EXTruvIIInb: The EXTended RUV-III-NB

Hsiao-Chi Liao

2024-10-12

Contents

1 Introduction

Single-cell multimodal technologies provide an opportunity to study biological mechanisms more comprehensively. The integrated analysis of matched single-cell transcriptomics (mRNA expression) and proteomics (protein abundance) data can help reveal biological insights that would not have been possible from separate analyses of each modality.

1.1 Motivation

Unwanted variation from sources such as shared batches and modality-specific library size effects inevitably exists in the data from both domains. If not properly corrected, the unwanted variation can potentially lead to misleading conclusions being made from the downstream analyses. However, none of the existing methods adequately takes biological factors into account in their models, often leading to the removal of both biological and unwanted variations when they are associated. To address the limitations, we propose the Extended RUV-III-NB. Our model accounts for biology and decomposes unwanted variation into joint and modality-specific components, providing a clearer understanding of the unwanted variation present in the data.

1.2 The RUV-III-NB framework and the Extended RUV-III-NB model

RUV-III-NB (Salim et al., 2022) is a method developed for removing unwanted variation from single-cell mRNA UMI counts, using a subset of annotated cells (as low as 5%) for model training. RUV-III-NB infers unwanted factors without requiring them to be specified in the model, and provides two types of corrected counts: logPAC (percentile adjusted counts on the natural logarithm scale) and Pearson residuals. logPAC is derived from the probability mass function of the negative binomial distribution, while Pearson residuals are calculated using the first and second moments (the expected value and the variance) of the negative binomial distribution. Pearson residuals tend to be more robust when count data deviates from the negative binomial distribution, whereas logPAC provides more accurate results when the data approximates negative binomial.

The model of the Extended RUV-III-NB which shares the same general structure with the RUV-III-NB model is as follows.

$$\log(\boldsymbol{\mu_f}) = \zeta_f \mathbf{1} + \boldsymbol{M\beta_f} + \boldsymbol{W\alpha_f},$$

The Extended RUV-III-NB has a different design for the unwanted component $\boldsymbol{W\alpha_f}$, where $\boldsymbol{W} = \left[\boldsymbol{W_1}| \quad \boldsymbol{W_2}| \quad \boldsymbol{W_3}\right]_{n\times(k_1+k_2+k_3)}$

$$\boldsymbol{\alpha_f} = \begin{bmatrix} \boldsymbol{\alpha_{1,f}} \\ \mathbf{0} \\ \boldsymbol{\alpha_{3,f}} \end{bmatrix}_{(k_1+k_3)\times 1} \quad \text{for } f \text{ from the mRNA domain, and}$$

$$\boldsymbol{\alpha_f} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\alpha_{2,f}} \\ \boldsymbol{\alpha_{3,f}} \end{bmatrix}_{(k_2+k_3)\times 1} \quad \text{for } f \text{ from the protein domain.}$$

2 Installing and loading the packages

We firstly install the Extended RUV-III-NB package with the following.

```
devtools::install_github("HsiaoChiLiao/EXTruvIIInb", build_vignettes = FALSE)
```

Below packages are required for the analysis in this Vignette and are loaded below.

```
library(EXTruvIIInb)

suppressPackageStartupMessages({
    library(ruvIIInb)
    library(DelayedArray)
    library(SingleCellExperiment)
    library(ggplot2)
    library(GGally)
    library(RColorBrewer)
    library(uwot)
    library(lisi)
    library(gridExtra)
    library(parallel)
    library(cowplot)
    library(metapod)
})
```

3 Anslysing the Triana dataset with EXTruvIIInb

3.1 The Triana dataset

This human bone marrow dataset was published by Triana et al. (2021, Nature Immunology). After quality control (QC), this dataset contains 422 targeted mRNA features, and 97 proteins measured by oligo-tagged antibodies (Antibody-Derived Tags, ADT). A total of 9,663 cells were randomly selected from 49,057 cells from 3 healthy young adults and 3 healthy aged adults with each sample from a different batch, are used for the below demonstration.

The build-in data `triana_data` can be called by the function `data()` and it contains (1) the raw counts of mRNA and ADT features (`rna_raw` and `adt_raw`), (2) the metadata of the cells (cell annotation and batch information), (3) the negative control sets for inferring $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\boldsymbol{W}_3$

```
# load the build-in data
data("triana_data")
str(triana_data)
#> List of 6
#>  $ rna_raw : num [1:9663, 1:422] 4 83 135 37 3 7 3 79 0 217 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:9663] "705706_Aged2_705706_4" "226650_BM3_226650_8"
↪  "85319_BM3_85319_8" "644530_BM3_644530_8" ...
#>   .. ..$ : chr [1:422] "mod1_ACTG1" "mod1_ADGRG1" "mod1_AIF1"
↪  "mod1_ANKRD28" ...
```

```
#>  $ adt_raw : num [1:9663, 1:97] 3 46 11 22 4 38 31 24 8 39 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:9663] "705706_Aged2_705706_4" "226650_BM3_226650_8"
↪  "85319_BM3_85319_8" "644530_BM3_644530_8" ...
#>   .. ..$ : chr [1:97] "mod2_B7.H4" "mod2_CD10" "mod2_CD103" "mod2_CD117"
↪  ...
#>  $ metadata:'data.frame':    9663 obs. of  4 variables:
#>   ..$ sample_class    : chr [1:9663] "Old" "Young" "Young" "Young" ...
#>   ..$ sample_batch    : chr [1:9663] "Aged2" "BM3" "BM3" "BM3" ...
#>   ..$ cellType_broad  : chr [1:9663] "CD8+ T cells" "Monocytes" "HSCs &
↪  MPPs" "CD8+ T cells" ...
#>   ..$ cellType_refined: Factor w/ 43 levels "Plasma cells",..: 19 13 14 27
↪  22 31 15 3 22 13 ...
#>  $ nc1     : chr [1:22] "mod1_CKS1B" "mod1_RSL1D1" "mod1_NASP" "mod1_PTMA"
↪  ...
#>  $ nc2     : chr [1:42] "mod2_B7.H4" "mod2_CD124" "mod2_CD126" "mod2_CD137"
↪  ...
#>  $ nc3     : chr [1:20] "mod2_CD133" "mod1_NPM1" "mod2_CD33" "mod1_CKAP5"
↪  ...

# the components in the 'triana_data' list
rna_raw <- triana_data$rna_raw
adt_raw <- triana_data$adt_raw
meta <- triana_data$meta
nc1 <- triana_data$nc1
nc2 <- triana_data$nc2
nc3 <- triana_data$nc3

# the raw counts
rna_raw[1:5, 1:3]
#>                       mod1_ACTG1 mod1_ADGRG1 mod1_AIF1
#> 705706_Aged2_705706_4          4           0         0
#> 226650_BM3_226650_8           83           0        11
#> 85319_BM3_85319_8            135           1        14
#> 644530_BM3_644530_8           37           0         0
#> 263768_Aged2_263768_4          3           4         0
adt_raw[1:5, 1:3]
#>                       mod2_B7.H4 mod2_CD10 mod2_CD103
#> 705706_Aged2_705706_4          3         3         11
#> 226650_BM3_226650_8           46         4          6
#> 85319_BM3_85319_8             11         4          0
#> 644530_BM3_644530_8           22         1          1
#> 263768_Aged2_263768_4          4         1          0

# the metadata for each cell
head(triana_data$meta)
#>                       sample_class sample_batch cellType_broad
#> 705706_Aged2_705706_4          Old        Aged2   CD8+ T cells
#> 226650_BM3_226650_8          Young          BM3      Monocytes
#> 85319_BM3_85319_8            Young          BM3    HSCs & MPPs
```

```
#> 644530_BM3_644530_8          Young          BM3   CD8+ T cells
#> 263768_Aged2_263768_4          Old        Aged2   CD8+ T cells
#> 223764_BM1_223764_9          Young          BM1     CD4+ cells
#>                                                  cellType_refined
#> 705706_Aged2_705706_4 CD8+CD103+ tissue resident memory T cells
#> 226650_BM3_226650_8                          Classical Monocytes
#> 85319_BM3_85319_8                            Early promyelocytes
#> 644530_BM3_644530_8                CD8+ central memory T cells
#> 263768_Aged2_263768_4              CD8+ effector memory T cells
#> 223764_BM1_223764_9                           CD4+ naive T cells
```

3.2 Running the vanilla EXTruvIIInb (the version without constraints)

Based on our experience, a majority of the library size effect of each modality can usually be captured with one unwanted factor. Thus, we set $k_1 = k_2 = 1$ for the inference of the modality-specific library size (and other unknown unwanted variation), and $k_3 = 2$ for capturing the joint unwanted effects.

```
# Using known cell types to define pseudo-replicates
anno <- meta$cellType_broad

# Construct the replicate matrix M using the known
# cell-types
unique_ctype <- unique(anno)[!is.na(unique(anno))]
M <- matrix(0, nrow(meta), length(unique_ctype))
dim(M)
#> [1] 9663   13
for (i in 1:length(unique_ctype)) {
    M[anno == unique_ctype[i], i] <- 1
}

# Running with the vanilla Extended RUV-III-NB
a <- Sys.time()
extruv3nb_broad <- extFastruvIIInb_vanilla(Y1 = DelayedArray(t(rna_raw)),
    Y2 = DelayedArray(t(adt_raw)), M = M, ctl1 = nc1, ctl2 = nc2,
    ctl3 = nc3, k1 = 1, k2 = 1, k3 = 2, ncores = 2)
#> [1] "time to Winsorize RNA count matrix: 0.741657018661499"
#> [1] "time to Winsorize ADT count matrix: 0.0369200706481934"
#> [1] "Batch variable not supplied...assuming cells come from one batch"
#> Loading required package: limma
#>
#> Attaching package: 'limma'
#> The following object is masked from 'package:BiocGenerics':
#>
#>     plotMA
#>
#> Attaching package: 'edgeR'
#> The following object is masked from 'package:SingleCellExperiment':
```

```
#>
#>     cpm
#> [1] "Start..."
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Outer Iter 1, Inner iter 1 logl-likelihood:-1697955.16063631"
#> [1] "Inner iter:2"
#> [1] "Outer Iter 1, Inner iter 2 logl-likelihood:-1667317.94083977"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Outer Iter 1, Inner iter 3 logl-likelihood:-1667317.94083977"
#> [1] "If Updating psi in Outer Iter 1; will obtain
↪  logl-likelihood:-1643089.81012007"
#> [1] "Updating psi in Outer Iter 1; logl-likelihood:-1643089.81012007"
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Outer Iter 2, Inner iter 1 logl-likelihood:-1643089.81012007"
#> [1] "Inner iter:2"
#> [1] "Outer Iter 2, Inner iter 2 logl-likelihood:-1642805.60197101"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Outer Iter 2, Inner iter 3 logl-likelihood:-1642805.60197101"
#> [1] "If Updating psi in Outer Iter 2; will obtain
↪  logl-likelihood:-1636977.2671381"
#> [1] "Updating psi in Outer Iter 2; logl-likelihood:-1636977.2671381"
#> [1] "Inner iter:1"
#> [1] "Outer Iter 3, Inner iter 1 logl-likelihood:-1626187.91248193"
#> [1] "Inner iter:2"
#> [1] "Outer Iter 3, Inner iter 2 logl-likelihood:-1592750.97047239"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Outer Iter 3, Inner iter 3 logl-likelihood:-1592750.97047239"
#> [1] "If Updating psi in Outer Iter 3; will obtain
↪  logl-likelihood:-1583931.9136506"
#> [1] "Updating psi in Outer Iter 3; logl-likelihood:-1583931.9136506"
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Outer Iter 4, Inner iter 1 logl-likelihood:-1582319.75025595"
#> [1] "Inner iter:2"
#> [1] "Outer Iter 4, Inner iter 2 logl-likelihood:-1572046.2707892"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Inner iter:3"
#> [1] "Outer Iter 4, Inner iter 3 logl-likelihood:-1572046.2707892"
#> [1] "If Updating psi in Outer Iter 4; will obtain
↪  logl-likelihood:-1569424.87666865"
```

```
#> [1] "Updating psi in Outer Iter 4; logl-likelihood:-1569424.87666865"
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Inner iter:1"
#> [1] "Outer Iter 5, Inner iter 1 logl-likelihood:-1569424.87666865"
#> [1] "Inner iter:2"
#> [1] "Inner iter:2"
#> [1] "Inner iter:2"
#> [1] "Outer Iter 5, Inner iter 2 logl-likelihood:-1569424.87666865"
#> [1] "If Updating psi in Outer Iter 5; will obtain
↪  logl-likelihood:-1569424.87666865"
#> [1] "Updating psi in Outer Iter 5; logl-likelihood:-1569424.87666865"
#> [1] "Both loops converged; checking the logl-likelihood again:
↪  -1569424.87666865"
#> [1] "Estimating W for all samples..."
#> [1] "Estimating Mb...."
b <- Sys.time()
print("finished!")
#> [1] "finished!"
print(b - a)
#> Time difference of 1.929838 mins
```

The `extruv3nb_broad` is an R object that contains the corrected data and the parameters estimated by the model.

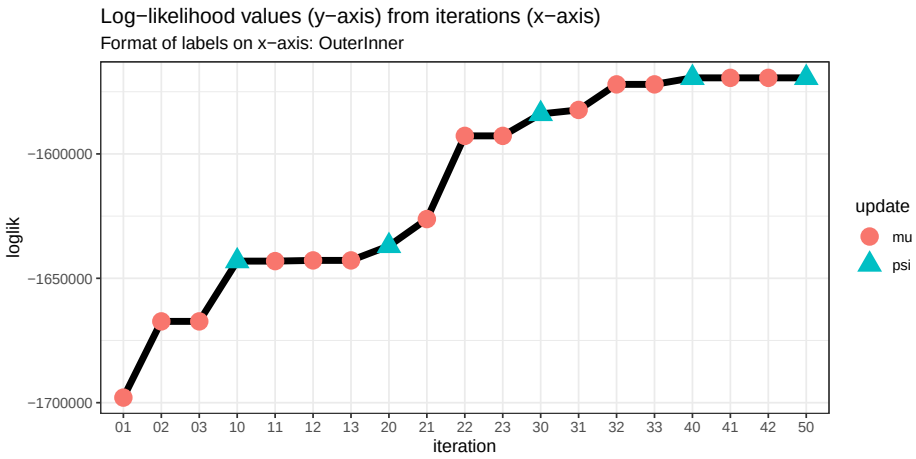
4 Results

4.1 The convergence

Firstly, we check the convergence of the modified IRLS by examining the log-likelihood values from the iterations. In the modified IRLS algorithm, the inner loop updates parameters relevant to the mean parameter μ and the outer loop updates the dispersion parameter ψ.

```
# extruv3nb_broad$logl.outer extruv3nb_broad$logl.inner.ls

loglik <- c(extruv3nb_broad$logl.inner.ls[[1]], extruv3nb_broad$logl.outer[1],
    extruv3nb_broad$logl.inner.ls[[2]], extruv3nb_broad$logl.outer[2],
    extruv3nb_broad$logl.inner.ls[[3]], extruv3nb_broad$logl.outer[3],
    extruv3nb_broad$logl.inner.ls[[4]], extruv3nb_broad$logl.outer[4],
    extruv3nb_broad$logl.inner.ls[[5]], extruv3nb_broad$logl.outer[5])
# iii o iii o iii o iii o ii o

df_graph <- data.frame(loglik = loglik, iteration = c(paste0(0,
    1:3), "10", paste0(1, 1:3), "20", paste0(2, 1:3), "30", paste0(3,
    1:3), "40", paste0(4, 1:2), "50"), update = c(rep("mu", 3),
    "psi", rep("mu", 3), "psi", rep("mu", 3), "psi", rep("mu",
        3), "psi", rep("mu", 2), "psi"))

ggplot(data = df_graph, aes(x = iteration, y = loglik, group = 1)) +
```

```
        geom_line(linewidth = 2) + geom_point(aes(shape = update,
        color = update), size = 5) + labs(title = "Log-likelihood values (y-axis)
        ↪  from iterations (x-axis)",
        subtitle = "Format of labels on x-axis: OuterInner") + theme_bw()
```



The log-likelihood values (on the y-axis) increase in the later iterations, indicating that the algorithm is performing as intended.

4.2 The corrected data

Then, to obtain the corrected data, we run the below.

```
        # Creating a SingleCellExperiment object
        sce_extruv3nb_broad <- makeSCE2(extruv3nb_broad, cData = meta)
        #> Loading required package: rhdf5
        #>
        #> Attaching package: 'HDF5Array'
        #> The following object is masked from 'package:rhdf5':
        #>
        #>     h5ls
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        #> [1] "1/2"
        #> [1] "2/2"
        print("passed converting to sce")
        #> [1] "passed converting to sce"

        # Obtaining corrected data the percentile adjusted count on
```

```
# the natural log scale
rna_logPAC <- as.matrix(assays(sce_extruv3nb_broad[[1]])$logPAC)
adt_logPAC <- as.matrix(assays(sce_extruv3nb_broad[[2]])$logPAC)
# the Pearson residuals
rna_PearsonRes <- as.matrix(assays(sce_extruv3nb_broad[[1]])$pearson)
adt_PearsonRes <- as.matrix(assays(sce_extruv3nb_broad[[2]])$pearson)
```

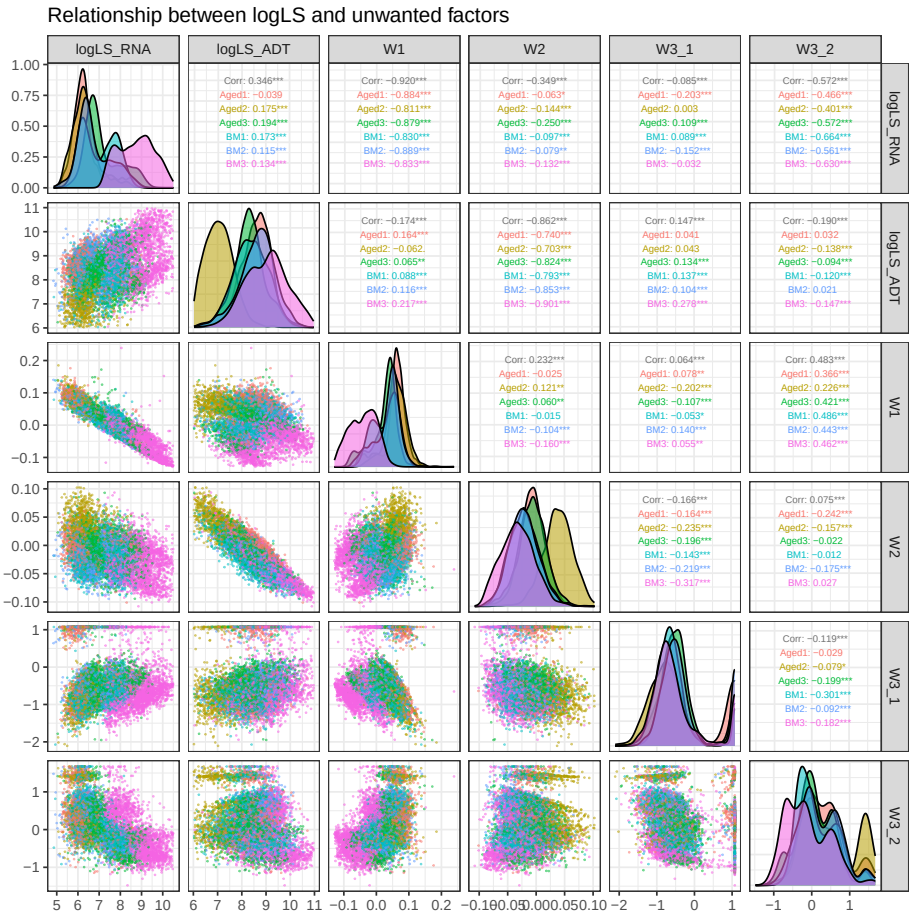
Even though we mostly perform downstream analyses using the percentile adjusted counts on the natural log scale, the corrected counts can be obtained with the following.

```
# the corrected count matrix of mRNA features (genes on
# rows and cells on columns)
(exp(rna_logPAC) - 1)[1:5, 1:3]
#>              705706_Aged2_705706_4 226650_BM3_226650_8 85319_BM3_85319_8
#> mod1_ACTG1                      18                  17                10
#> mod1_ADGRG1                      0                   0                 0
#> mod1_AIF1                        0                   4                 1
#> mod1_ANKRD28                     0                   0                 0
#> mod1_ANLN                        0                   0                 0
# the corrected count matrix of protein features (proteins
# on rows and cells on columns)
(exp(adt_logPAC) - 1)[1:5, 1:3]
#>            705706_Aged2_705706_4 226650_BM3_226650_8 85319_BM3_85319_8
#> mod2_B7.H4                    19                  17                23
#> mod2_CD10                     10                   1                 6
#> mod2_CD103                    39                   0                 0
#> mod2_CD117                    10                  10                78
#> mod2_CD11a                   161                 152                55
```

4.3 The inferred unwanted factors

For interpreting the inferred unwanted factors, we make panels of scatter plots, coloured the cells (dots) by the batch they came from, to examine the relationship between $logLS$ and the vectors in the $\boldsymbol{W}$ matrics.

```
gp.dat <- data.frame(logLS_RNA = log(apply(rna_raw, 1, sum) +
    1), logLS_ADT = log(apply(adt_raw, 1, sum) + 1), Ws = cbind(W1 =
    ↪  extruv3nb_broad$W1,
    W2 = extruv3nb_broad$W2, W3 = extruv3nb_broad$W3))
colnames(gp.dat)[-(1:2)] <- c("W1", "W2", "W3_1", "W3_2")

p <- ggpairs(gp.dat, columns = 1:dim(gp.dat)[2], aes(color =
↪  as.factor(meta$sample_batch),
    alpha = 0.5), upper = list(continuous = wrap("cor", size = 2)),
    lower = list(continuous = wrap("points", size = 0.1)), title =
    ↪  paste0("Relationship between logLS and unwanted factors")) +
    theme(axis.text = element_text(size = 10), axis.text.x =
    ↪  element_text(angle = 45,
        hjust = 1)) + theme_bw()
print(p)
```



From the figure above, as shown in the high correlations, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ capture $logLS_{mRNA}$ and $logLS_{ADT}$ effects, respectively. The batch effects that is associated with the library size effect are captured by the modality-specific unwanted factors $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ as well. $\boldsymbol{W}_3$ captures leftover batch effects and unknown unwanted variation.

4.4 Assessing the corrected data with LISI scores and UMAP representations

For the assessment, we employ the Local Inverse Simpson's Index (LISI) scores to examine the biological signal and leftover unwanted variation in the corrected data, and use UMAP plots for visualisation.

Firstly, the UMAP coordinates need to be derived before calculating the LISI scores and the visualisation.

```
# umap for raw
umap_raw_rna <- umap(log(rna_raw + 1))
umap_raw_adt <- umap(log(adt_raw + 1))
umap_raw_both <- umap(log(cbind(rna_raw, adt_raw) + 1))
# umap for logPAC
```

```
umap_logpac_rna <- umap(t(rna_logPAC))
umap_logpac_adt <- umap(t(adt_logPAC))
umap_logpac_both <- umap(cbind(t(rna_logPAC), t(adt_logPAC)))
# umap for PR
umap_pr_rna <- umap(t(rna_PearsonRes))
umap_pr_adt <- umap(t(adt_PearsonRes))
umap_pr_both <- umap(cbind(t(rna_PearsonRes), t(adt_PearsonRes)))

all.umap.ls <- list(umap_raw_both, umap_logpac_both, umap_pr_both)
names(all.umap.ls) <- c("raw", "logPAC", "PearsonResidual")
```

4.4.1 The LISI scores

The LISI score measures the diversity within each observation's local neighbourhood, and can be applied to evaluate the diversity of the neighbourhood around each cell. For cell i, LISI is defined as

$$LISI_i = \frac{1}{\lambda} = \frac{1}{\Sigma_{k=1}^{R} p_k^2},$$

where R is the total number of categories (e.g., cluster labels, cell annotation) and p_k is the proportional abundance of category k, and $\frac{1}{\lambda}$ ranges from 1 to R (inclusive). When all cells are in the same cluster, representing the smallest diversity of the neighbouring cells, the LISI score is 1. Higher LISI scores indicate higher local diversity of the data. The largest LISI score R represents the highest diversity of the neighbourhood.

```
ord.meta <- meta[match(rownames(all.umap.ls[[1]]), rownames(meta)),
    ]

## only calculate for big enough CTs
freq.tab <- data.frame(table(ord.meta$cellType_broad))
big.ct <- as.character(freq.tab$Var1[which(freq.tab$Freq > 150)])

extr.cell <- which(ord.meta$cellType_broad %in% big.ct)
meta <- ord.meta[extr.cell, ]

##
umap.ls <- list()
for (i in 1:length(all.umap.ls)) {
    umap.ls[[i]] <- all.umap.ls[[i]][extr.cell, ]
}
names(umap.ls) <- names(all.umap.ls)

## calculating LISI bio. LISI
lisi.bio.ls <- list()
for (i in 1:length(umap.ls)) {

    if (is.character(umap.ls[[i]])) {
        lisi.bio.ls[[i]] <- NA
    } else {
        lisi_bio <- compute_lisi(umap.ls[[i]], meta, "cellType_broad")
        lisi.bio.ls[[i]] <- lisi_bio
```

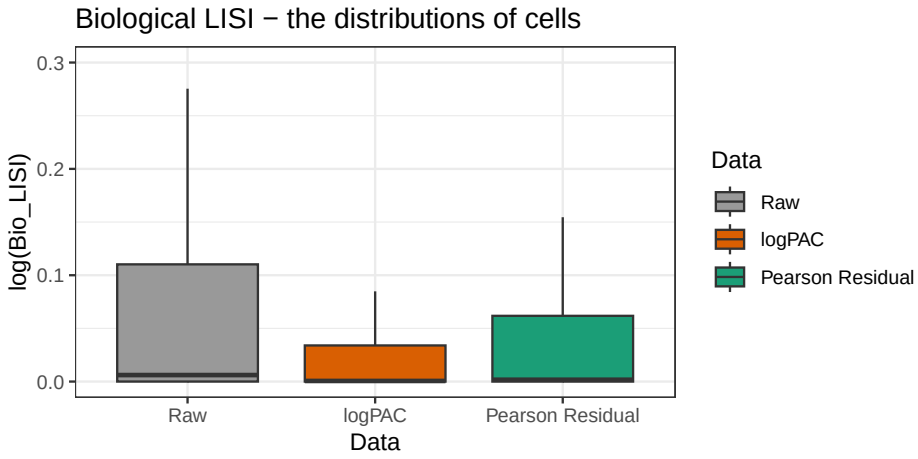
```
    }
}

### bat. LISI
lisi.bat.ls <- list()
for (i in 1:length(umap.ls)) {

    if (is.character(umap.ls[[i]])) {
        lisi.bat.ls[[i]] <- NA
    } else {
        lisi.bat.ls.ls <- list()
        for (z in 1:length(unique(meta$cellType_broad))) {
            umap.here <- umap.ls[[i]][which(meta$cellType_broad ==
                unique(meta$cellType_broad)[z]), ]
            meta.here <- meta[which(meta$cellType_broad ==
↪   unique(meta$cellType_broad)[z]),
                ]
            lisi_bat <- compute_lisi(umap.here, meta.here, "sample_batch")
            lisi.bat.ls.ls[[z]] <- lisi_bat
        }
        names(lisi.bat.ls.ls) <- unique(meta$cellType_broad)
        lisi.bat.ls[[i]] <- lisi.bat.ls.ls
    }
}

names(lisi.bio.ls) = names(lisi.bat.ls) <- names(umap.ls)
```

The LISI score for each cell allows us to explore the diversity of its neighbourhood. When the LISI metric for a cell reaches the number of the categories of interest (the upper bound), it represents the highest diversity of its neighbourhood achieved. In general, larger batch LISI scores for cells are expected from an effective normalisation method as it indicates that the local neighbours of each cell are of high diversity, suggesting that cells from different batches are well-mixed. On the other hand, smaller biological LISI scores are preferred for a good normalisation method because lower values represent the preservation of the biological structure in the corrected data.

Below are the distributions of the LISI scores for the Extended RUV-III-NB corrected data in comparison to the raw data, presented on the natural logarithm scale.

```
adjDat_col <- c("#999999", brewer.pal(3, "Dark2")[c(2, 1)])

## bio LISI
bio.lisi <- data.frame(Bio_LISI = c(lisi.bio.ls[[1]]$cellType_broad,
    lisi.bio.ls[[2]]$cellType_broad, lisi.bio.ls[[3]]$cellType_broad),
    Data = c(rep("Raw", length(lisi.bio.ls[[1]]$cellType_broad)),
        rep("logPAC", length(lisi.bio.ls[[2]]$cellType_broad)),
        rep("Pearson Residual", length(lisi.bio.ls[[3]]$cellType_broad))))
bio.lisi$Data <- factor(bio.lisi$Data, levels = c("Raw", "logPAC",
    "Pearson Residual"))
p1 <- ggplot(bio.lisi, aes(x = Data, y = log(Bio_LISI), fill = Data)) +
    ggtitle("Biological LISI - the distributions of cells") +
```

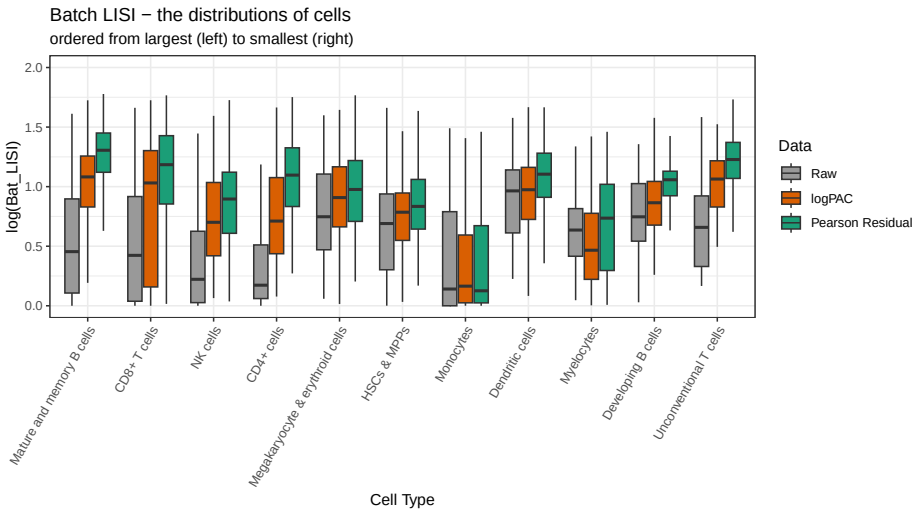
```
    geom_boxplot(outlier.shape = NA) + coord_cartesian(ylim = c(0,
    0.3)) + scale_fill_manual(values = adjDat_col) + theme_bw()
print(p1)
```



For better comparison, the biological LISI scores are shown on the natural logarithm scale. We expect low biological LISI scores for cells derived from the corrected data that was effectively normalised, indicating the preservation of biology. From the above figure, the distributions of the LISI scores from the corrected data (both logPAC and the Pearson residuals) are lower than those from the raw data ($log(raw.count + 1)$), suggesting successful retention and uncovering of biological signals.

```
adjDat_col <- c("#999999", brewer.pal(3, "Dark2")[c(2, 1)])

## bat LISI
dat.v <- c("Raw", "logPAC", "Pearson Residual")
lisi.bat.ls.redu <- list()
for (i in 1:length(lisi.bat.ls)) {
    for (j in 1:length(lisi.bat.ls[[i]])) {
        lisi.bat.ls[[i]][[j]]$Cell_Type <- names(lisi.bat.ls[[i]][j])
        lisi.bat.ls[[i]][[j]]$Data <- dat.v[i]
    }
    lisi.bat.ls.redu[[i]] <- do.call(rbind, lisi.bat.ls[[i]])
}

bat.lisi <- do.call(rbind, lisi.bat.ls.redu)
colnames(bat.lisi)[1] <- "Bat_LISI"
bat.lisi$Data <- factor(bat.lisi$Data, levels = c("Raw", "logPAC",
    "Pearson Residual"))
# ordering by the size of the clusters
bat.lisi$Cell_Type <- factor(bat.lisi$Cell_Type, levels =
↪  data.frame(table(meta$cellType_broad))[order(data.frame(table(meta$cellType_broad))$Freq,
    decreasing = T), "Var1"])
p2 <- ggplot(bat.lisi, aes(x = Cell_Type, y = log(Bat_LISI),
    fill = Data)) + ggtitle("Batch LISI - the distributions of cells") +
    labs(x = "Cell Type", subtitle = "ordered from largest (left) to smallest
    ↪  (right)") +
```

```
    geom_boxplot(outlier.shape = NA) + coord_cartesian(ylim = c(0,
    2)) + scale_fill_manual(values = adjDat_col) + theme_bw() +
    theme(axis.text.x = element_text(angle = 60, vjust = 1, hjust = 1))
print(p2)
```



For comparison, the batch LISI scores are shown on the natural logarithm scale. High batch LISI scores are anticipated for cells derived from the corrected data from an effective normalisation, indicating a well-mixed representation of neighbouring cells from different batches. From the above figure, the distributions of the LISI scores from the corrected data (both logPAC and the Pearson residuals) are higher than those from the raw data ($log(raw.count + 1)$), demonstrating successful mitigation of unwanted variation (batch effect, in this case).

4.4.2 The UMAP plots

We firstly form a list to convenient the plotting of several figure panels.

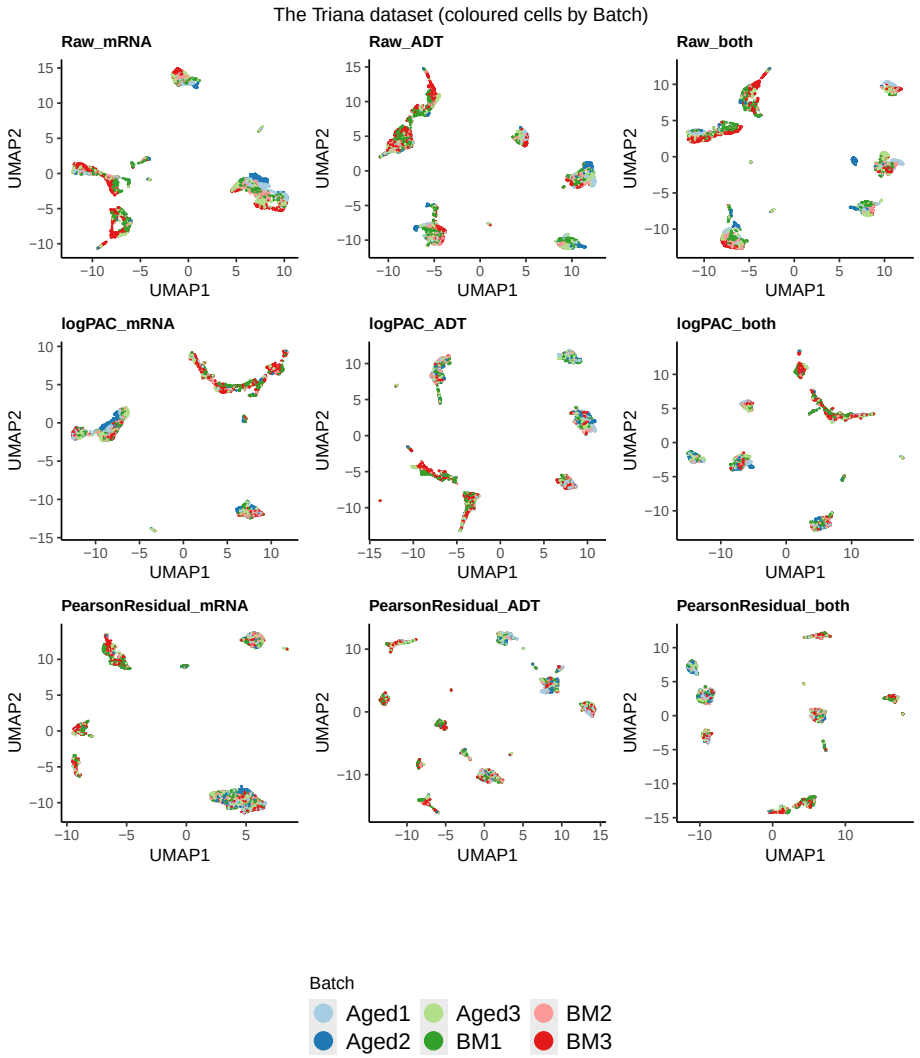
```
plot.umap <- list(umap_raw_rna, umap_raw_adt, umap_raw_both,
    umap_logpac_rna, umap_logpac_adt, umap_logpac_both, umap_pr_rna,
    umap_pr_adt, umap_pr_both)
names(plot.umap) <- c(paste0("Raw", c("_mRNA", "_ADT", "_both")),
    paste0("logPAC", c("_mRNA", "_ADT", "_both")), paste0("PearsonResidual",
        c("_mRNA", "_ADT", "_both")))
```

The figure panels below display $log(raw + 1)$, $log(PAC + 1)$ (logPAC), and Pearson residuals arranged in rows, with UMAP representations derived from mRNA features only, protein (ADT) features only, and both domains combined in the columns.

We firstly colour the cells by the batch they came from.

```
mycolors <- c(brewer.pal(name = "Paired", n = 12), brewer.pal(name = "Dark2",
    n = 8))

# metadata for all cells in the build-in dataset
meta <- triana_data$metadata
## batch
```

```
p.ls <- list()
for (i in 1:length(plot.umap)) {
    ord.meta <- meta[match(rownames(plot.umap[[i]]), rownames(meta)),
        ]
    df.graph <- data.frame(plot.umap[[i]], Batch = ord.meta$sample_batch)

    colnames(df.graph)[1:2] <- c("UMAP1", "UMAP2")
    p.ls[[i]] <- ggplot(df.graph, mapping = aes(x = UMAP1, y = UMAP2,
        color = Batch)) + theme_classic() + ggtitle(names(plot.umap)[i]) +
        guides(colour = guide_legend(override.aes = list(size = 5))) +
        geom_point(size = 0.1) + theme(plot.title = element_text(size = 10,
        face = "bold"), legend.position = "none") + scale_color_manual(values
        ↪  = mycolors)
}
p.leg <- ggplot(df.graph, mapping = aes(x = UMAP1, y = UMAP2,
    color = Batch)) + theme(legend.text = element_text(size = 14)) +
    ggtitle(names(plot.umap)[i]) + guides(colour = guide_legend(ncol = 3,
    override.aes = list(size = 5))) + geom_point(size = 0.1) +
    scale_color_manual(values = mycolors)
legend <- cowplot::get_legend(p.leg)
#> Warning in get_plot_component(plot, "guide-box"): Multiple components
↪  found;
#> returning the first one. To return all, use `return_all = TRUE`.
p.ls[[i + 2]] <- legend

grid.arrange(grobs = p.ls, ncol = 3, top = "The Triana dataset (coloured cells
↪  by Batch)")
```

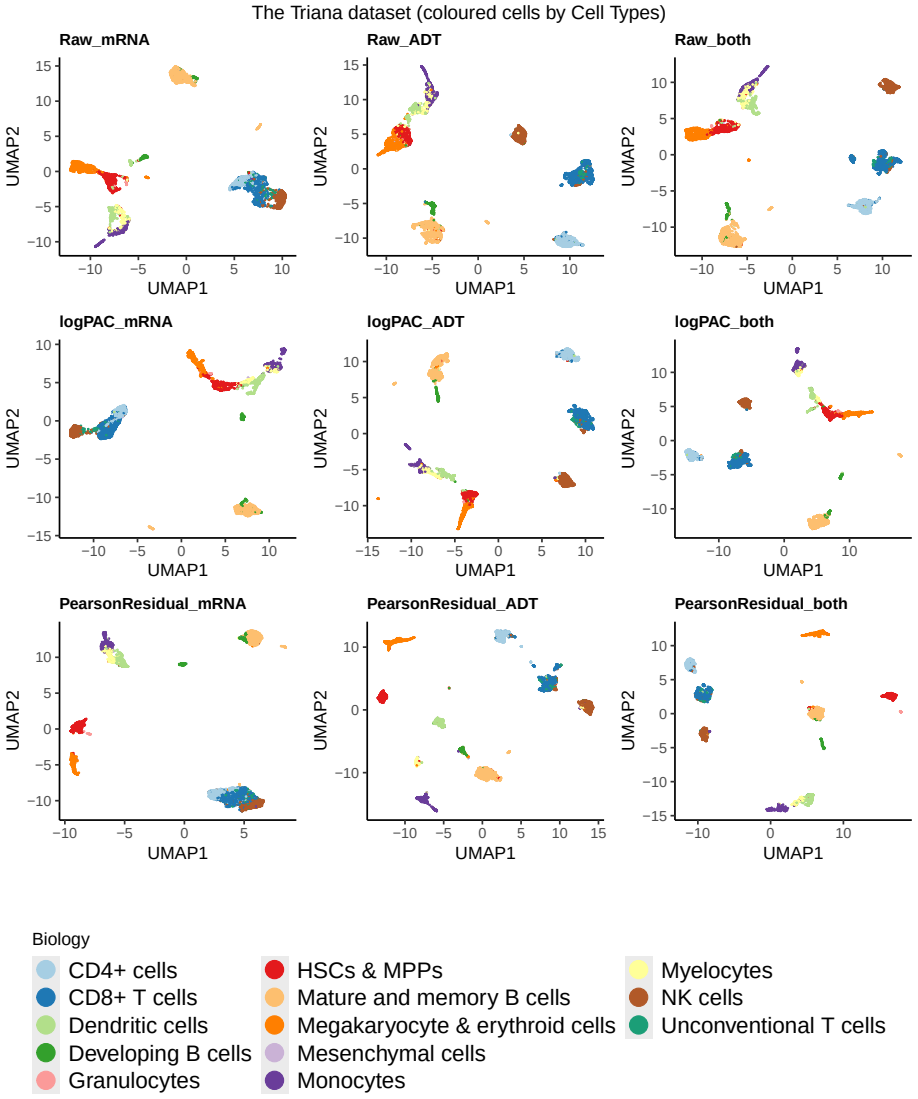


For each cluster of cells, there is a slightly better mixture of cells from different batches in the middle and bottom rows of the panels, compared to the top panels, which represent the uncorrected data.

Secondly, we coloured the cells by the cell type annotation from the data publisher.

```
mycolors <- c(brewer.pal(name = "Paired", n = 12), brewer.pal(name = "Dark2",
    n = 8))

# metadata for all cells in the build-in dataset
meta <- triana_data$metadata
## biology
p.ls <- list()
for (i in 1:length(plot.umap)) {
    ord.meta <- meta[match(rownames(plot.umap[[i]]), rownames(meta)),
        ]
    df.graph <- data.frame(plot.umap[[i]], Biology = ord.meta$cellType_broad)
```

```
        colnames(df.graph)[1:2] <- c("UMAP1", "UMAP2")
        p.ls[[i]] <- ggplot(df.graph, mapping = aes(x = UMAP1, y = UMAP2,
            color = Biology)) + theme_classic() + ggtitle(names(plot.umap)[i]) +
            guides(colour = guide_legend(override.aes = list(size = 5))) +
            geom_point(size = 0.1) + theme(plot.title = element_text(size = 10,
            face = "bold"), legend.position = "none") + scale_color_manual(values
            ↪  = mycolors)
    }
    p.leg <- ggplot(df.graph, mapping = aes(x = UMAP1, y = UMAP2,
        color = Biology)) + theme(legend.text = element_text(size = 14)) +
        ggtitle(names(plot.umap)[i]) + guides(colour = guide_legend(ncol = 3,
        override.aes = list(size = 5))) + geom_point(size = 0.1) +
        scale_color_manual(values = mycolors)
    legend <- cowplot::get_legend(p.leg)
    #> Warning in get_plot_component(plot, "guide-box"): Multiple components
    ↪  found;
    #> returning the first one. To return all, use `return_all = TRUE`.
    p.ls[[i + 2]] <- legend

    grid.arrange(grobs = p.ls, ncol = 3, top = "The Triana dataset (coloured cells
    ↪  by Cell Types)")
```



Overall, the biological structure is visually obvious in the UMAP representations using the raw counts (the first row of the panels), while the corrected data (the middle and bottom rows) retain this structure with slight refinement. It is particularly evident in the improved separation of Monocytes (purple cells in the figure) and Myelocytes (light yellow ones) in the middle and the bottom panels.

4.5 Differential Expression (DE) analysis

In this subsection, we perform DE analysis to assess the corrected data at the feature-level.

4.5.1 Differentially expressed features of unwanted variation

We investigate the features that are statistically significantly affected by batch factors. To prevent biological effects from influencing the assessment of the batch DE analysis, we stratified the cells by cell-types, and retained only those with more than 150 cells. For each feature, we performed the Kruskal-Wallis Rank Sum Test (K-W test) (cite), as this non-parametric method prevents the testing method in favour of any data having a similar characteristic that

aligns with the assumptions of the statistical testing. The p-values for all features obtained from the K-W tests were adjusted by the Benjamini-Hochberg procedure (cite 1995) to control the false discovery rate.

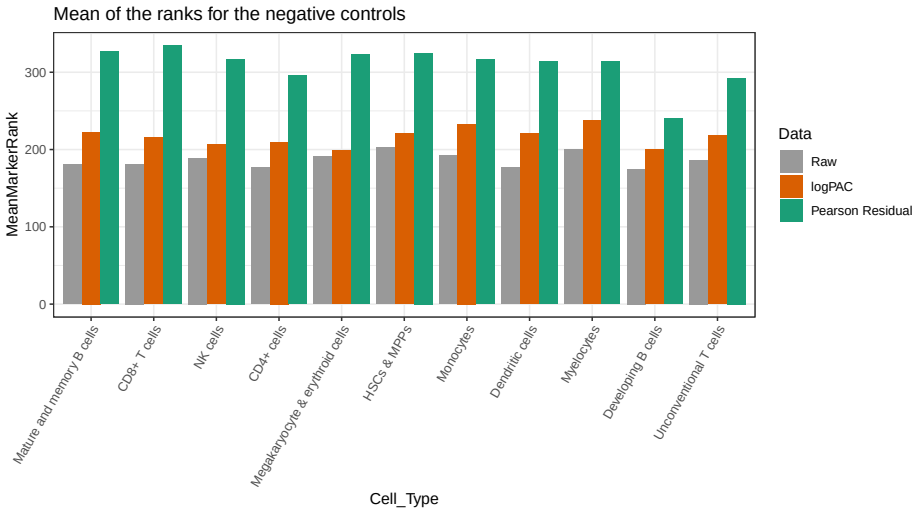
The negative controls are defined as the "markers" for assessing unwanted variation in the corrected data due to their characteristic of being affected by technical effects. Next, we transform the adjusted p-values of all features into ranks. Then, we calculate the mean of the ranks of the negative controls.

```
dat.ls <- list(raw = log(cbind(rna_raw, adt_raw) + 1), logPAC =
↪  cbind(t(rna_logPAC),
    t(adt_logPAC)), PearsonRes = cbind(t(rna_PearsonRes), t(adt_PearsonRes)))
nc_adjPVAL_testing.ls = nc_MeanMarkerRank_testing.ls <- list()
for (i in 1:3) {
    meta <- triana_data$metadata
    dat.here <- dat.ls[[i]]

    ord.meta <- meta[match(rownames(dat.here), rownames(meta)),
        ]

    ## only calculate for big enough CTs
    freq.tab <- data.frame(table(ord.meta$cellType_broad))
    big.ct <- as.character(freq.tab$Var1[which(freq.tab$Freq >
        150)])

    extr.cell <- which(ord.meta$cellType_broad %in% big.ct)

    meta.nc <- ord.meta[extr.cell, ]
    dat.here.nc <- dat.here[extr.cell, ]
    ##

    NC_inner_ADJpval = NC_inner_MeanMarkerRank <- list()
    for (ct in 1:length(unique(meta.nc$cellType_broad))) {
        meta.here <- meta.nc[which(meta.nc$cellType_broad ==
            unique(meta.nc$cellType_broad)[ct]), ]
        dat.here.here <- dat.here.nc[which(meta.nc$cellType_broad ==
            unique(meta.nc$cellType_broad)[ct]), ]

        dat.here.here.ls <- lapply(seq_len(ncol(dat.here.here)),
            function(i) dat.here.here[, i])
        names(dat.here.here.ls) <- colnames(dat.here.here)

        nc_PVAL_testing <- mclapply(dat.here.here.ls, function(z) {
            test <- kruskal.test(z ~ meta.here$sample_batch)
            orig.pval <- test$p.value
            adj.pval <- p.adjust(orig.pval, method = "BH")
            # Benjamini & Hochberg (1995) ('BH' or its
            # alias 'fdr')
            out.ls <- list(orig.pval, adj.pval)
            return(out.ls)
        }, mc.cores = 2L)
```

```
        # orig.pval.v <- sapply(nc_PVAL_testing, '[[', 1)
        adj.pval.v <- sapply(nc_PVAL_testing, "[[", 2)

        NC_inner_ADJpval[[ct]] <- adj.pval.v

        rank.adj.pvals <- rank(adj.pval.v)
        marker.rank <- rank.adj.pvals[match(c(nc1, nc2, nc3),
            names(rank.adj.pvals))]
        NC_inner_MeanMarkerRank[[ct]] <- mean(marker.rank)
    }
    names(NC_inner_ADJpval) = names(NC_inner_MeanMarkerRank) <-
    ↪  unique(meta.nc$cellType_broad)

    nc_adjPVAL_testing.ls[[i]] <- NC_inner_ADJpval
    nc_MeanMarkerRank_testing.ls[[i]] <- NC_inner_MeanMarkerRank
}
names(nc_adjPVAL_testing.ls) = names(nc_MeanMarkerRank_testing.ls) <-
↪  names(dat.ls)
```

Since we expect that the unwanted variation has been properly removed from the negative controls, a successful corrected data is anticipated to result in a higher mean of the ranks for these markers, indicating larger adjusted p-values of the negative controls.

```
adjDat_col <- c("#999999", brewer.pal(3, "Dark2")[c(2, 1)])

df.graph <- data.frame(MeanMarkerRank = unlist(do.call("c",
↪  nc_MeanMarkerRank_testing.ls)),
    Data = sub("\\..*", "", names(unlist(do.call("c",
    ↪  nc_MeanMarkerRank_testing.ls)))),
    Cell_Type = sub("^[^.]+.", "", names(unlist(do.call("c",
        nc_MeanMarkerRank_testing.ls)))))
df.graph$Data[df.graph$Data == "raw"] <- "Raw"
df.graph$Data[df.graph$Data == "PearsonRes"] <- "Pearson Residual"

df.graph$Data <- factor(df.graph$Data, levels = c("Raw", "logPAC",
    "Pearson Residual"))
orderedCT <-
↪  data.frame(table(meta$cellType_broad))[order(data.frame(table(meta$cellType_broad))$Freq,
    decreasing = T), "Var1"]
df.graph$Cell_Type <- factor(df.graph$Cell_Type, levels = orderedCT)

p <- ggplot(df.graph, aes(x = Cell_Type, y = MeanMarkerRank,
    fill = Data)) + geom_col(position = "dodge") + ggtitle("Mean of the ranks
    ↪  for the negative controls") +
    scale_fill_manual(values = adjDat_col) + theme_bw() + theme(axis.text.x =
    ↪  element_text(angle = 60,
    vjust = 1, hjust = 1))
print(p)
```



The barplot shows higher means of the ranks for negative controls from the corrected data, suggesting that they return larger adjusted p-values compared to the raw data.

4.5.2 Differentially expressed biological features

We examine the features that contain useful biological information. Here, the top 3 cell types with the highest cell count are used for the assessment (`Mature and memory B cells`, `CD8+ T cells`, and `NK cells`). In the DE analysis for this cell-type, cells are relabelled as `The cell type of interest` or `others`. To avoid the leftover unwanted variation from affecting the assessment of the biological DE analysis, we conduct the analysis batch by batch and combine p-values from different batches In each batch, we perform the Wilcoxon Rank Sum Test (two-tailed) (cite) for each feature, and then combine the unadjusted p-values across batches using the Fisher method (cite). The combined p-values are then adjusted by the Benjamini-Hochberg procedure to control the false discovery rate.

The markers for cell-type identification are defined as positive controls because they are useful for assessing biological variation in the corrected data. To incorporate the knowledge of positive controls for further assessment, we take the adjusted p-values and transform them into ranks. The summary of the ranks for the positive controls is obtained by calculating the mean of the ranks across the markers.

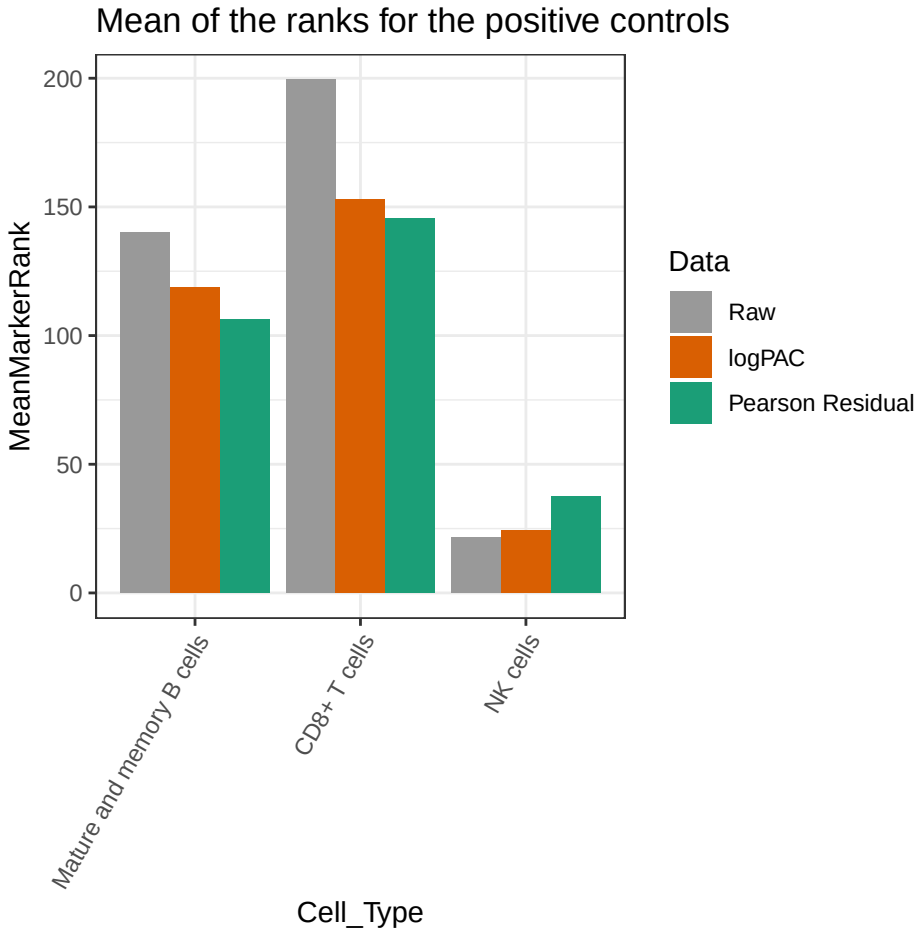
```
dat.ls <- list(raw = log(cbind(rna_raw, adt_raw) + 1), logPAC =
↪  cbind(t(rna_logPAC),
    t(adt_logPAC)), PearsonRes = cbind(t(rna_PearsonRes), t(adt_PearsonRes)))

markers <- list(B = c("mod1_CD19", "mod1_CD20", "mod1_CD27",
    "mod1_CD38", "mod1_IgD", "mod1_IgM", "mod1_IgG", "mod1_IgA",
    "mod1_IgE", "mod1_CD24", "mod1_CD138", "mod1_Syndecan-1",
    "mod1_IgG1", "mod1_IgG2a", "mod1_CD138", "mod2_CD19", "mod2_CD20",
    "mod2_CD27", "mod2_CD38", "mod2_IgD", "mod2_IgM", "mod2_IgG",
    "mod2_IgA", "mod2_IgE", "mod2_CD24", "mod2_CD138", "mod2_Syndecan-1",
    "mod2_IgG1", "mod2_IgG2a", "mod2_CD138"), CD8T = c("mod1_CD3",
    "mod1_CD8", "mod1_CD45RO", "mod1_CD45RA", "mod1_CCR7", "mod1_CD62L",
    "mod1_L-selectin", "mod1_Granzyme B", "mod1_Perforin", "mod1_CD45RO-Bio",
    "mod1_CD45RA-Bio", "mod1_CD62L", "mod2_CD3", "mod2_CD8",
    "mod2_CD45RO", "mod2_CD45RA", "mod2_CCR7", "mod2_CD62L",
```

```
        "mod2_L-selectin", "mod2_Granzyme B", "mod2_Perforin", "mod2_CD45RO-Bio",
        "mod2_CD45RA-Bio", "mod2_CD62L"), NK = c("mod1_CD56", "mod1_NCAM",
        "mod1_CD16", "mod1_CD3", "mod1_NKp46", "mod1_NCR1", "mod1_NKG2D",
        "mod1_KLRK1", "mod1_CD94", "mod1_KLRD1", "mod1_CD57", "mod2_CD56",
        "mod2_NCAM", "mod2_CD16", "mod2_CD3", "mod2_NKp46", "mod2_NCR1",
        "mod2_NKG2D", "mod2_KLRK1", "mod2_CD94", "mod2_KLRD1", "mod2_CD57"))

    pc_combineAdjPVAL.ls = pc_MeanMarkerRank_testing.ls <- list()
    cts <- c("Mature and memory B cells", "CD8+ T cells", "NK cells")
    for (i in 1:3) {
        meta <- triana_data$metadata
        dat.here <- dat.ls[[i]]

        ord.meta <- meta[match(rownames(dat.here), rownames(meta)),
            ]

        PC_inner_combineAdjPVAL = PC_inner_MeanMarkerRank <- list()
        for (ct in 1:3) {
            cluster.nam <- cts[ct]

            ## analyse batch by batch and integrate
            bat.pval.mt <- matrix(NA, nrow = length(unique(meta$sample_batch)),
                ncol = ncol(dat.here))
            colnames(bat.pval.mt) <- colnames(dat.here)
            pc_inbat_ls <- list()
            for (b in 1:length(unique(meta$sample_batch))) {

                dat.here.bat <- dat.here[which(meta$sample_batch ==
                    unique(meta$sample_batch)[b]), ]
                meta.here.bat <- meta[which(meta$sample_batch ==
                    unique(meta$sample_batch)[b]), ]
                # within each batch do
                {
                    extr <- which(meta.here.bat$cellType_broad ==
                      cluster.nam)

                    dat.here.bat.ls <- lapply(seq_len(ncol(dat.here.bat)),
                      function(i) dat.here.bat[, i])
                    names(dat.here.bat.ls) <- colnames(dat.here.bat)

                    pc_PVAL_testing <- mclapply(dat.here.bat.ls,
                      function(z) {
                        test <- wilcox.test(x = z[extr], y = z[-extr],
                          alternative = "two.sided")
                        orig.pval <- test$p.value
                        # adj.pval <- p.adjust(orig.pval,
                        # method = 'BH') Benjamini & Hochberg
                        # (1995) ('BH' or its alias 'fdr')
                        out.ls <- list(orig.pval)
                        return(out.ls)
```

```
                    }, mc.cores = 2L)

                orig.pval.v <- sapply(pc_PVAL_testing, "[[",
                  1)

                pc_inbat_ls[[b]] <- orig.pval.v
                bat.pval.mt[b, ] <- orig.pval.v  #for later combination use

                # https://www.biostars.org/p/338286/
                # combine (using Fisher's method) then
                # adjust
            }
        }
        names(pc_inbat_ls) <- unique(meta$sample_batch)

        bat.pval.mt.ls <- lapply(seq_len(ncol(bat.pval.mt)),
            function(i) bat.pval.mt[, i])
        names(bat.pval.mt.ls) <- colnames(bat.pval.mt)

        combinePVAL <- mclapply(bat.pval.mt.ls, function(x) {
            # x <- bat.pval.mt[,2]
            pval.ls <- as.list(x)
            result <- metapod::combineParallelPValues(pval.ls,
                method = "fisher")
            comb.orig.pval <- result$p.value
            comb.adj.pval <- p.adjust(comb.orig.pval, method = "BH")
            out.ls <- list(comb.orig.pval, comb.adj.pval)
            return(out.ls)
        }, mc.cores = 2L)

        adj.pval.v <- sapply(combinePVAL, "[[", 2)

        PC_inner_combineAdjPVAL[[ct]] <- adj.pval.v

        rank.adj.pvals <- rank(adj.pval.v)
        # take the intersection for matching
        (intersect_marker <- intersect(markers[[ct]], names(rank.adj.pvals)))
        marker.rank <- rank.adj.pvals[match(intersect_marker,
            names(rank.adj.pvals))]
        PC_inner_MeanMarkerRank[[ct]] <- mean(marker.rank)
    }

    names(PC_inner_combineAdjPVAL) = names(PC_inner_MeanMarkerRank) <- cts

    pc_combineAdjPVAL.ls[[i]] <- PC_inner_combineAdjPVAL
    pc_MeanMarkerRank_testing.ls[[i]] <- PC_inner_MeanMarkerRank
}
names(pc_combineAdjPVAL.ls) = names(pc_MeanMarkerRank_testing.ls) <-
↪  names(dat.ls)
```

Effectively normalised data is expected to return smaller mean of the marker ranks because they reflect smaller p-values of the positive controls from the normalisation.

```
adjDat_col <- c("#999999", brewer.pal(3, "Dark2")[c(2, 1)])

df.graph <- data.frame(MeanMarkerRank = unlist(do.call("c",
↪  pc_MeanMarkerRank_testing.ls)),
    Data = sub("\\..*", "", names(unlist(do.call("c",
    ↪  pc_MeanMarkerRank_testing.ls)))),
    Cell_Type = sub("^[^.]+.", "", names(unlist(do.call("c",
        pc_MeanMarkerRank_testing.ls)))))
df.graph$Data[df.graph$Data == "raw"] <- "Raw"
df.graph$Data[df.graph$Data == "PearsonRes"] <- "Pearson Residual"

df.graph$Data <- factor(df.graph$Data, levels = c("Raw", "logPAC",
    "Pearson Residual"))
df.graph$Cell_Type <- factor(df.graph$Cell_Type, levels = cts)

p <- ggplot(df.graph, aes(x = Cell_Type, y = MeanMarkerRank,
    fill = Data)) + geom_col(position = "dodge") + ggtitle("Mean of the ranks
    ↪  for the positive controls") +
    scale_fill_manual(values = adjDat_col) + theme_bw() + theme(axis.text.x =
    ↪  element_text(angle = 60,
    vjust = 1, hjust = 1))
print(p)
```



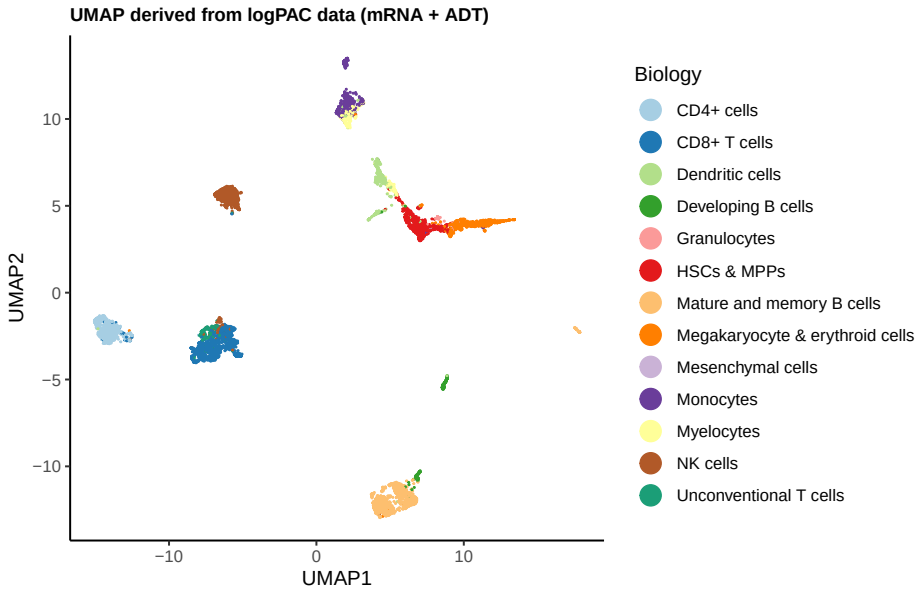
The barplot shows lower means of the ranks for positive controls of `Mature and memory B cells` and `CD8+ T cells` from the corrected data, suggesting that they return smaller adjusted p-values compared to the raw data. This indicates that biological signals can be better revealed after data normalisation. While the signal in the markers for identifying `NK cells` was not improved after normalisation, the difference in `MeanMarkerRank` value between `Data` is less pronounced.

4.5.2.1 Marker expressions on UMAP representations

Based on the UMAP representation derived from logPAC data, we observe that the cell types are distinctly clustered.

```
meta <- triana_data$metadata
mycolors <- c(brewer.pal(name = "Paired", n = 12), brewer.pal(name = "Dark2",
    n = 8))

ord.meta <- meta[match(rownames(plot.umap[[i]]), rownames(meta)),
    ]
df.graph <- data.frame(umap_logpac_both, Biology = ord.meta$cellType_broad)

colnames(df.graph)[1:2] <- c("UMAP1", "UMAP2")
p <- ggplot(df.graph, mapping = aes(x = UMAP1, y = UMAP2, color = Biology)) +
    theme_classic() + ggtitle("UMAP derived from logPAC data (mRNA + ADT)") +
    guides(colour = guide_legend(override.aes = list(size = 5))) +
```

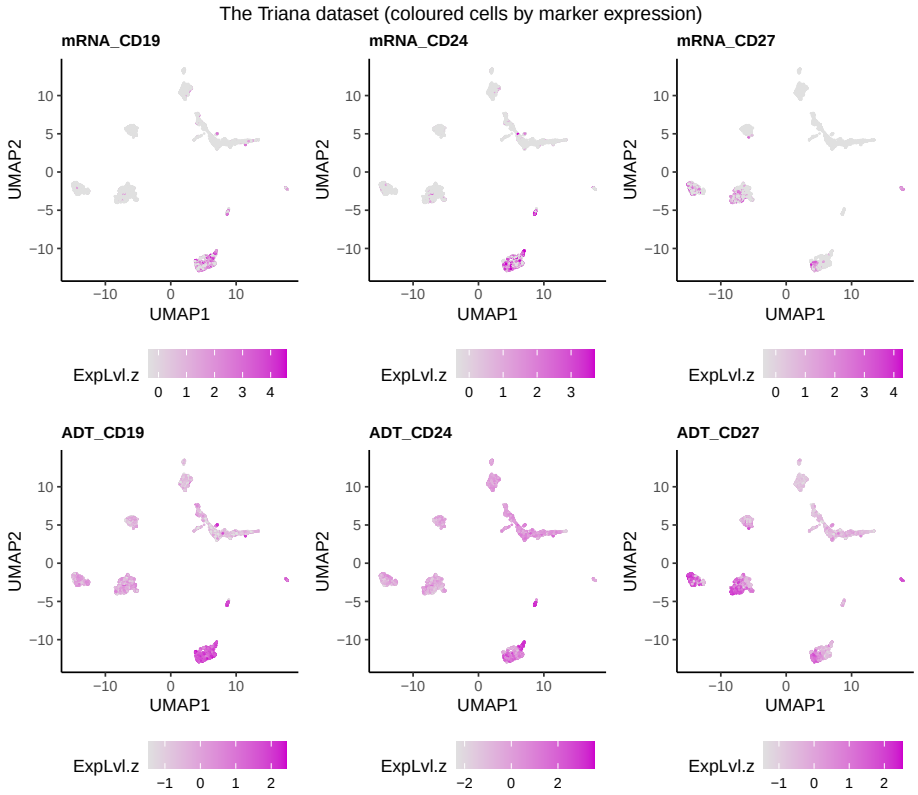
```
    geom_point(size = 0.1) + theme(plot.title = element_text(size = 10,
    face = "bold"), legend.position = "right") + scale_color_manual(values =
    ↪  mycolors)
print(p)
```



Then, we focus on `Mature and memory B cells`, and examine the expression levels of their markers in both mRNA and ADT domains. Specifically, we analyse the expressions of CD19 (a pan-B cell marker present on all B cells), CD24 (expressed on naïve B cells and some memory B cells), and CD27 (a marker of memory B cells).

```
dat <- cbind(t(rna_logPAC), t(adt_logPAC))

markers <- c("mod1_CD19", "mod1_CD24", "mod1_CD27", #mRNA domain
             "mod2_CD19", "mod2_CD24", "mod2_CD27") #protein domain
markers.title <- c("mRNA_CD19", "mRNA_CD24", "mRNA_CD27",
                   "ADT_CD19", "ADT_CD24", "ADT_CD27")
p.ls <- list()
for(i in 1:length(markers)){
  df.graph <- data.frame(umap_logpac_both, dat[,which(colnames(dat) ==
↪  markers[i])])
  colnames(df.graph) <- c("UMAP1", "UMAP2", markers[i])

  p.ls[[i]] <- local({
    i <- i
    ExpLvl.z <- (df.graph[,3]-mean(df.graph[,3]))/sd(df.graph[,3])

    ggplot(df.graph, mapping = aes(x=UMAP1, y=UMAP2, color=ExpLvl.z)) +
    theme_classic() +
    ggtitle(markers.title[i]) +
    geom_point(size=0.1) +
    theme(plot.title = element_text(size = 10, face = "bold"), legend.position
    ↪  = "bottom") +
```

```
    scale_color_gradient(low = "gray88", high = "magenta3")
  })
}

grid.arrange(grobs = p.ls, ncol = 3,
             top = "The Triana dataset (coloured cells by marker expression)")
```



We do see higher expression levels of the mRNA and ADT markers for the `Mature and memory B cells` cluster in the bottom of each panel. However, the surface protein markers in the bottom panels tend to express in a broader range of cell types beyond just `Mature and memory B cells`.

5 Summary

The Extended RUV-III-NB model has effectively reduced unwanted variation and enhanced biological signals. Additionally, the corrected data allows for feature-level analyses, including differential expression (DE) analysis.