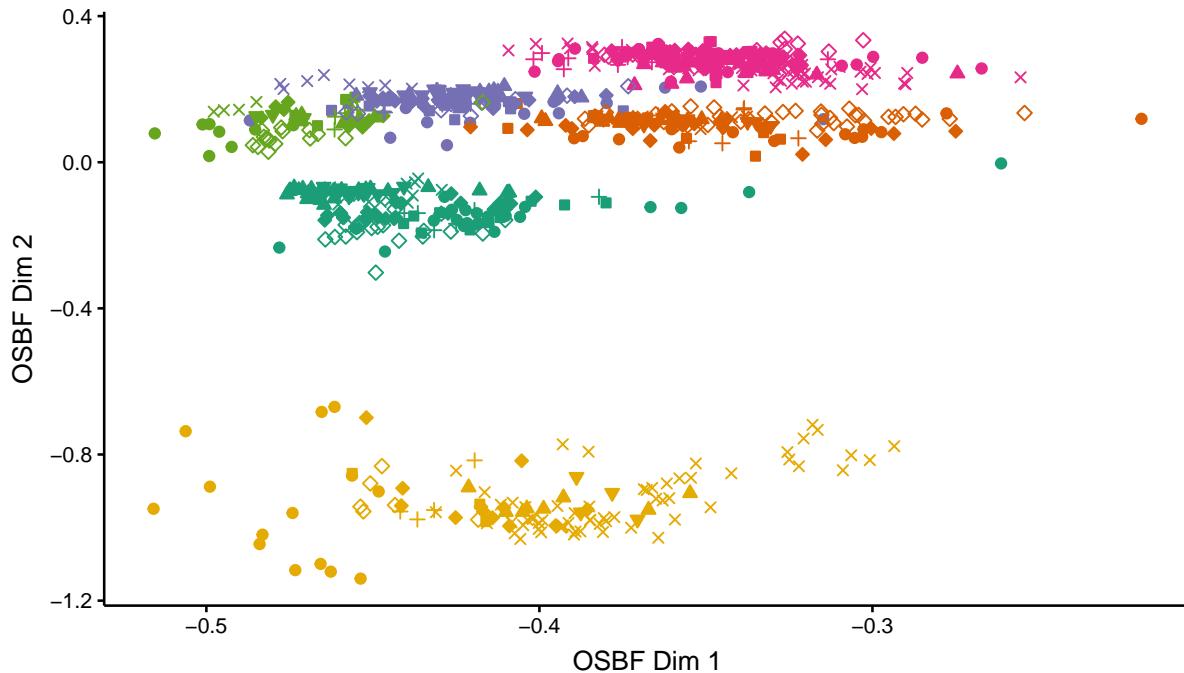
```
# 2D plot for Dim1 and Dim2 [No V update]
sel_colors <- RColorBrewer::brewer.pal(8, "Dark2") [seq_len(length(unique(df_proj_noVupdate$tissue)))]
i <- 1
j <- 2
ggplot2::ggplot(df_proj_noVupdate, aes(x = df_proj_noVupdate[, i],
```

```

y = df_proj_noUpdate[, j], col = tissue,
shape = species, fill = tissue)) +
xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
scale_shape_manual(values = c(21:25, 3:7)) +
scale_fill_manual(values = sel_colors) +
customTheme() +
theme(legend.title = element_blank())

```

● brain ● kidney ● lung ○ hsapiens ◇ mmulatta ▽ rnorvegicus × sscrofa
● heart ● liver ● testis □ ptroglydotes △ mmusculus + btaurus ◇ ggallus

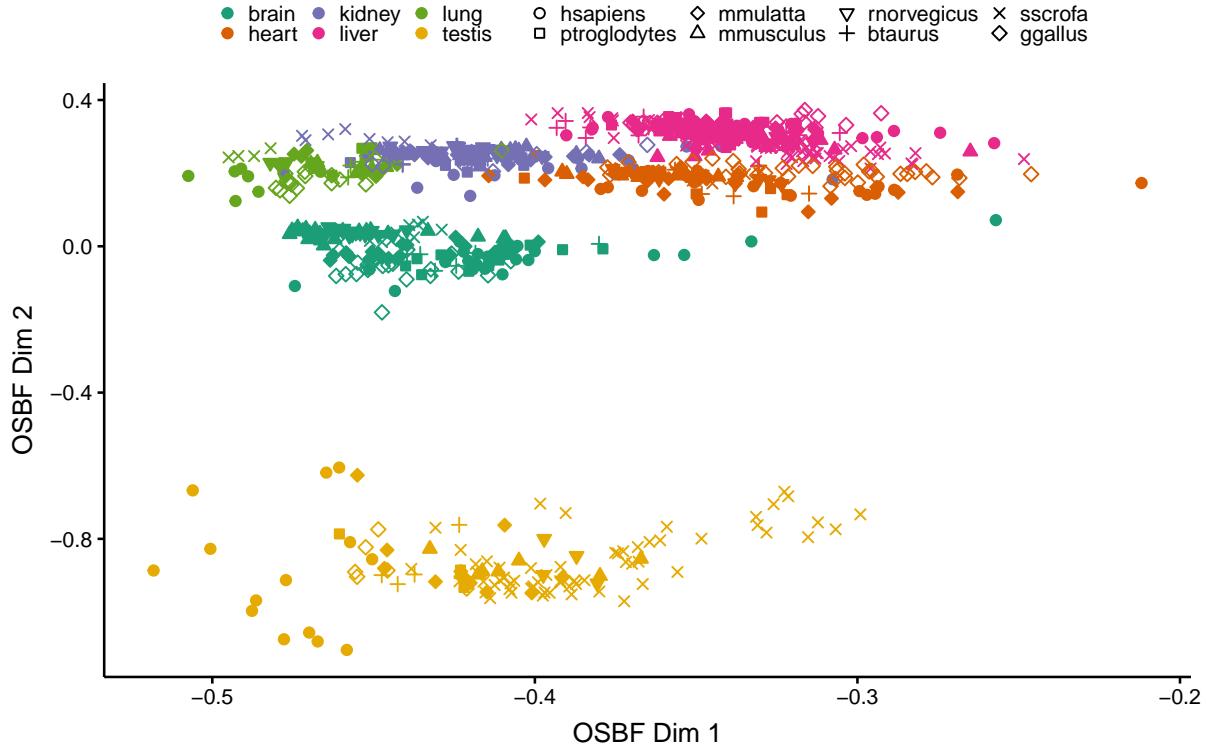


The same with optimized V

```

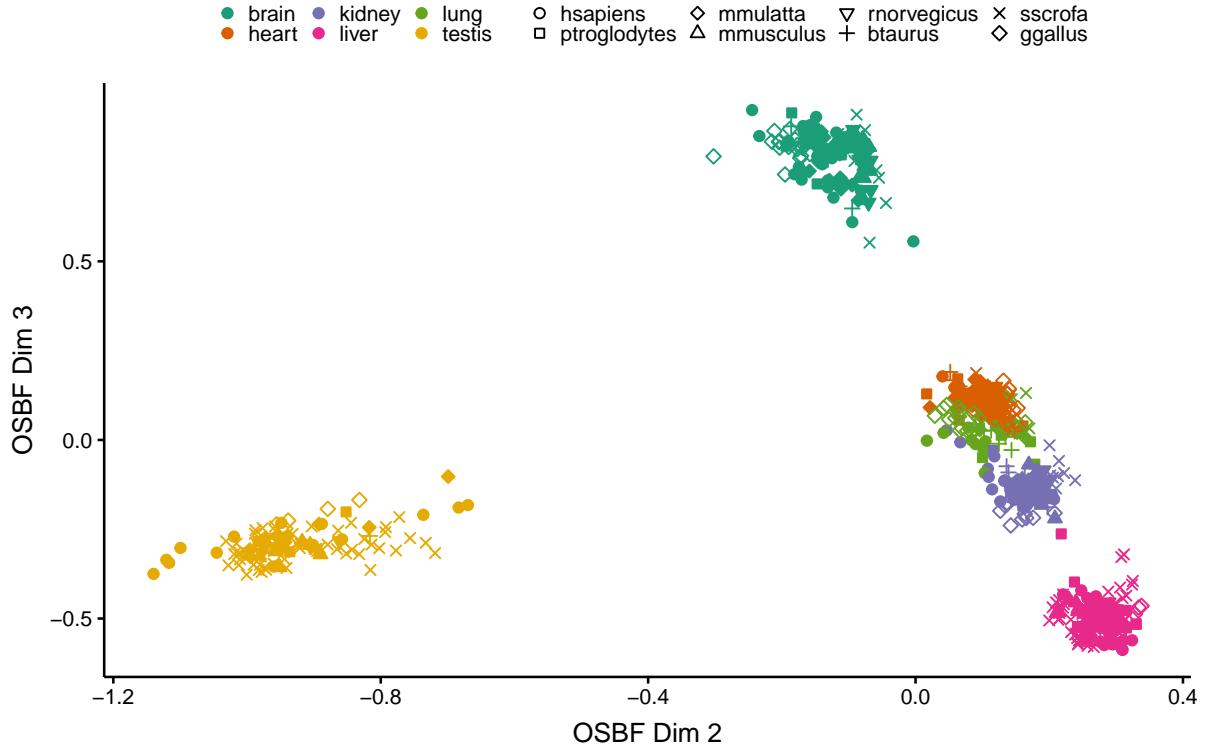
# 2D plot for Dim1 and Dim2 [With V update]
i <- 1
j <- 2
ggplot2::ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j], col = tissue,
                               shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())

```

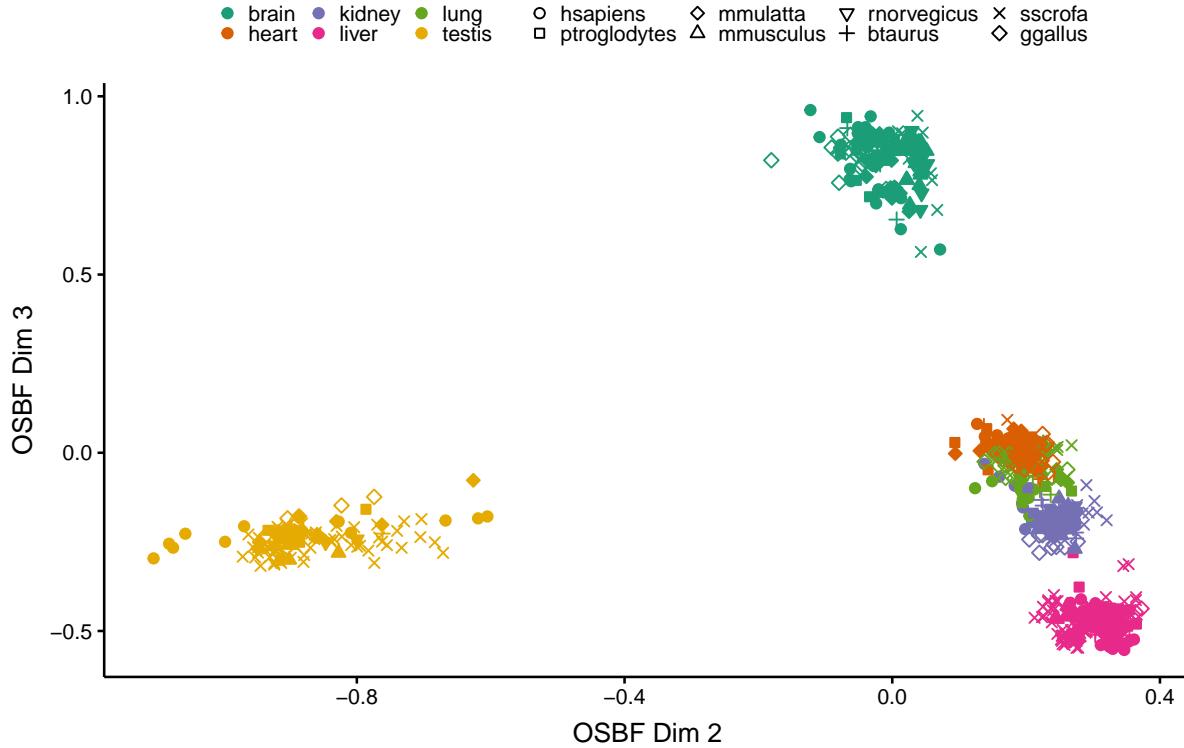


Let us plot the projected libraries in dimensions 2 and 3.

```
# 2D plot for Dim2 and Dim3 [No V update]
i <- 2
j <- 3
ggplot2::ggplot(df_proj_noVupdate, aes(x = df_proj_noVupdate[, i],
                                         y = df_proj_noVupdate[, j], col = tissue,
                                         shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```



```
# 2D plot for Dim2 and Dim3 [With V update]
i <- 2
j <- 3
ggplot2::ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j], col = tissue,
                               shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```



Save all 2D plots

```

sel_colors <- c("#1B9E77", "#D95F02", "#7570B3", "#E7298A", "mediumturquoise",
                 "#E6AB02", "darkmagenta", "#666666", "black", "darkolivegreen2")
outputname <- "human_mouse_blood"
finished <- c()
for (i in 1:(ncol(df_proj) - 2)) {
  for (j in 1:(ncol(df_proj) - 2)) {
    if (i == j) next
    if (j %in% finished) next
    ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j],
                          col = tissue, shape = species, fill = tissue)) +
      xlab(paste0("OSBF Dim ", i)) +
      ylab(paste0("OSBF Dim ", j)) +
      geom_point(size = 1.5) +
      scale_color_manual(values = sel_colors) +
      scale_shape_manual(values = c(21:25, 3:7)) +
      scale_fill_manual(values = sel_colors) +
      customTheme(base_size = 12) +
      theme(legend.title = element_blank())
    #ggsave(filename = paste0(outdir, "2Dplots/opt_2Dplot_Dim_", i, "-", j, "_",
    #                     outputname, ".pdf"), device = "pdf",
    #                     width = 7, height = 7, useDingbats = FALSE)
  }
  finished <- c(finished, i)
}

```

Similarly, we can check for other dimensions of the common space. We observe that both clustering and 2D projections plots in the common space with optimized V are similar to those without V update. So we will use the common space with optimized V for future analysis.

7.2 Expression specificity and eigengene loadings

Functions to compute tissue specificity (τ) and scaled average expression profile.

```
# function to compute Tau
calc_tissue_specificity <- function(a) {
  a <- as.matrix(a)
  b <- a / matrixStats::rowMaxs(a)
  return(rowSums(1 - b) / (ncol(b) - 1))
}

Tau <- lapply(avg_counts, function(x) { calc_tissue_specificity(x)})
avg_counts_scaled <- lapply(avg_counts, function(x) { t(scale(t(x)))})

combine_expr <- list()
for (sp in names(avg_counts_scaled)) {
  x <- as.data.frame(avg_counts_scaled[[sp]])
  x[["Tau"]] <- Tau[[sp]]
  combine_expr[[sp]] <- x
}
```

From the 2D projection plot, we find that along the dimension 2, testis profiles lie on the negative axis and separate from the rest of the tissues. We will plot the relationship between U_i loadings in dimension 2 and the testis expression.

```
sel_dim <- 2
sel_tissue <- "testis"
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
head(expr1)
#>           tissue_zscore      Tau       coef
#> ENSG000000000003  1.15812809 0.3824796 -0.0027709444
#> ENSG000000000005 -0.05534986 0.8531503  0.0002702072
#> ENSG000000000419  1.43944024 0.1848732  0.0002641079
#> ENSG000000000457  1.15751198 0.3444267 -0.0017597439
#> ENSG000000000460  1.95491402 0.8630708 -0.0106484590
#> ENSG000000000938 -0.54023331 0.6759041  0.0076031503
```

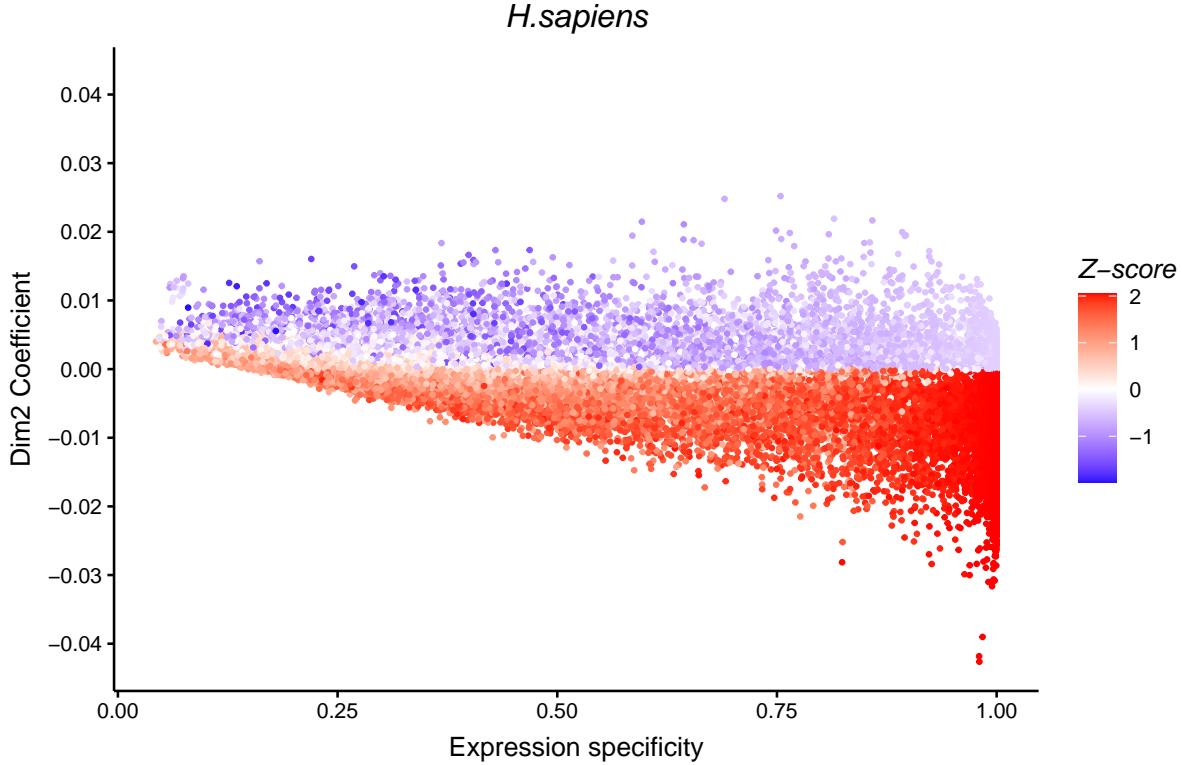
Dimension 2 U_i loadings vs expression specificity (τ) for humans

```
# plot scatter
mid <- 0
p1 <- ggplot2::ggplot(expr1, aes(x = Tau, y = coef, col = tissue_zscore)) +
  theme_bw() +
  geom_point(size = 0.5) + xlab("Expression specificity") +
  ylab(paste0("Dim", sel_dim, " Coefficient")) +
  scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                        high = "red", space = "Lab") +
  scale_y_continuous(limits = c(-1 * max(abs(expr1$coef)),
                                max(abs(expr1$coef))),
                     breaks = seq(-1 * round(max(abs(expr$coef)), 2),
                                  round(max(abs(expr$coef)), 2), by = 0.01)) +
  customTheme() + theme(legend.position = "right",
```

```

        legend.direction = "vertical") +
  labs(title = paste0(getScientificName(species)), color = "Z-score") +
  theme(legend.key.size = unit(0.5, "cm"),
        plot.title = element_text(face = "italic"))
p1

```



Dimension 2 U_i loadings vs expression specificity (τ) for pig.

```

sel_dim <- 2
sel_tissue <- "testis"
species <- "Sus_scrofa"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")

# plot scatter
mid <- 0
p2 <- ggplot2::ggplot(expr1, aes(x = Tau, y = coef, col = tissue_zscore)) +
  theme_bw() +
  geom_point(size = 0.5) + xlab("Expression specificity") +
  ylab(paste0("Dim", sel_dim, " Coefficient")) +
  scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                        high = "red", space = "Lab") +
  scale_y_continuous(limits = c(-1 * max(abs(expr1$coef)),
                                max(abs(expr1$coef))),
                     breaks = seq(-1 * round(max(abs(expr$coef))), 2,
                                round(max(abs(expr$coef)), 2), by = 0.01)) +

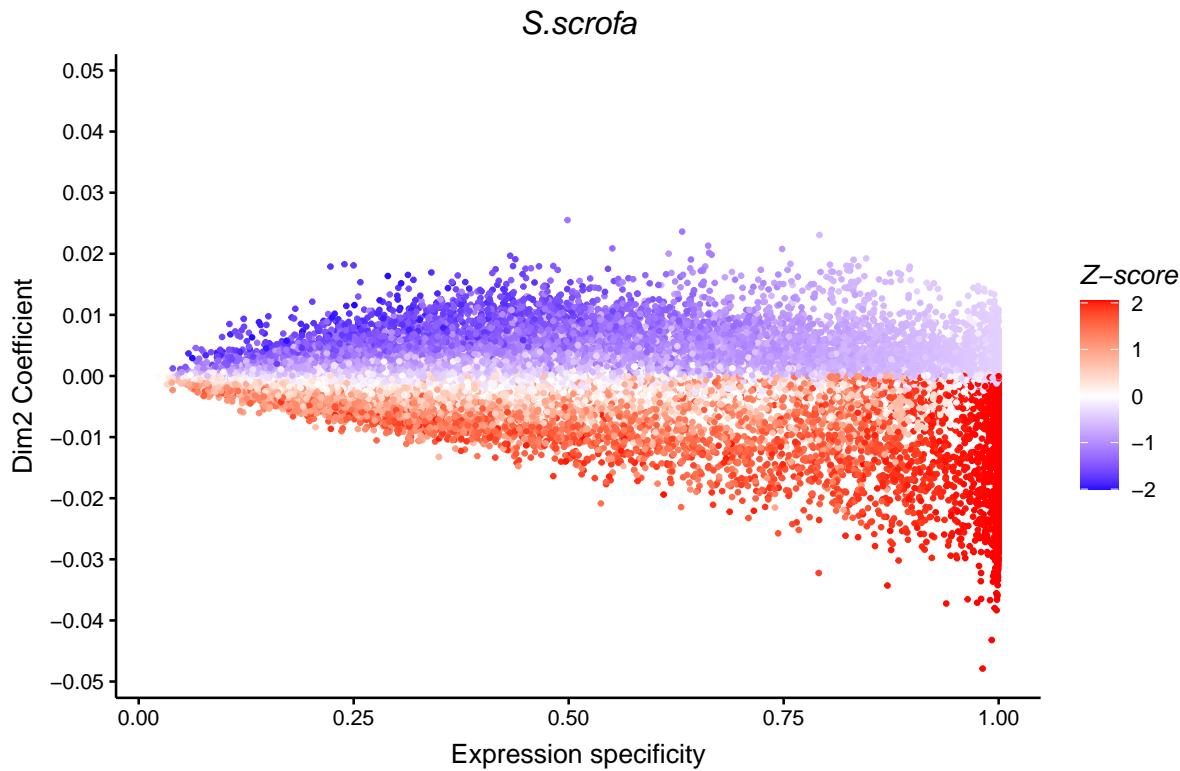
```

```

customTheme() + theme(legend.position = "right",
                      legend.direction = "vertical") +
  labs(title = paste0(getScientificName(species)), color = "Z-score") +
  theme(legend.key.size = unit(0.5, "cm"),
        plot.title = element_text(face = "italic"))

```

p2



7.3 GO analysis

Download the GO files from: <https://figshare.com/s/d96c586d5e53199d5370> We will perform the gene ontology analysis for genes with high coefficients. For GO analysis, we will use shared 1-to-1 orthologs in eight species as the background.

```

# load GO analysis files
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
file <- "allwayOrthologs_hsapiens-ptroglodytes-mmulatta-mmusculus-rnorvegicus-btaurus-sscrofa-ggallus_e"
orthologs <- read.table(file.path(path, "GOKeggFiles/"), file, header = TRUE,
                        sep = "\t")
head(orthologs)
#>           hsapiens external_gene_name   gene_biotype      ptroglodytes
#> 1 ENSG00000223224          SNORD71    snRNA ENSPTRG00000036308
#> 2 ENSG00000199574          SNORD18C    snRNA ENSPTRG00000026302
#> 3 ENSG00000284202          MIR137     miRNA ENSPTRG00000027612
#> 4 ENSG00000207797          MIR187     miRNA ENSPTRG00000050189
#> 5 ENSG00000198888          MT-ND1 protein_coding ENSPTRG00000042641
#> 6 ENSG00000198763          MT-ND2 protein_coding ENSPTRG00000042626
#>           mmulatta      mmusculus rnorvegicus          btaurus
#> 1 ENSMMUG00000034773 ENSMUSG00000077549 ENSRNOG00000059926 ENSBTAG00000043472

```

```

#> 2 ENSMMUG00000024353 ENSMUSG00000064514 ENSRNOG00000059435 ENSBTAG00000043531
#> 3 ENSMMUG00000026871 ENSMUSG00000065569 ENSRNOG00000035491 ENSBTAG00000029809
#> 4 ENSMMUG00000027111 ENSMUSG00000065532 ENSRNOG00000035521 ENSBTAG00000029796
#> 5 ENSMMUG00000028699 ENSMUSG00000064341 ENSRNOG00000030644 ENSBTAG00000043558
#> 6 ENSMMUG00000028695 ENSMUSG00000064345 ENSRNOG00000031033 ENSBTAG00000043571
#>           sscrofa          ggallus hsapiens_genelength
#> 1 ENSSSCG00000019395 ENSGALG00000025302                      86
#> 2 ENSSSCG00000018926 ENSGALG00000043763                      69
#> 3 ENSSSCG00000019038 ENSGALG00000018336                     102
#> 4 ENSSSCG00000019111 ENSGALG00000018285                     109
#> 5 ENSSSCG00000018065 ENSGALG00000042750                     956
#> 6 ENSSSCG00000018069 ENSGALG00000043768                   1042
#>           ptroglodytes_genelength mmulatta_genelength mmusculus_genelength
#> 1                      86                      86                      85
#> 2                      69                      70                      71
#> 3                     102                     102                     73
#> 4                     109                     109                     61
#> 5                     957                     955                     957
#> 6                    1044                    1042                   1038
#>           rnorvegicus_genelength btaurus_genelength sscrofa_genelength
#> 1                      84                      86                      86
#> 2                      71                      70                      70
#> 3                     102                     102                     102
#> 4                     104                     109                     104
#> 5                     955                     956                     955
#> 6                    1039                    1042                   1042
#>           ggallus_genelength
#> 1                      87
#> 2                      71
#> 3                      96
#> 4                      86
#> 5                     975
#> 6                    1041

```

We will use the goseq Bioconductor package to perform GO enrichment analysis.

```

# install packages
pkgs <- c("goseq")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install)) {
  if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
  BiocManager::install("goseq")
}
suppressPackageStartupMessages({
  library(goseq)
})

```

Let us perform GO analysis for top testis-specific genes in Dimension 2. The testis-specific genes have negative loadings.

```

sel_dim <- 2
sel_tissue <- "testis"
top_genes <- 100
# axis positive (pos) or negative (neg)

```

```
sel_sign <- "neg"
```

We will perform GO analysis for human first.

```
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue), "Tau",
                 "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$coef < 0, ]
} else {
  cat("\n selecting positive loadings")
  expr1_selsign <- expr1[expr1$coef >= 0, ]
}
#>
#> selecting negative loadings
expr1_selsign$score <- expr1_selsign$Tau * abs(expr1_selsign$coef)
expr1_selsign$rank <- rank(-1 * expr1_selsign$score)
expr1_selsign <- expr1_selsign[order(expr1_selsign$rank), ]
genes_fg <- row.names(expr1_selsign[expr1_selsign$rank <= top_genes, ])
if (species == "Homo_sapiens") {
  cat("\nUsing orthologs (human IDs) as background")
  genes_bg <- orthologs$hsapiens
}
#>
#> Using orthologs (human IDs) as background
if (species == "Mus_musculus") {
  cat("\nUsing orthologs (mouse IDs) as background")
  genes_bg <- orthologs$mmusculus
}
genes_bg <- genes_bg[!genes_bg %in% genes_fg]
genome <- "hg38"
total_genes <- unique(c(genes_fg, genes_bg))
up_genes <- as.integer(total_genes %in% genes_fg)
names(up_genes) <- total_genes

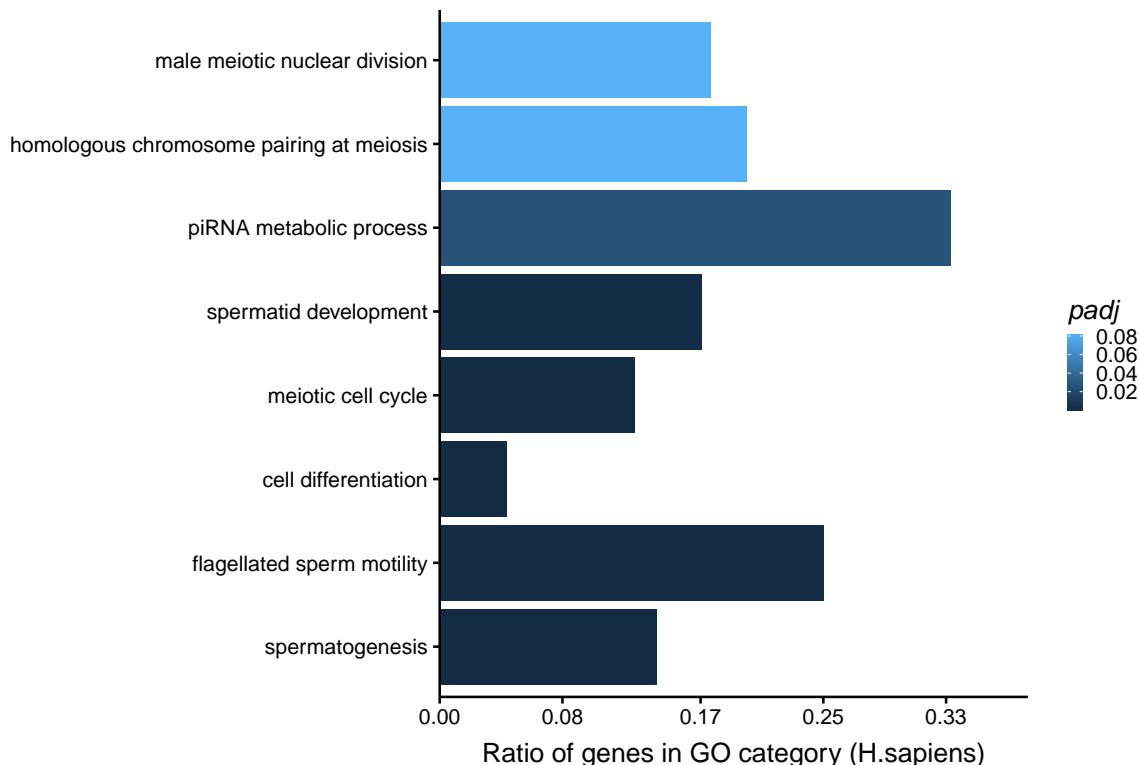
#> Using manually entered categories.
#> For 183 genes, we could not find any categories. These genes will be excluded.
#> To force their use, please run with use_genes_without_cat=TRUE (see documentation).
#> This was the default behavior for version 1.15.1 and earlier.
#> Calculating the p-values...
#> 'select()' returned 1:1 mapping between keys and columns

head(go.sub)
#>      category numDEInCat numInCat          term ontology
#> 2863 GO:0007283        28     198    spermatogenesis      BP
#> 5356 GO:0030317         8      32 flagellated sperm motility      BP
#> 5253 GO:0030154        22     500   cell differentiation      BP
#> 9828 GO:0051321         7      55    meiotic cell cycle      BP
#> 2865 GO:0007286         7      41    spermatid development      BP
#> 6760 GO:0034587         3       9    piRNA metabolic process      BP
```

```
#>           padj   ratio
#> 2863 2.132495e-19 0.1414
#> 5356 2.299849e-05 0.2500
#> 5253 1.157759e-04 0.0440
#> 9828 3.692907e-04 0.1273
#> 2865 1.539853e-03 0.1707
#> 6760 2.616171e-02 0.3333
```

Barplot with top human GO terms and their p-value.

```
# GO enrichment plot for human
go_out <- head(go.sub, n = 8)
go_out$padj <- as.numeric(go_out$padj)
go_out$term <- factor(go_out$term, levels = go_out$term)
breaks <- round(c(0, 1 / 4, 2 / 4, 3 / 4, 1) * max(go_out[["ratio"]]), 2)
go_plot <- ggplot2::ggplot(go_out, aes(x = term, y = ratio, fill = padj)) +
  geom_col() +
  scale_y_continuous(expand = c(0, 0), breaks = breaks,
                     limits = c(0, max(go_out[["ratio"]] + 0.05))) +
  scale_x_discrete() + coord_flip() +
  scale_color_gradient(low = "blue", high = "red") +
  ylab(paste0("Ratio of genes in GO category (",
              getScientificName(species), ")")) +
  xlab("") + customTheme() + theme(legend.position = "right",
                                    legend.direction = "vertical",
                                    plot.margin = unit(c(10, 5, 5, 5), "mm"))
go_plot
```



GO analysis for mouse.

```

species <- "Mus_musculus"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue), "Tau",
                  "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$coef < 0, ]
} else {
  cat("\n selecting positive loadings")
  expr1_selsign <- expr1[expr1$coef >= 0, ]
}
#>
##> selecting negative loadings
expr1_selsign$score <- expr1_selsign$Tau * abs(expr1_selsign$coef)
expr1_selsign$rank <- rank(-1 * expr1_selsign$score)
expr1_selsign <- expr1_selsign[order(expr1_selsign$rank), ]
genes_fg <- row.names(expr1_selsign[expr1_selsign$rank <= top_genes, ])
if (species == "Homo_sapiens") {
  cat("\nUsing orthologs (human IDs) as background")
  genes_bg <- orthologs$hsapiens
}
if (species == "Mus_musculus") {
  cat("\nUsing orthologs (mouse IDs) as background")
  genes_bg <- orthologs$mmusculus
}
#>
##> Using orthologs (mouse IDs) as background
genes_bg <- genes_bg[!genes_bg %in% genes_fg]
genome <- "mm10"
total_genes <- unique(c(genes_fg, genes_bg))
up_genes <- as.integer(total_genes %in% genes_fg)
names(up_genes) <- total_genes

##> Using manually entered categories.
##> For 133 genes, we could not find any categories. These genes will be excluded.
##> To force their use, please run with use_genes_without_cat=TRUE (see documentation).
##> This was the default behavior for version 1.15.1 and earlier.
##> Calculating the p-values...
##> 'select()' returned 1:1 mapping between keys and columns

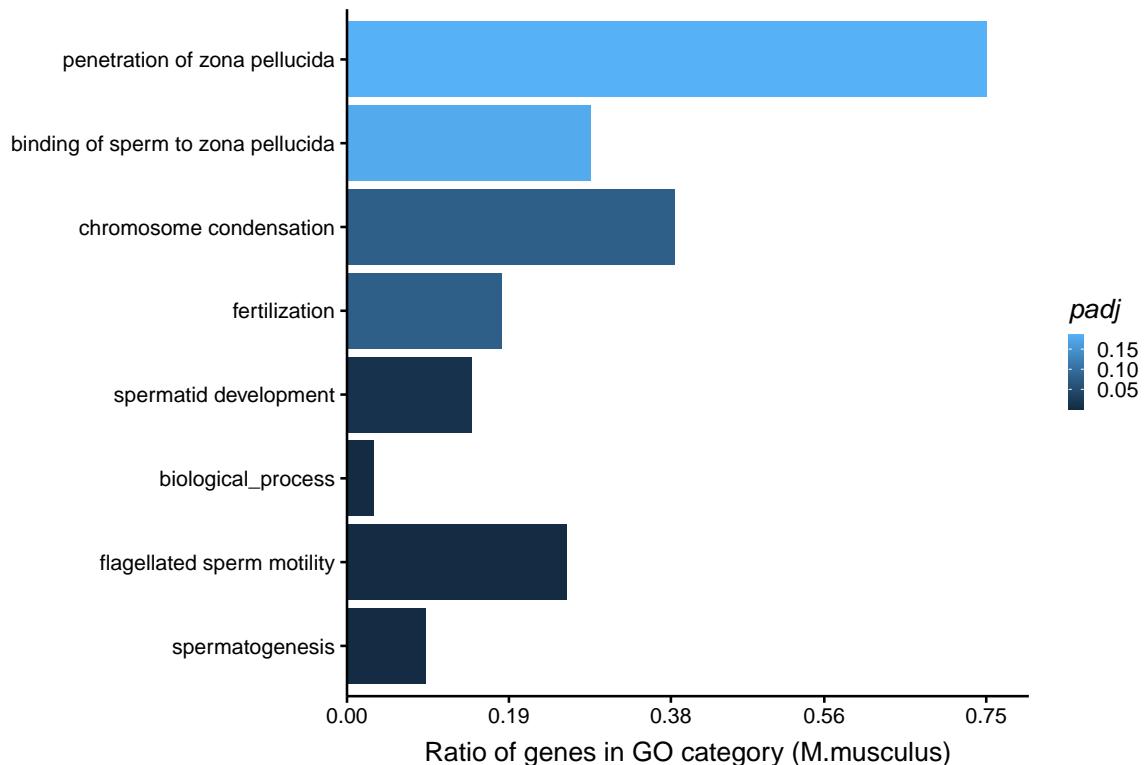
head(go.sub)
##>      category numDEInCat numInCat          term ontology
##> 2794 GO:0007283       17     185   spermatogenesis    BP
##> 5203 GO:0030317        9      35 flagellated sperm motility    BP
##> 2965 GO:0008150       29     919   biological_process    BP
##> 2796 GO:0007286        7      48   spermatid development    BP
##> 3346 GO:0009566        4      22   fertilization    BP
##> 5171 GO:0030261       5      13 chromosome condensation    BP
##>           padj ratio
##> 2794 1.390642e-06 0.0919
##> 5203 7.624827e-06 0.2571

```

```
#> 2965 2.245074e-04 0.0316
#> 2796 1.045540e-02 0.1458
#> 3346 7.747928e-02 0.1818
#> 5171 7.747928e-02 0.3846
```

Barplot with top mouse GO terms and their p-value.

```
# GO enrichment plot for mouse
go_out <- head(go_sub, n = 8)
go_out$padj <- as.numeric(go_out$padj)
go_out$term <- factor(go_out$term, levels = go_out$term)
breaks <- round(c(0, 1 / 4, 2 / 4, 3 / 4, 1) * max(go_out[["ratio"]]), 2)
go_plot <- ggplot2::ggplot(go_out, aes(x = term, y = ratio, fill = padj)) +
  geom_col() +
  scale_y_continuous(expand = c(0, 0), breaks = breaks,
                     limits = c(0, max(go_out[["ratio"]] + 0.05))) +
  scale_x_discrete() + coord_flip() +
  scale_color_gradient(low = "blue", high = "red") +
  ylab(paste0("Ratio of genes in GO category (",
              getScientificName(species), ")")) +
  xlab("") + customTheme() + theme(legend.position = "right",
                                    legend.direction = "vertical",
                                    plot.margin = unit(c(10, 5, 5, 5), "mm"))
go_plot
```



8 Identify tissue-specific genes

Next, we will identify those genes with significant contribution to different cell types. We will first create a null distribution of scores for the coefficient and identify genes of interest with respect to the null.

8.1 Create shuffled counts to generate null

We will create random average counts based on shuffled reads. Then we will apply TPM normalization for these counts using `normalizeTPM` function from the `SBF` package.

```
# set seed
s1 <- 135
s2 <- 13835
species <- c("Homo_sapiens", "Pan_troglodytes", "Macaca_mulatta",
            "Mus_musculus", "Rattus_norvegicus", "Bos_taurus",
            "Sus_scrofa", "Gallus_gallus")
species_short <- sapply(species, getSpeciesShortName)
tissues <- c("brain", "heart", "kidney", "liver", "lung", "testis")
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
counts_list_shuff <- metadata_list_shuff <- avg_counts_shuff <- list()
for (sp in species) {
  # reading raw counts
  counts <- read.table(paste0(path, "counts/", sp, "_rawcounts.tsv"),
                        header = TRUE, sep = "\t", row.names = 1)
  info <- data.table::tstrsplit(colnames(counts), "_")
  metadata <- data.frame(project = info[[1]],
                         species = info[[2]],
                         tissue = info[[3]],
                         gsm = info[[4]],
                         name = colnames(counts),
                         stringsAsFactors = FALSE)
  metadata$ref <- seq_len(nrow(metadata))
  metadata$key <- paste0(metadata$species, "_", metadata$tissue)
  metadata$tissue_factor <- factor(metadata$tissue)
  counts_avg <- calcAvgCounts(counts, metadata)
  cnames <- colnames(counts_avg)
  rnames <- row.names(counts_avg)
  set.seed(s1)
  counts_avg <- as.data.frame(apply(counts_avg, 2, sample))
  set.seed(s2)
  counts_avg <- as.data.frame(t(apply(counts_avg, 1, sample)))
  colnames(counts_avg) <- cnames
  row.names(counts_avg) <- rnames
  # normalize the shuffled counts to log TPM
  # set the path to the working directory. Change this accordingly
  path <- "~/Dropbox/0.Analysis/0.paper/"
  gene_length <- read.table(paste0(path, "ensembl94_annotation/", sp,
                                    "_genelength.tsv"), sep = "\t",
                            header = TRUE, row.names = 1,
                            stringsAsFactors = FALSE)
  if (!all(row.names(counts_avg) %in% row.names(gene_length))) stop("Error")
  gene_length$Length <- gene_length$Length / 1e3
  gene_length <- gene_length[row.names(counts_avg), , drop = TRUE]
  names(gene_length) <- row.names(counts_avg)
  counts_tpm <- normalizeTPM(rawCounts = counts_avg, gene_len = gene_length)
  min_tpm <- 1
  counts_tpm[counts_tpm < min_tpm] <- 1
  counts_tpm <- log2(counts_tpm)
```

```

info <- data.table::tstrsplit(colnames(counts TPM), "_")
metadata <- data.frame(
  species = info[[1]],
  tissue = info[[2]],
  name = colnames(counts TPM),
  stringsAsFactors = FALSE)
metadata$key <- paste0(metadata$species, "_", metadata$tissue)
avg_counts_shuff[[sp]] <- calcAvgCounts(counts TPM, metadata)
# write.table(avg_counts_shuff[[sp]], file=paste0(mainDir, "shuffledavgCounts/avg_logTPM_",
# #                                         species, "_SHUFF_counts_seed", z, "_", z1, ".tsv"),
# #                                         sep="\t", quote=F, col.names=NA)
counts_list_shuff[[sp]] <- counts TPM
metadata_list_shuff[[sp]] <- metadata
}
#>
#> TPM counts returned
sapply(counts_list_shuff, dim)
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus Rattus_norvegicus
#> [1,]      58676           31373        30807       54446       32623
#> [2,]          6              6            6           6           6
#>      Bos_taurus Sus_scrofa Gallus_gallus
#> [1,]     24596         25880       19152
#> [2,]          6              6            6
# remove zero counts
avg_counts_shuff <- lapply(avg_counts_shuff, removeZeros)
sapply(avg_counts_shuff, dim)
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus Rattus_norvegicus
#> [1,]     46083         29166       28353       42780       29571
#> [2,]          6              6            6           6           6
#>      Bos_taurus Sus_scrofa Gallus_gallus
#> [1,]     23968         24811       18996
#> [2,]          6              6            6

```

8.2 OSBF call for shuffled counts

Depending upon the shuffled counts this could take a while. Decrease tol for lower factorization error.

```

cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:07:29 PM 2022
# decrease tol to minimize error
osbf_shuf <- SBF(avg_counts_shuff, transform_matrix = TRUE, orthogonal = TRUE,
                  tol = 1e-2)
#
#> OSBF optimizing factorization error
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")
#> Wed Jun 22 04:07:31 PM 2022
osbf_shuf$error

```

```
#> [1] 9704.67
```

Compute Tau and scaled expression for the null datasets

```
Tau_null <- lapply(avg_counts_shuff, function(x) {calc_tissue_specificity(x)})
avg_counts_shuff_scaled <- lapply(avg_counts_shuff, function(x) {
  t(scale(t(x)))
})
combine_expr_null <- list()
for (sp in names(avg_counts_shuff_scaled)) {
  x <- as.data.frame(avg_counts_shuff_scaled[[sp]])
  x[["Tau"]] <- Tau_null[[sp]]
  combine_expr_null[[sp]] <- x
}
```

8.3 Identify testis-specific genes

Now let us find genes with significant loadings in dimension 2. We will use the empirical null generated from the shuffled counts to get the p-value.

```
sel_dim <- 2
sel_tissue <- "testis"
# axis positive (pos) or negative (neg)
sel_sign <- "neg"
species <- "Homo_sapiens"
species_short <- "hsapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(species_short, "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")

# null loadings for the same dimensions
expr_null <- combine_expr_null[[species]]
null_u <- osbf_shuf$u[[species]]
expr_null[["coef"]] <- null_u[, sel_dim, drop = TRUE]
expr1_null <- expr_null[, c(paste0(species_short, "_", sel_tissue), "Tau",
                             "coef")]
if (sel_sign == "pos") {
  expr1 <- expr1[expr1$coef >= 0, ]
  expr1_null <- expr1_null[expr1_null$coef >= 0, ]
} else if (sel_dim == "neg") {
  expr1 <- expr1[expr1$coef < 0, ]
  expr1_null <- expr1_null[expr1_null$coef < 0, ]
}
expr1$score <- expr1$Tau * abs(expr1$coef)
expr1$rank <- rank(-1 * expr1$score)
expr1 <- expr1[order(expr1$rank), ]

expr1_null$score <- expr1_null$Tau * abs(expr1_null$coef)
expr1$pvalue <- sapply(expr1$score, function(x) {
  sum(as.integer(expr1_null$score > x)) / length(expr1_null$score)
})
head(expr1)
```

```

#>          tissue_zscore      Tau      coef      score rank pvalue
#> ENSG00000122304    2.039236 0.9803727 -0.04262250 0.04178594    1     0
#> ENSG00000175646    2.039104 0.9800546 -0.04182715 0.04099289    2     0
#> ENSG00000118245    2.039801 0.9840092 -0.03901429 0.03839042    3     0
#> ENSG00000198033    2.040917 0.9948419 -0.03162446 0.03146134    4     0
#> ENSG00000203784    2.040995 0.9943771 -0.03103801 0.03086348    5     0
#> ENSG00000106013    2.041167 0.9978056 -0.03080571 0.03073811    6     0

```

Find the number of genes with significant p-value

```

# cut off for the p-value
alpha <- 1e-3
summary(expr1$pvalue <= alpha)
#>   Mode      FALSE      TRUE
#> logical    46730      571

```

Lets check the top 10 genes for dimension 4

```

# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
gene_info <- read.table(paste0(path, "ensembl94_annotation/", species_short,
                               "_genes_completeinfo.tsv"),
                        sep = "\t", header = TRUE, quote = "\"")
gene_info <- gene_info[!duplicated(gene_info$ensembl_gene_id), ]
gene_info <- gene_info[gene_info$ensembl_gene_id %in% row.names(expr1), ]
row.names(gene_info) <- gene_info$ensembl_gene_id
gene_info <- gene_info[row.names(expr1), ]
expr1$gene_name <- gene_info$external_gene_name
expr1$biotype <- gene_info$gene_biotype
head(expr1, n = 10)
#>          tissue_zscore      Tau      coef      score rank pvalue
#> ENSG00000122304    2.039236 0.9803727 -0.04262250 0.04178594    1     0
#> ENSG00000175646    2.039104 0.9800546 -0.04182715 0.04099289    2     0
#> ENSG00000118245    2.039801 0.9840092 -0.03901429 0.03839042    3     0
#> ENSG00000198033    2.040917 0.9948419 -0.03162446 0.03146134    4     0
#> ENSG00000203784    2.040995 0.9943771 -0.03103801 0.03086348    5     0
#> ENSG00000106013    2.041167 0.9978056 -0.03080571 0.03073811    6     0
#> ENSG00000163206    2.040205 0.9907169 -0.03100985 0.03072198    7     0
#> ENSG00000228075    2.041122 0.9965716 -0.03064384 0.03053878    8     0
#> ENSG00000163467    2.039460 0.9692823 -0.03000953 0.02908770    9     0
#> ENSG00000155087    2.041096 0.9961337 -0.02911094 0.02899839   10    0
#>          gene_name      biotype
#> ENSG00000122304      PRM2 protein_coding
#> ENSG00000175646      PRM1 protein_coding
#> ENSG00000118245      TNP1 protein_coding
#> ENSG00000198033      TUBA3C protein_coding
#> ENSG00000203784      LELP1 protein_coding
#> ENSG00000106013      ANKRD7 protein_coding
#> ENSG00000163206      SMCP protein_coding
#> ENSG00000228075      BOD1L2 protein_coding
#> ENSG00000163467      TSACC protein_coding
#> ENSG00000155087      ODF1 protein_coding

```

We see that the top genes are well studied genes with testis-specific expression.

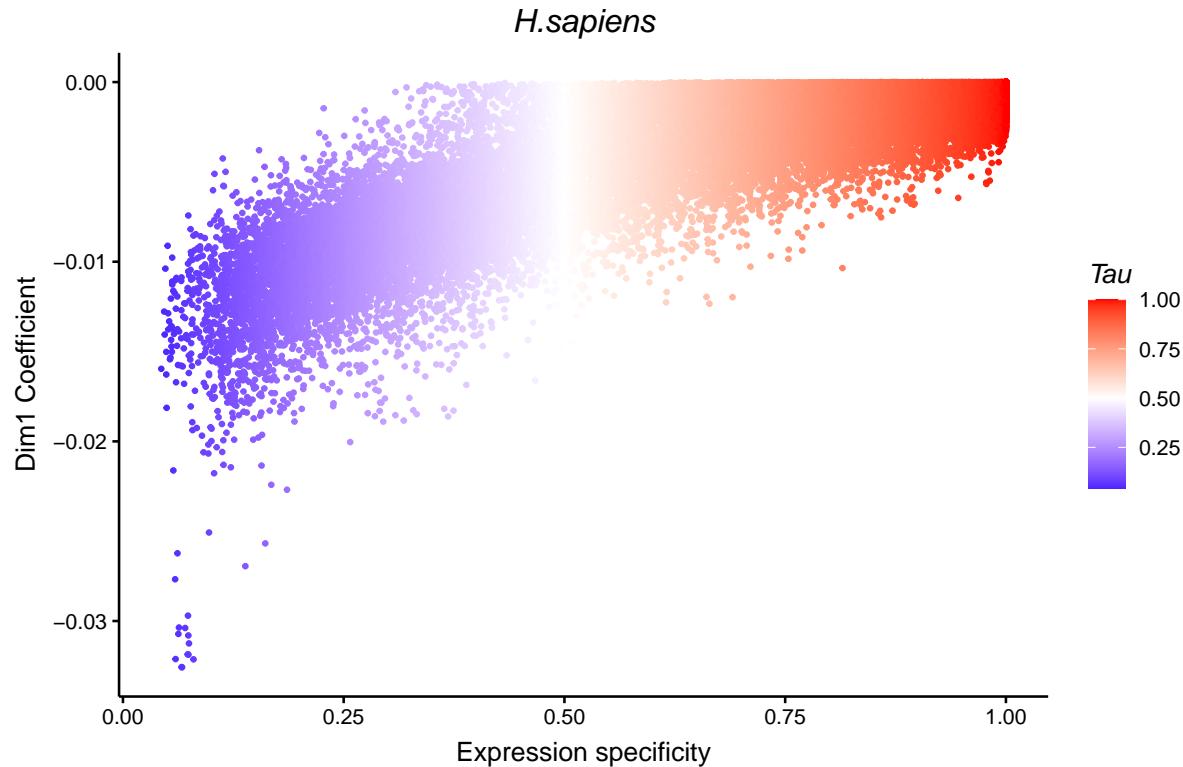
9 Exploring dimension 1 of the common space

Lets plot the U_1 loadings vs expression specificity

Lets plot for human

```
sel_dim <- 1
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]

# plot scatter
mid <- 0.5
p1 <- ggplot2::ggplot(expr, aes(x = Tau, y = coef, col = Tau)) +
  theme_bw() +
  geom_point(size = 0.5) + xlab("Expression specificity") +
  ylab(paste0("Dim", sel_dim, " Coefficient")) +
  scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                        high = "red", space = "Lab") +
  customTheme() + theme(legend.position = "right",
                        legend.direction = "vertical") +
  labs(title = getScientificName(species), color = "Tau") +
  theme(legend.key.size = unit(0.5, "cm"),
        plot.title = element_text(face = "italic"))
p1
```



9.1 Identify significant genes

Now let us find genes with significant loadings in dimension 1. We will use the empirical null generated from the shuffled counts to get the p-value.

```

sel_dim <- 1
combine_expr_Dim1 <- list()
for (sp in names(combine_expr)) {
  expr <- combine_expr[[sp]]
  osbf_coef <- osbf$u[[sp]]
  expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
  expr_null <- combine_expr_null[[sp]]
  null_u <- osbf_shuf$u[[sp]]
  expr_null[["coef"]] <- null_u[, sel_dim, drop = TRUE]
  expr$score <- abs(expr$coef)
  expr$rank <- rank(-1 * expr$score)
  expr_null$score <- abs(expr_null$coef)
  expr[[paste0("Dim", sel_dim, "_pval")]] <- sapply(expr$score, function(x) {
    sum(as.integer(expr_null$score > x)) / length(expr_null$score)
  })
  combine_expr_Dim1[[sp]] <- expr
}

# cut off for the p-value
alpha <- 1e-3
sapply(combine_expr_Dim1, function(x) {summary(x$Dim1_pval <= alpha)})
#>      Homo_sapiens Pan_troglodytes Macaca_mulatta Mus_musculus
#> Mode "logical"      "logical"      "logical"      "logical"
#> FALSE "47233"       "26150"       "27741"       "39055"
#> TRUE  "68"          "23"          "21"          "42"
#>      Rattus_norvegicus Bos_taurus Sus_scrofa Gallus_gallus
#> Mode "logical"      "logical"      "logical"      "logical"
#> FALSE "22976"       "20008"       "21418"       "17791"
#> TRUE  "28"          "20"          "30"          "19"

```

Now lets find the details of these genes.

- Are these protein coding or not?
- How many of these genes are mitochondrial genes?

```

# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
file <- "allwayOrthologs_hsapiens-ptroglodytes-mmulatta-mmusculus-rnorvegicus-btaurus-sscrofa-ggallus_e"
orthologs <- read.table(file.path(path, "GOKEggFiles/"), file), header = TRUE,
sep = "\t"

Dim1_genes <- list()
for (sp in names(combine_expr_Dim1)) {
  x <- combine_expr_Dim1[[sp]][combine_expr_Dim1[[sp]]$Dim1_pval <= alpha, ]
  x <- x[order(x$rank), c("rank", "score", "Tau", "Dim1_pval"), drop = FALSE]
  # get gene details
  gene_info <- read.table(paste0(path, "ensembl94_annotation/",
                                getSpeciesShortName(sp),
                                "_genes_completeinfo.tsv"),
                            sep = "\t", header = TRUE, quote = "\"")
  gene_info <- gene_info[!duplicated(gene_info$ensembl_gene_id), ]
  gene_info <- gene_info[gene_info$ensembl_gene_id %in% row.names(x), ]
  row.names(gene_info) <- gene_info$ensembl_gene_id
  gene_info <- gene_info[row.names(x), ]
  gene_info <- gene_info[, c("ensembl_gene_id", "external_gene_name",

```

```

    "gene_biotype", "chromosome_name")]
colnames(gene_info) <- c("id", "gene_name", "biotype", "chr")
species_allway_orthogs <- orthologs[, getSpeciesShortName(sp), drop = TRUE]
ortholog_status <- rep(FALSE, nrow(x))
ortholog_status[row.names(x) %in% species_allway_orthogs] <- TRUE
x$allwayOrtholog <- as.factor(ortholog_status)
df <- cbind(gene_info, x)
row.names(df) <- NULL
Dim1_genes[[sp]] <- df
}

```

Top Dim1 human genes

```

head(Dim1_genes[["Homo_sapiens"]])
#>      id gene_name      biotype chr rank      score      Tau
#> 1 ENSG00000198886 MT-ND4 protein_coding MT 1 0.03257503 0.06692962
#> 2 ENSG00000198938 MT-CO3 protein_coding MT 2 0.03256351 0.06646709
#> 3 ENSG00000198899 MT-ATP6 protein_coding MT 3 0.03213476 0.07996523
#> 4 ENSG00000198712 MT-CO2 protein_coding MT 4 0.03211765 0.05966365
#> 5 ENSG00000198804 MT-CO1 protein_coding MT 5 0.03187087 0.07418785
#> 6 ENSG00000210082 MT-RNR2 Mt_rRNA MT 6 0.03185734 0.07336057
#> Dim1_pval allwayOrtholog
#> 1 0 TRUE
#> 2 0 TRUE
#> 3 0 FALSE
#> 4 0 TRUE
#> 5 0 TRUE
#> 6 0 FALSE

```

Top Dim1 mouse genes

```

head(Dim1_genes[["Mus_musculus"]])
#>      id gene_name      biotype chr rank      score      Tau
#> 1 ENSMUSG00000064351 mt-Co1 protein_coding MT 1 0.03037895 0.1715010
#> 2 ENSMUSG00000064370 mt-Cytb protein_coding MT 2 0.02922452 0.1681381
#> 3 ENSMUSG00000064341 mt-Nd1 protein_coding MT 3 0.02826009 0.1706786
#> 4 ENSMUSG00000064345 mt-Nd2 protein_coding MT 4 0.02713685 0.1812218
#> 5 ENSMUSG00000064363 mt-Nd4 protein_coding MT 5 0.02642295 0.1783649
#> 6 ENSMUSG00000064367 mt-Nd5 protein_coding MT 6 0.02639644 0.1729718
#> Dim1_pval allwayOrtholog
#> 1 0 TRUE
#> 2 0 TRUE
#> 3 0 TRUE
#> 4 0 TRUE
#> 5 0 TRUE
#> 6 0 TRUE

```

Summary stats of dimension 1 genes

```

cnames <- c("Dim", "species", "nTotal", "nCoding", "nMt", "nAllwayOrtholog")
summary_stats <- data.frame(matrix(NA, nrow = 0, ncol = length(cnames)))
colnames(summary_stats) <- cnames

for (sp in names(Dim1_genes)) {
  df <- Dim1_genes[[sp]]
  stats <- data.frame(matrix(0L, nrow = 1, ncol = length(cnames)))

```

```

colnames(stats) <- cnames
stats$Dim <- "Dim1"
stats$species <- getSpeciesShortName(sp)
stats$nTotal <- nrow(df)
stats$nCoding <- round(nrow(df[df$biotype == "protein_coding",
                                , drop = FALSE]) * 100 / nrow(df), 2)
stats$nMt <- round(nrow(df[tolower(df$chr) == "mt",
                                , drop = FALSE]) * 100 / nrow(df), 2)
stats$nAllwayOrtholog <- round(nrow(df[df$allwayOrtholog == TRUE,
                                         , drop = FALSE]) * 100 / nrow(df), 2)
summary_stats <- rbind(summary_stats, stats)
}
summary_stats
#>   Dim      species nTotal nCoding   nMt nAllwayOrtholog
#> 1 Dim1     hsapiens    68  91.18 23.53      27.94
#> 2 Dim1 ptroglodytes    23  86.96 34.78      39.13
#> 3 Dim1 mmulatta      21  90.48 42.86      38.10
#> 4 Dim1 mmusculus      42  88.10 21.43      28.57
#> 5 Dim1 rnorvegicus    28  89.29 50.00      32.14
#> 6 Dim1 btaurus        20  90.00 75.00      50.00
#> 7 Dim1 sscrofa         30  96.67 46.67      33.33
#> 8 Dim1 ggallus         19  89.47 78.95      52.63

```

10 Session info

```

sessionInfo()
#> R version 4.2.0 (2022-04-22)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 20.04.4 LTS
#>
#> Matrix products: default
#> BLAS:    /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
#>
#> locale:
#> [1] LC_CTYPE=en_US.UTF-8          LC_NUMERIC=C
#> [3] LC_TIME=en_US.UTF-8          LC_COLLATE=en_US.UTF-8
#> [5] LC_MONETARY=en_US.UTF-8       LC_MESSAGES=en_US.UTF-8
#> [7] LC_PAPER=en_US.UTF-8          LC_NAME=C
#> [9] LC_ADDRESS=C                 LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=en_US.UTF-8  LC_IDENTIFICATION=C
#>
#> attached base packages:
#> [1] grid      stats      graphics grDevices utils      datasets methods
#> [8] base
#>
#> other attached packages:
#> [1] goseq_1.48.0           geneLen DataBase_1.32.0 BiasedUrn_1.07
#> [4] ggplot2_3.3.6          ggthemes_4.2.4        RColorBrewer_1.1-3
#> [7] ComplexHeatmap_2.12.0  matrixStats_0.62.0   dplyr_1.0.9
#> [10] data.table_1.14.2      SBF_1.0.0.0
#>

```

```

#> loaded via a namespace (and not attached):
#> [1] colorspace_2.0-3                  rjson_0.2.21
#> [3] ellipsis_0.3.2                   rprojroot_2.0.3
#> [5] circlize_0.4.15                  XVector_0.36.0
#> [7] GenomicRanges_1.48.0             GlobalOptions_0.1.2
#> [9] fs_1.5.2                         clue_0.3-61
#> [11] rstudioapi_0.13                farver_2.1.0
#> [13] remotes_2.4.2                  bit64_4.0.5
#> [15] AnnotationDbi_1.58.0           fansi_1.0.3
#> [17] xml2_1.3.3                     splines_4.2.0
#> [19] codetools_0.2-18               doParallel_1.0.17
#> [21] cachem_1.0.6                  knitr_1.39
#> [23] pkgload_1.2.4                 Rsamtools_2.12.0
#> [25] GO.db_3.15.0                  dbplyr_2.1.1
#> [27] cluster_2.1.3                 png_0.1-7
#> [29] compiler_4.2.0                httr_1.4.3
#> [31] assertthat_0.2.1              Matrix_1.4-1
#> [33] fastmap_1.1.0                 cli_3.3.0
#> [35] htmltools_0.5.2              prettyunits_1.1.1
#> [37] tools_4.2.0                   gtable_0.3.0
#> [39] glue_1.6.2                    GenomeInfoDbData_1.2.8
#> [41] rappdirs_0.3.3              Rcpp_1.0.8.3
#> [43] Biobase_2.56.0              vctrs_0.4.1
#> [45] Biostrings_2.64.0            nlme_3.1-157
#> [47] rtracklayer_1.56.0           iterators_1.0.14
#> [49] xfun_0.31                   stringr_1.4.0
#> [51] ps_1.7.0                     brio_1.1.3
#> [53] testthat_3.1.4              lifecycle_1.0.1
#> [55] restfulr_0.0.13            devtools_2.4.3
#> [57] XML_3.99-0.9                zlibbioc_1.42.0
#> [59] scales_1.2.0                 hms_1.1.1
#> [61] MatrixGenerics_1.8.0        parallel_4.2.0
#> [63] SummarizedExperiment_1.26.1 curl_4.3.2
#> [65] yaml_2.3.5                  memoise_2.0.1
#> [67] biomaRt_2.52.0              stringi_1.7.6
#> [69] RSQLite_2.2.14              highr_0.9
#> [71] S4Vectors_0.34.0            BiocIO_1.6.0
#> [73] desc_1.4.1                  foreach_1.5.2
#> [75] filelock_1.0.2              GenomicFeatures_1.48.1
#> [77] BiocGenerics_0.42.0         pkgbuild_1.3.1
#> [79] BiocParallel_1.30.2          shape_1.4.6
#> [81] GenomeInfoDb_1.32.2          rlang_1.0.2
#> [83] pkgconfig_2.0.3              bitops_1.0-7
#> [85] evaluate_0.15                lattice_0.20-45
#> [87] purrrr_0.3.4                GenomicAlignments_1.32.0
#> [89] labeling_0.4.2              bit_4.0.4
#> [91] processx_3.5.3              tidyselect_1.1.2
#> [93] magrittr_2.0.3              R6_2.5.1
#> [95] IRanges_2.30.0              generics_0.1.2
#> [97] DelayedArray_0.22.0          DBI_1.1.2
#> [99] mgcv_1.8-40                 pillar_1.7.0
#> [101] withr_2.5.0                KEGGREST_1.36.0
#> [103] RCurl_1.98-1.6             tibble_3.1.7

```

```
#> [105] crayon_1.5.1           utf8_1.2.2
#> [107] BiocFileCache_2.4.0    rmarkdown_2.14
#> [109] GetoptLong_1.0.5       progress_1.2.2
#> [111] usethis_2.1.6         blob_1.2.3
#> [113] callr_3.7.0          digest_0.6.29
#> [115] stats4_4.2.0         munsell_0.5.0
#> [117] sessioninfo_1.2.2
```

11 References