Correcting for age in differential expression analysis

RAPToR 1.2.0

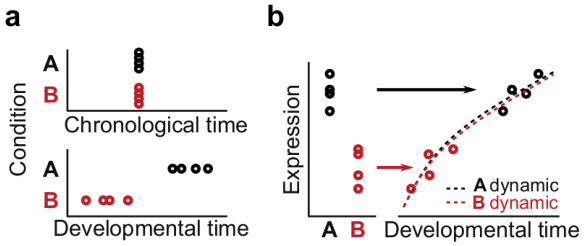
Romain Bulteau

March 2023

Contents

1 Introduction

The developmental speed of fast-growing organisms such as *C. elegans* is influenced by many different factors – including experimental conditions themselves – making it difficult or impossible to collect age-matched individuals between conditions. For example, if a mutant has delayed development but controls and mutants are collected at the same chronological (and therefore different physiological) time, the perturbation of interest will be completely confounded with development (**a**). Because of this, purely developmental gene expression changes can be mislabeled as the effect of a variable of interest (**b**).



When sample groups have age differences but still overlap, the developmental effect can simply be accounted for by including age as a covariate in the Differential Expression (DE) analysis. If there is no age overlap however, it becomes impossible to know whether an observed effect is due to the perturbation or age since both variables are completely confounded.

Using `RAPToR` reference data, we can bridge the gap between non-overlapping sample groups and rescue otherwise impossible DE analyses.

2 Quickstart

2.1 Quantifying age-driven expression changes with `ref_compare()`

```
# load libraries
library(RAPToR)
library(wormRef)
```

Given transcript per million (TPM) expression profiles from 3 control and 3 mutant samples.

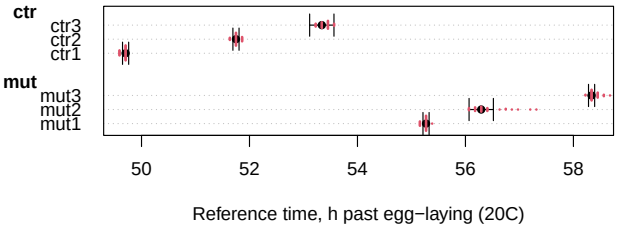
```
head(qs$tpm)
#>                    ctr1     ctr2     ctr3     mut1     mut2     mut3
#> WBGene00000001 2.862195 3.118323 3.175077 3.340883 3.426089 3.546783
#> WBGene00000002 3.107584 2.579527 2.156750 1.783193 1.843059 1.780094
#> WBGene00000003 2.586109 3.101123 3.152630 3.335000 3.244302 2.833467
#> WBGene00000004 1.930230 1.801899 1.706351 1.938189 2.018981 2.191545
#> WBGene00000005 2.866435 2.483658 2.125052 2.360653 2.545615 2.647073
#> WBGene00000006 1.583126 1.693566 1.873643 2.296500 2.266956 2.017384
print(qs$pdat)
#>   sname strain
#> 1  ctr1    ctr
#> 2  ctr2    ctr
#> 3  ctr3    ctr
#> 4  mut1    mut
```

```
#> 5  mut2    mut
#> 6  mut3    mut
```

First, infer sample age with RAPToR.

```
# load reference
r_cel <- prepare_refdata("Cel_larv_YA", "wormRef", 600)

# estimate sample age
ae_qs <- ae(qs$tpm, r_cel)
```

```
plot(ae_qs, group = qs$pdat$strain, g.line=3, lmar=5)
```

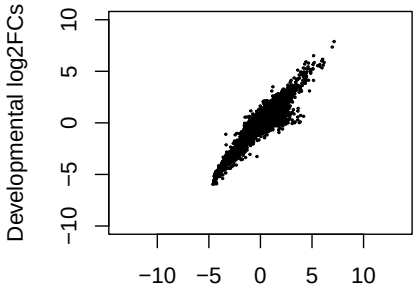


Run `ref_compare()` to estimate developmental expression changes between groups.

```
qs_rc <- ref_compare(
    X      = qs$tpm,        # sample data, log(TPM+1)
    ref    = r_cel,         # ref object
    ae_obj = ae_qs,         # ae object
    group  = qs$pdat$strain # factor defining compared groups (wt/mut)
)
```

Plot sample log2 fold-changes (logFCs) vs. expected developmental logFCs inferred from the reference.

```
qs_lfc <- get_logFC(qs_rc)
#> Comparing ctr vs. mut

plot(qs_lfc, cex=.2, asp=1,
     xlab = "ctr vs. mut log2FCs", xlim = c(-10,10),
     ylab = "Developmental log2FCs", ylim = c(-10, 10))
```



Print the correlation between sample and reference logFCs, as well as the average age difference between groups.

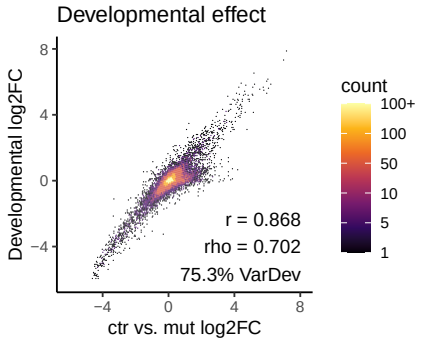
```
print(qs_rc)
#> DE comparison with reference
#> ---
#>                 ref.logFC.r ref.logFC.r2 ae.avg.dif
#> ctr (ctrl, n=3)          NA           NA         NA
#> mut (n=3)             0.868        0.753      5.035
#> ---
```

In this example, ctr *vs.* mut logFCs have a correlation of 0.868 with the expected developmental logFCs computed from the reference data. Using r^2, this corresponds to approximately 75.3% of variance explained by development. Such a large developmental effect between the control and mutant groups is expected as they don't have any age overlap.

For an explanation of the analysis, see *How it works*.

A custom `ggplot` function to plot logFCs is also provided in *Plotting functions*, below.

```
gg_logFC(qs_lfc, main = "Developmental effect",
         xlab = "ctr vs. mut log2FC", ylab = "Developmental log2FC")
```



2.2 Correcting for age effects in differential expression analysis

```
# load libraries
library(RAPToR)
library(wormRef)

library(DESeq2)
library(splines)
```

Given raw counts and transcript per million (TPM) expression profiles from 3 control and 3 mutant samples.

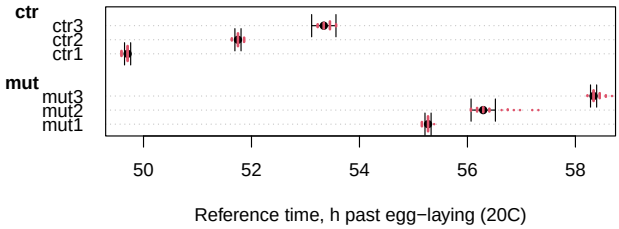
```
head(qs$count)
#>                ctr1 ctr2 ctr3 mut1 mut2 mut3
#> WBGene00000001 1000 1266 1843 1984 2310 2540
#> WBGene00000002 1387  765  658  386  442  398
#> WBGene00000003  620 1036 1500 1643 1594 1005
#> WBGene00000004  531  441  539  644  754  891
#> WBGene00000005 1361  872  803  947 1236 1339
#> WBGene00000006  423  469  799 1174 1211  886
head(qs$tpm)
#>                   ctr1     ctr2     ctr3     mut1     mut2     mut3
```

```
#> WBGene00000001 2.862195 3.118323 3.175077 3.340883 3.426089 3.546783
#> WBGene00000002 3.107584 2.579527 2.156750 1.783193 1.843059 1.780094
#> WBGene00000003 2.586109 3.101123 3.152630 3.335000 3.244302 2.833467
#> WBGene00000004 1.930230 1.801899 1.706351 1.938189 2.018981 2.191545
#> WBGene00000005 2.866435 2.483658 2.125052 2.360653 2.545615 2.647073
#> WBGene00000006 1.583126 1.693566 1.873643 2.296500 2.266956 2.017384
print(qs$pdat)
#>   sname strain
#> 1  ctr1    ctr
#> 2  ctr2    ctr
#> 3  ctr3    ctr
#> 4  mut1    mut
#> 5  mut2    mut
#> 6  mut3    mut
```

First, infer sample age with RAPToR.

```
# load reference
r_cel <- prepare_refdata("Cel_larv_YA", "wormRef", 600)

# estimate sample age
ae_qs <- ae(qs$tpm, r_cel)
```

```
plot(ae_qs, group = qs$pdat$strain, g.line=3, lmar=5)
```

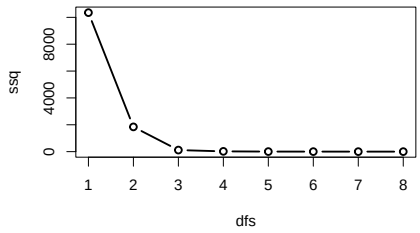


Define a reference window to integrate.

```
r.window <- range(ae_qs$age.estimates[,1]) + c(1,-1) # +1h margin
r.idx <- get_refTP(r_cel, ae_values = r.window) # indices in the ref object
r.idx <- r.idx[1]:r.idx[2]

# extract time and expression values
r.time <- r_cel$time[r.idx]
r.tpm <- r_cel$interpGE[,r.idx]
```

Find the optimal spline degree-of-freedom (df) to model expression dynamics in the selected window.

```
dfs <- 1:8
ssq <- sapply(dfs, function(i){
  # compute SSQ of linear model on expression data
  sum(residuals(lm( t(r.tpm) ~ splines::ns(r.time, df=i) ))^2)
})

plot(dfs, ssq, type='b', lwd=2)
```




```
r.df <- 3 # select df=3
```

Transform reference TPMs into artificial counts using an arbitrary library size.

```
# get gene lengths
g.l <- wormRef::Cel_genes[match(rownames(r.tpm), wormRef::Cel_genes[,"wb_id"]),
                          "transcript_length"]
libsize <- 25e6

# convert tpm to artificial counts
r.count <- t( (t(exp(r.tpm)-1)/1e6) * (libsize/median(g.l))) * g.l
r.count[r.count < 0] <- 0
r.count <- round(r.count)
```

Combine sample and reference count matrices and pdata.

```
# filter low expressed genes (at least >5 in one sample)
qs$count <- qs$count[apply(qs$count, 1, max)>5, ]

# keep overlapping sample and ref genes, and merge in one matrix
comb.count <- do.call(cbind, format_to_ref(qs$count, r.count)[1:2])
#>                 nb.genes
#> refdata            19953
#> samp               17596
#> intersect.genes    17274

# combine pdata
comb.p <- data.frame(
  time = c(ae_qs$age.estimates[,1], r.time),
  strain = c(qs$pdat$strain, factor(rep('ctr', ncol(r.count)),
                                    levels = c('ctr', 'mut'))), # ref is ctr
  batch = factor(rep(c('samp', 'ref'), c(ncol(qs$count), ncol(r.count))),
                 levels = c("samp", "ref"))
)
```

Infer gene dispersions by fitting a model on sample data only.

```
# build DESeq object
dd0 <- DESeqDataSetFromMatrix(countData = comb.count[ ,1:6],
                              colData = comb.p[1:6, ],
                              design = ~strain) # no age yet
# compute gene dispersions
dd0 <- estimateSizeFactors(dd0)
dd0 <- estimateDispersions(dd0, fitType = "local")
d0 <- dispersions(dd0) # store dispersions
d0[is.na(d0)] <- 0 # remove NAs
```

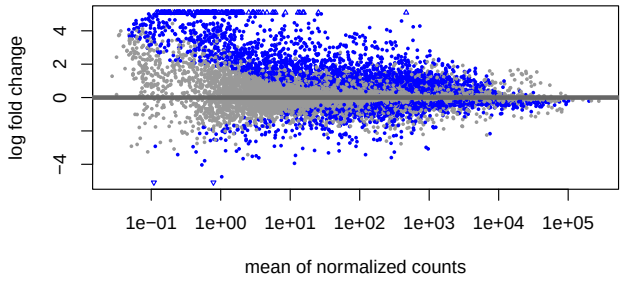
Build a model with combined reference and sample data, injecting previously estimated gene dispersions.

```
dd1 <-  DESeqDataSetFromMatrix(
  countData = comb.count,
  colData = comb.p,
  design = ~ splines::ns(time, df=3)+batch+strain # use df found above
)
dd1 <- estimateSizeFactors(dd1)
# inject dispersions from sample-only model
dispersions(dd1) <- d0

dd1 <- nbinomWaldTest(dd1)
```

Extract DE results.

```
res <- results(dd1)
summary(res)
#>
#> out of 17274 with nonzero total read count
#> adjusted p-value < 0.1
#> LFC > 0 (up)       : 2961, 17%
#> LFC < 0 (down)     : 1894, 11%
#> outliers [1]       : 0, 0%
#> low counts [2]     : 0, 0%
#> (mean count < 0)
#> [1] see 'cooksCutoff' argument of ?results
#> [2] see 'independentFiltering' argument of ?results
```

```
DESeq2::plotMA(dd1)
```



Results are a standard DESeq2 output and can be manipulated as such.

For an explanation of the analysis, see *How it works*.

Custom functions to transform reference data into artificial counts and run DESeq2 as described above are given in *DE functions*, below (please note they may require tweaking). The following is equivalent to the code above.

```
dd1 <- run_DESeq2_ref(
  X = qs$count,          # sample counts
  p = qs$pdat,           # sample pdata
  formula = "~ strain", # formula (without age)
  ref = r_cel,           # ref. object
  ae_values = ae_qs$age.estimates[,1], # sample age estimates
  window.extend = 1,     # ref. window extension
```

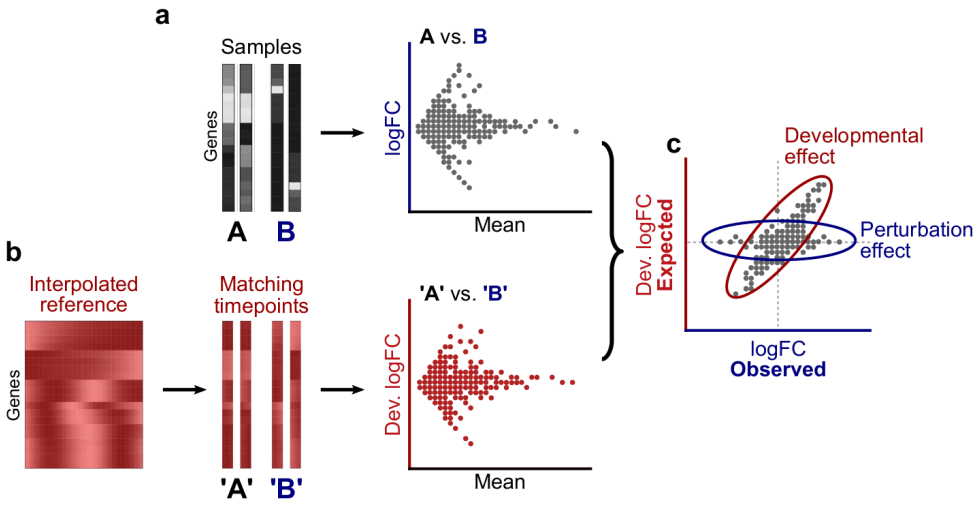
```
  ns.df = 3             # spline df for ref. window
)
```

3 How it works

3.1 Quantifying age-driven expression changes

Between two conditions 'A' and 'B' where the sample groups have developmental differences, the expression changes (or log-fold changes, logFCs) observed between the groups will be a combination of perturbation and developmental effects (**a**).

We can determine the expression changes expected only from the difference in development using reference expression profiles matching the samples (**b**). Any correlation between observed (sample) and expected (reference) logFCs will then correspond to the developmental effect, with uncorrelated logFCs corresponding to the perturbation effect (**c**).



The `ref_compare()` function inputs:

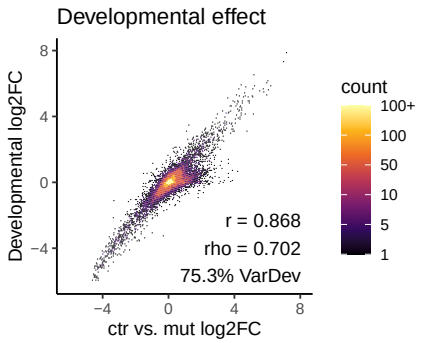
- sample data (expression matrix, expects $log(TPM + 1)$ as transcripts per million are more comparable across datasets)
- the reference object used for age estimation,
- the `ae` object (or age estimate values for the samples),
- and a group variable (*e.g.* a factor defining WT and mutant).

```
qs_rc <- ref_compare(
    X      = qs$tpm,        # sample data, log(TPM+1)
    ref    = r_cel,         # ref object
    ae_obj = ae_qs,         # ae object
    group  = qs$pdat$strain # factor defining compared groups (wt/mut)
)
```

The resulting sample and reference (log2) logFCs can be extracted with `get_logFC()` and plotted. When there are more than 2 compared conditions, you can specify the factor levels to compare to the control (with `l`) when extracting the logFCs.

A custom `ggplot2` function for plotting logFCs can be found in the *Plotting functions* section below.

```
gg_logFC(qs_lfc, main = "Developmental effect",
         xlab = "ctr vs. mut log2FC", ylab = "Developmental log2FC")
```



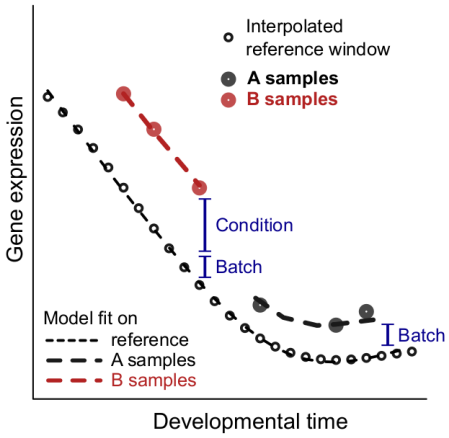
The r^2 between sample and reference logFCs is a rough estimate of the percentage of variance explained by development (`VarDev`, in the bottom right). In this case, over 75% of expression changes can be explained by development, which is expected given the large age difference between the sample groups.

The average age difference between groups and correlation between sample and reference are also given directly in the output of `ref_compare()`.

```
print(qs_rc)
#> DE comparison with reference
#> ---
#>                  ref.logFC.r ref.logFC.r2 ae.avg.dif
#> ctr (ctrl, n=3)           NA           NA         NA
#> mut (n=3)              0.868        0.753      5.035
#> ---
```

3.2 Correcting for age effects by integrating reference data in DE analysis

When there is no developmental overlap between the sample groups to compare, integrating reference data in the DE analysis makes it possible to recover truly DE genes by bridging the gap, as shown in the cartoon below.



Most DE analysis tools input *raw counts* for their particular statistical properties, so the interpolated reference data must be converted from TPM to (artificial) counts assuming an arbitrary fixed library size (25×10^6 counts). Then, because gene dispersions needed for statistical testing cannot be estimated from the noiseless artificial reference, they are inferred from a model without reference data and injected into the model with reference data.

Thus, resulting model coefficients (logFCs) between sample groups are corrected for development by the reference, and their respective statistical tests use dispersion values inferred only from samples. This approach improves upon what is presented in the RAPToR article (Bulteau and Francesconi (2022)), generating valid p-values.

DISCLAIMER: we provide the functions `find_df()`, `run_DESeq2_ref()` and sub-functions needed for age correction DE analysis only in this vignette and not as part of the package because (1) they go beyond the scope of RAPToR, and (2) they might need to be tweaked by the user to adapt to their needs and experimental designs (unlike `ref_compare()`). This approach should also work with tools other than `DEseq2` (e.g. `edgeR`).

4 Demonstration of DE correction in a controlled case

4.1 Data and strategy

Miki, Carl, and Großhans (2017) profiled time-series of *C. elegans* wild-type (WT) and *xrn-2* mutants (GSE97775). Code to download this data and generate the `dsmiki2017` object can be found at the end of this vignette

By selecting specific samples of matching development from both time-series, we define a gold-standard of truly DE genes. Then, by sliding the window of WT samples we can evaluate the effect of increasing developmental shifts between the groups on the DE analysis, as well as measure the advantages of integrating reference data in the model.

```
library(RAPToR)
library(wormRef)

library(DESeq2)
library(splines)
```

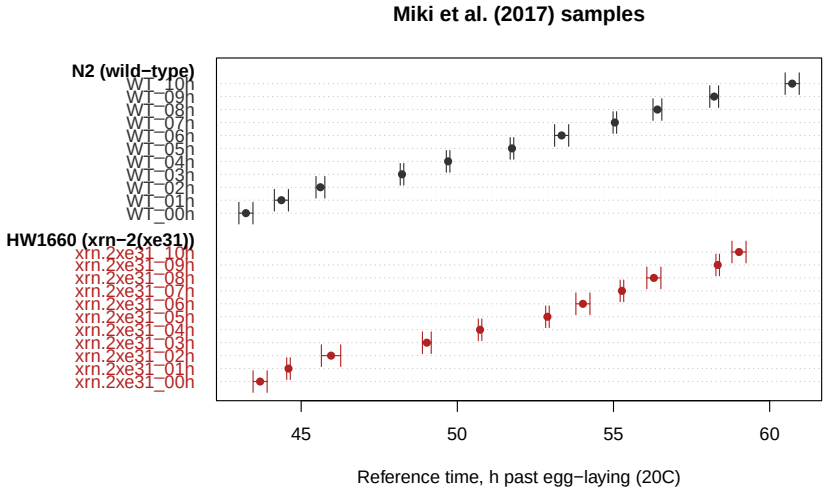
```
library(ROCR)

# for plotting
library(ggplot2)
library(ggpubr)
library(viridis)

col.palette1 <- c('grey20', 'firebrick', 'royalblue', 'forestgreen')
col.palette2 <- viridisLite::viridis(5)
```

4.2 Defining age-matched and shifted sample sets

We start by infering sample age.

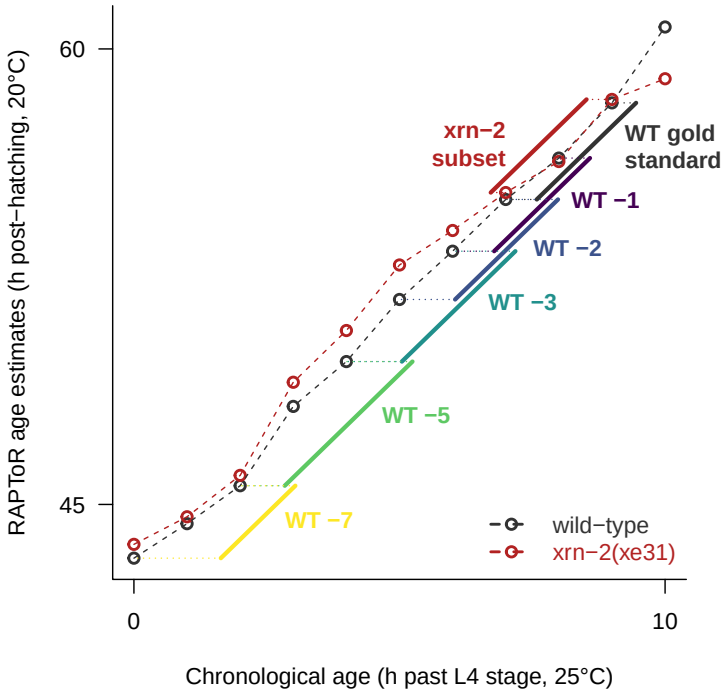
```
# load reference
r_cel <- prepare_refdata("Cel_larv_YA", "wormRef", 600)

# estimate sample age
ae_miki <- ae(dsmiki2017$g, r_cel)
dsmiki2017$p$ae <- ae_miki$age.estimates[, 1]
```

```
plot(ae_miki, main = "Miki et al. (2017) samples",
     group = dsmiki2017$p$strain_long, show.boot_estimates = F,
     color = col.palette1[dsmiki2017$p$strain], lmar = 10, g.line= 1)
```



We then select a set of 3 WT and 3 mutant samples as the age-matched gold-standard, and define sets of 3 WT samples shifted by 1, 2, 3, 5, and 7 time points compared to the mutants.

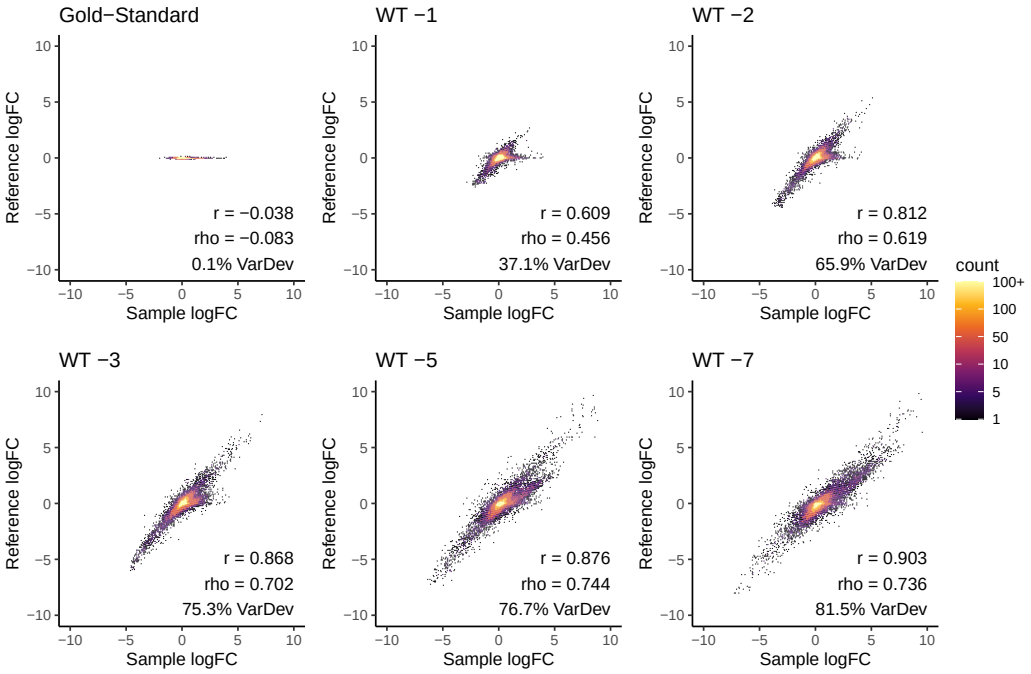
```
GS_wt <- 8:10 # gold standard WT samples
GS_mut <- 19:21 # GS mutant samples

shifts <- -c(1,2,3,5,7) # number of timepoints to shift
subs <- c(list(gold.standard = c(GS_wt, GS_mut)),
          lapply(shifts, function(s) c(GS_wt + s, GS_mut)))
names(subs)[-1] <- paste0('s', abs(shifts))
```



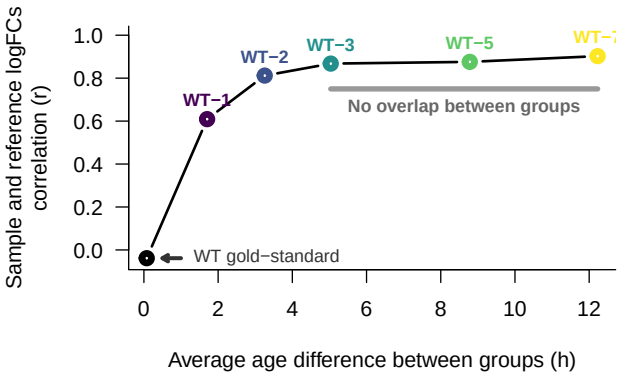
4.3 Quantifying age-driven expression changes in the shifted sets

With `ref_compare()`, we quantify the effect of development in all the sample sets defined above, and plot the reference vs. sample logFCs.

```
rcs <- lapply(subs, function(s){
    RAPToR::ref_compare(
      X = dsmiki2017$g[, s],
      ref = r_cel,
      ae_values = ae_miki$age.estimates[s,1],
      group = dsmiki2017$p$strain[s]
      )
})
```



Unsurprisingly, age-matched reference logFCs (*i.e* development) account for increasing proportions of expression changes with larger age differences between groups.



4.4 Effect of developmental shifts on DE analysis performance

4.4.1 Gold-standard DE

Using the age-matched mutant and WT samples defined as gold-standard, we find a set truly DE genes using a standard `DESeq2` workflow.

We filter genes to keep those overlapping with the reference, and with at least 5 counts in any sample.

```
g.filt <- dsmiki2017$g.raw[apply(dsmiki2017$g.raw, 1, function(r) max(r)>5),]
g.filt <- RAPToR::format_to_ref(g.filt, r_cel$interpGE)$samp
#>                nb.genes
#> refdata           19953
#> samp              18073
```

```
#> intersect.genes    17659
```

We then build a DESeq model including age and strain and test for significant differences between mutant and WT groups. We define a `run_DESeq2_age()` function to do this (see *DE functions* below for code).

```
DE.GS <- run_DESeq2_age(g.filt[, subs$gold.standard],
                        dsmiki2017$p[subs$gold.standard, ])
res.GS <- get_DEres(DE.GS, coefname = "strain_xrn2_vs_wt") # get results table
```

We consider genes as DE with the thresholds below.

```
# Define DE (with p < thr AND |logFC| > thr)
thr.p <- 0.01
thr.logFC <- 1.0

# get true DE genes from gold-standard
truth <- (res.GS$padj < thr.p) & abs(res.GS$log2FoldChange) >= thr.logFC
table(truth)
#> truth
#> FALSE  TRUE
#> 16534  1125
```

With thresholds of $p.value < 0.01$ and $|logFC| > 1$, we find 1125 DE genes in the absence of developmental effects ("true" DE genes).

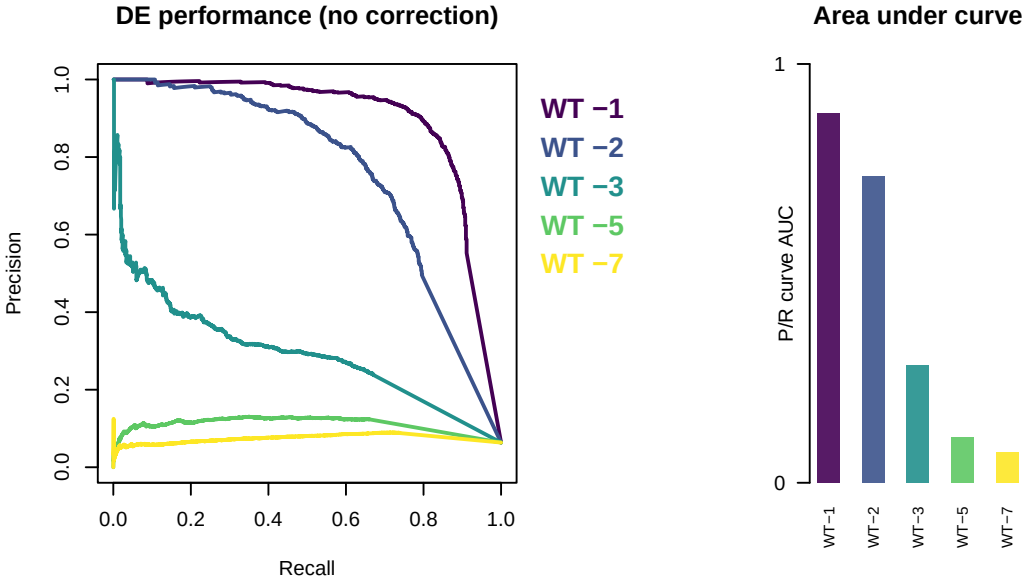
4.4.2 DE with developmental shifts

We now apply the same model as above on the 5 sample subsets with shifted WT.

```
DE.shifts <- lapply(subs[-1], function(s){
  run_DESeq2_age(g.filt[, s], dsmiki2017$p[s, ])
  })
res.shifts <- lapply(DE.shifts, get_DEres)
```

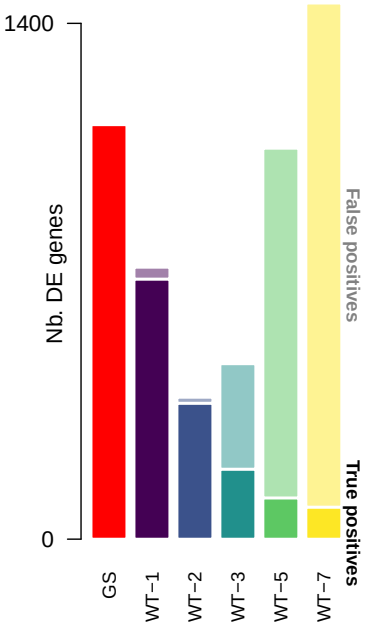
Precision and recall (P/R) metrics allow us to measure the performance the DE analysis (p-value and logFC) in discriminating truly DE genes (*i.e.* genes found in the gold-standard DE above) from non-DE genes. An area under the P/R curve of 1 is a perfect classifier.

```
# compute precision-recall curves
rocs_initial_age <- lapply(seq_along(subs[-1]), function(i){
  v <- res.shifts[[i]]$padj
  v[abs(res.shifts[[i]]$log2FoldChange)<=thr.logFC] <- 1
  ROCR::prediction(predictions = -v, labels = truth)
})
```



We see a sharp decline in the DE model performance in detecting true DE genes as the developmental shift increases, particularly once there is no more developmental overlap between the compared groups (starting at WT-3).

If we select DE genes with the same thresholds as the gold-standard, this drop in performance translates to both a decrease of true positives and an increase of false positives, as shown below.



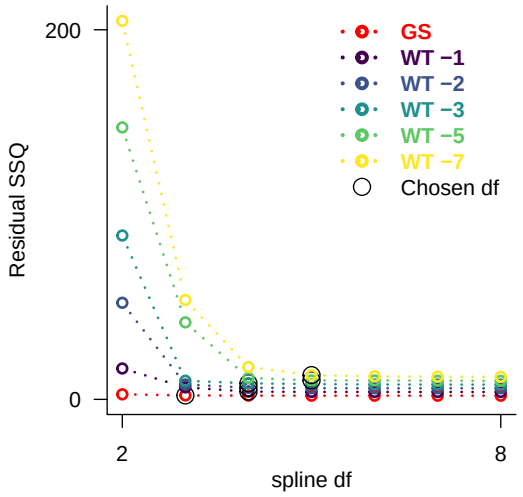
4.5 DE analysis integrating reference data to correct age effect

4.5.1 Selecting reference windows

We select a reference window spanning the sample age range with an added margin of 1 hour on either side for each subset, and find the optimal spline degree-of-freedom (df) to model expression dynamics covered by the reference window.

We define a `find_df()` function which computes the residual sum of squares for a range of spline dfs and choose the optimal df values where a plateau is reached (see *DE functions* below for code).

```
dfs <- 2:8
dfs_ssq <- lapply(subs, function(s){
  find_df(r_cel, # ref object
          w = range(ae_miki$age.estimates[s,1]) + c(-1, 1), # time window
          dfs = dfs) # df range to test
})
# optimal df for each subset based on plot below
df_opti <- c(3,4,4,4,5,5)
```



The selected df increases with the developmental shift, which is expected since the reference window to include gets larger and may thus contain more complex dynamics.

4.5.2 DE models with reference data

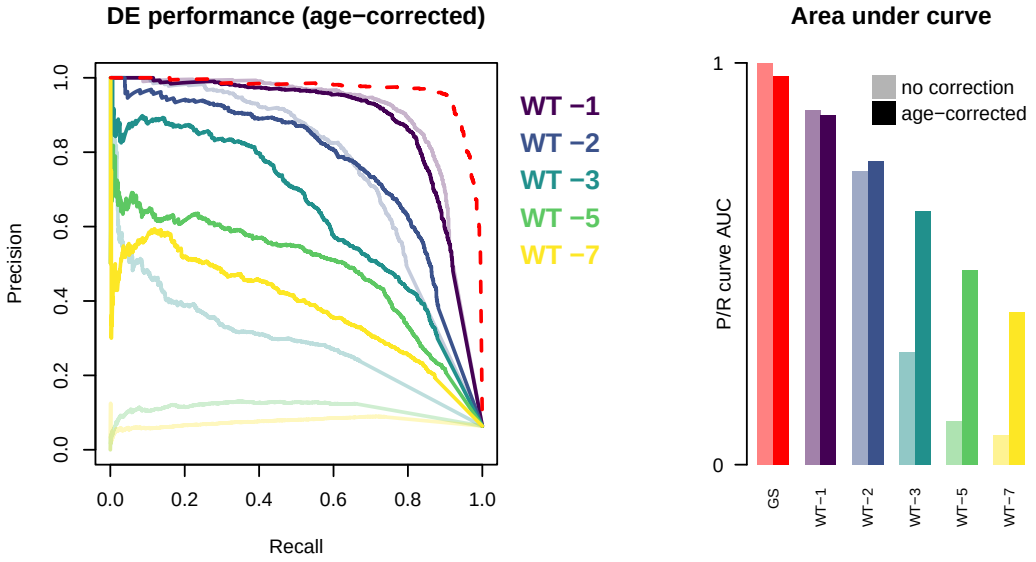
Having defined the appropriate fit parameters for the reference, we can integrate in the DE model. We define `run_DESeq2_ref()` to do this (see *DE functions* below for code).

```
# run DESeq2 with reference data inclusion on shifted subsets
DEref.shifts <- lapply(seq_along(subs), function(i){
  s <-subs[[i]]  # select sample subset
  run_DESeq2_ref(X = g.filt[, s],                # sample counts
                 p = dsmiki2017$p[s, ],          # sample pdata
                 ae_values = dsmiki2017$p$ae[s], # age estimates
                 formula = ~strain,   # formula (age is added by the function)
                 ref = r_cel,         # ref object
                 ns.df = df_opti[i],  # df for spline fit
```

```
                  window.extend = 1)   # reference window extension
})
res_ref.shifts <- lapply(DEref.shifts, get_DEres) # get results table
```

As done above, let's assess the performance of these models in detecting truly DE genes.

```
# compute age-corrected precision-recall curves
rocs_corrected_age <- lapply(seq_along(subs), function(i){
  v <- res_ref.shifts[[i]]$padj
  v[abs(res_ref.shifts[[i]]$log2FoldChange)<=thr.logFC] <- 1
  ROCR::prediction(predictions = -v, labels = truth)
})
```



Compared to the non-corrected models we strongly rescue the performance of the analysis when there is no overlap between the compared sample groups (starting at `WT-3`).

In less shifted subsets or in the gold-standard, we see that the performance of the age-corrected DE analysis to find truly DE genes is comparable to applying no correction. This is expected since developmental dynamics can be properly modeled without reference data as long as there is overlap between the compared groups, nullifying the advantage of adding it to the model.

5 Functions and code

5.1 Code to generate `dsmiki2017`

Required libraries and variables:

```
data_folder <- "../inst/extdata/"

library("GEOquery") # bioconductor package

requireNamespace("wormRef", quietly = T)
requireNamespace("utils", quietly = T)
```

Note : set the `data_folder` variable to an existing path on your system where you want to store the objects.

```
raw2tpm <- function(rawcounts, genelengths){
  if(nrow(rawcounts) != length(genelengths))
    stop("genelengths must match nrow(rawcounts).")
  x <- rawcounts/genelengths
  return(t( t(x) * 1e6 / colSums(x) ))
}

fpkm2tpm <- function(fpkm){
  return(exp(log(fpkm) - log(colSums(fpkm)) + log(1e6)))
}
```

```
geo_dsmiki2017 <- "GSE97775"
url_dsmiki2017 <- as.character(
  GEOquery::getGEOSuppFiles(geo_dsmiki2017,
                            makeDirectory = FALSE,
                            fetch_files = FALSE)[9,"url"]
  )
tmpf <- file.path(data_folder, "miki2017.txt.gz")
utils::download.file(url_dsmiki2017, destfile = tmpf)
g <- read.table(gzfile(tmpf), h=T, as.is = T, row.names = 1)
file.remove(tmpf)


# format genes to WB ids
g <- RAPToR::format_ids(g, wormRef::Cel_genes,
                        from = "wb_id", to = "wb_id",
                        aggr.fun = sum)


# store raw counts and convert to log(TPM+1)
g.raw <- g
g <- raw2tpm(g.raw,
             wormRef::Cel_genes$transcript_length[
               match(rownames(g.raw), wormRef::Cel_genes$wb_id)
               ])
g <- log1p(g)


# sample metadata
p <- Biobase::pData(
  GEOquery::getGEO(geo_dsmiki2017, getGPL = F)[[1]]
  )[14:35, c(1,2, 45)]
colnames(p)[3] <- "strain_long"
p[,3] <- factor(p[, 3], levels = unique(p[,3]),
                labels = c("N2 (wild-type)", "HW1660 (xrn-2(xe31))"))
p$strain <- p[,3]
levels(p$strain) <- c("wt", "xrn2")

# get chronological age from sample name
p$age <- as.numeric(gsub(".*_(\\d+)h", "\\1", p$title))
```

```
p$title <- colnames(g)


dsmiki2017 <- list(g = g, p = p, g.raw = g.raw)

save(dsmiki2017, file = file.path(data_folder, "dsmiki2017.RData"),
     compress = "xz")
rm(url_dsmiki2017, geo_dsmiki2017, tmpf, g, g.raw, p, raw2tpm)
```

5.2 Code to generate quickstart data

Given the `dsmiki2017` data generated above:

```
sel <- c(5:7, 19:21) # select 3 WT and 3 mut

qs <- list(
  tpm = dsmiki2017$g[, sel],
  count = dsmiki2017$g.raw[, sel],
  pdat = data.frame(
    sname = paste0(rep(c('ctr', 'mut'), e=3), rep(1:3,2)),
    strain = factor(rep(c('ctr', 'mut'), e=3)),
    stringsAsFactors = F)
)
colnames(qs$tpm) <- qs$pdat$sname
colnames(qs$count) <- qs$pdat$sname

save(qs, file = file.path(data_folder, 'sc4_qsdata.RData'), compress = "xz")
```

5.3 Plotting functions

Plotting a logFC comparison as binned heatmap (`gg_logFC()`)

```
gg_logFC <- function(x, y=NULL, rg = range(xy, na.rm = T)*1.02,
                     l.breaks = log1p(c(0, 1, 5, 10, 50, 100, Inf)),
                     l.labels = c('1', '5', '10', '50', '100', '100+'),
                     nbins = 200,
                     xlab = "log2(FC) in x",
                     ylab = "log2(FC) in y",
                     main = "", get.r =T, add.vd = T,
                     DEgsel = NULL,
                     ...){
  # Make a 2d binned hexplot for showing logFC comparison
  require(ggplot2)
  require(viridisLite)

  if(is.null(y))
    xy <- x
  else
    xy <- cbind(x, y)
  xy <- as.data.frame(xy)
```

```
  colnames(xy) <- c("x", "y")

  g <- ggplot(data = xy, mapping = aes(x=x, y=y)) +
    stat_bin_hex(aes(fill = after_stat(cut(log1p(after_stat(count)), breaks = l.breaks,
                                labels = F, right = T, include.lowest = T))),
             bins=nbins) +
    scale_fill_gradientn(colors = viridisLite::inferno(length(l.breaks)),
                         name = 'count', labels = l.labels) +
    theme_classic() + xlim(rg) + ylim(rg) +
    xlab(xlab) + ylab(ylab) + ggtitle(main) + coord_fixed()
    if(get.r){
    if(any(is.na(xy))){
      message("Warning: removed NA values to compute correlation coefs")
      rmsel <- apply(xy, 1, function(r) any(is.na(r)))
      xy <- xy[!rmsel,]
      if(!is.null(DEgsel))
        DEgsel <- DEgsel[!rmsel]
    }

    cc <- cor(xy)[1,2]
    cc2 <- cor(xy, method='spearman')[1,2]
    cctxt <- paste0("r = ", round(cc, 3), '\nrho = ', round(cc2, 3))
    if(add.vd)
      cctxt <- paste0(cctxt, "\n", round(100*(cc^2), 1), "% VarDev")
    if(!is.null(DEgsel)){
      ccDE <- cor(xy[DEgsel,])[1,2]
      cctxt <- paste0(cctxt, "\nr(DE) = ", round(ccDE, 3))
      if(add.vd)
        cctxt <- paste0(cctxt, "\n", round(100*(ccDE^2), 1), "% VarDev(DE)")
    }

    g <- g + annotate(geom="text", hjust=1, vjust=0,
                      x = rg[2],
                      y = rg[1],
                      label = cctxt)
  }

  return(g)
}
```

Make a color transparent (`transp()`) and add an axis with 2 tick marks `twoTicks()`

```
transp <- function(col, a=.5){
  # Make a color transparent
  colr <- col2rgb(col)
  return(rgb(colr[1,], colr[2,], colr[3,], a*255, maxColorValue = 255))
}

twoTicks <- function(side = c(1,2), col = NA,
                     col.ticks = 1, las = c(1,2), ...){
  # Generate 2 (edge) ticks on given axis of current plot
  # col = NA & col.ticks = 1 makes the axis line dissapear, but keeps ticks
```

```
  for(i in seq_along(side)){
    axt <- axTicks(side[i])
    axis(side[i], at = axt[c(1, length(axt))], col = col,
         col.ticks = col.ticks, las = las[i], ...)
  }
}
```

5.4 DE functions

Running a standard DESeq2 workflow (`run_DESeq2_age()`) and extracting a results table (`get_DEres()`).

```
run_DESeq2_age <- function(X, p){
  # Run DEseq2 wt vs. mutant (with age covariate)
  require(DESeq2)
  rownames(p) <- colnames(X)
  p$aes <- scale(p$ae) # scale age value
  dds <- DESeqDataSetFromMatrix(countData = X,
                                colData = p,
                                design = ~aes+strain)
  dds <- DESeq(dds, test = "Wald", fitType = "local")
  return(dds)
}

get_DEres <- function(dds, coefname="strain_xrn2_vs_wt"){
  # Get results table from deseq output, managing NAs

  res <- results(dds, name=coefname)
  # manage NAs
  res$padj[is.na(res$padj)] <- 1
  res$log2FoldChange[is.na(res$log2FoldChange)] <- 0

  return(res)
}
```

Finding optimal df to fit a reference window (`find_df()`) and converting reference data to counts (`log1ptpm_2rawcounts()`, `ref_2counts()`).

```
find_df <- function(ref, w, dfs=2:8){
  # Compute ssq of spline fits to find optimal df

  w.idx <- seq(max(c(1L, which.min(abs(w[1]-ref$time))-1 )),
               min(c(length(ref$time), which.min(abs(w[2]-ref$time))+1)))
  # get time values of window
  ts <- ref$time[w.idx]
  # get PCs of reference window
  pc <- summary(stats::prcomp(t(ref$interpGE[,w.idx]), scale=F, center=T))
  # keep enough components for 99% var
  spc <- sum(pc$importance[3, ] < 0.99)+1
  pcfit <- pc$x[, 1L:spc]
  w <- pc$importance[2, 1L:spc]
  # compute ssq of fit residuals with different dfs
```

```
  # for each df, compute weighted ssq of spline fit on
  ssqs <- cbind(sapply(dfs, function(dfi){
    ssq <- sum(w * colSums(
      stats::residuals(stats::lm(pcfit~splines::ns(ts, df=dfi)))^2
    ))
  }))/(length(ts))
  return(ssqs)
}


log1ptpm_2rawcounts <- function(X, glengths, nreadbygl){
  # Transform log1p(tpm) to (artificial) raw counts
  # note : nreadbygl = colSums(rawcounts/genelengths)
  if(length(nreadbygl) != ncol(X))
    stop("nreadbygl != ncol(X)")
  if(length(glengths) != nrow(X))
    stop("glengths must be of length nrow(X)")
  X <- t( (t(exp(X) - 1)/1e6) * nreadbygl ) * glengths
  X[X<0] <- 0
  return(round(X))

}


ref_2counts <- function(ref, ae_values,
                        gltable = wormRef::Cel_genes[,c("wb_id",
                                                        "transcript_length")],
                        avg_librarysize = 25e6){
  # Get expression profiles of given age from a RAPToR reference as
  # (artificial) count data.
  # note : gltable must have WBids as col 1 and gene length as col 2.

  # ref expression profiles at given timepoints :
  rX <- RAPToR::get_refTP(ref, ae_values=ae_values, return.idx = F)

  # transform to counts
  gl <- gltable[match(rownames(rX), gltable[,1]), 2]
  rX <- log1ptpm_2rawcounts(rX, gl, nreadbygl = rep(avg_librarysize,
                                                    ncol(rX))/median(gl))

  return(rX)
}
```

Integrating reference data in a DESeq model (`run_DESeq2_ref()`).

```
run_DESeq2_ref <- function(X, p, formula, ref,
                           ae_values=NULL, window.extend=1, ns.df=3){
  # Run DEseq2 wt vs. mutant correcting for development with ref. data.

  # Do no not specify age in the formula, it is added directly by the function.
  # Age estimates should either be an 'ae' column of p or given as 'ae_values'.
```

```
require(DESeq2)

if(!any(colnames(p)=='ae') & is.null(ae_values)){
  stop("Age estimates should either be a column of p, or given to ae_values.")
}
if(!is.null(ae_values)){
  p$ae <- ae_values
}

## Extract reference expression profiles in sample time window
w.rg <- range(p$ae) + c(-window.extend, window.extend)
w.idx <- seq(
  max(c(1, which.min(abs(w.rg[1]-ref$time))-1 )),
  min(c(length(ref$time), which.min(abs(w.rg[2]-ref$time))+1))
  )
# ref window time values
w.ts <- ref$time[w.idx]
# ref window expression values
w.GE <- ref_2counts(ref = ref, ae_values = w.ts)

## Join ref & sample data
nX <- ncol(X)
nR <- ncol(w.GE)

# get overlapping genes & join expression data
ovl <- RAPToR::format_to_ref(samp = X, refdata = w.GE, verbose = F)
Xj <- as.matrix(cbind(ovl$refdata, ovl$samp))

# get relevant fields from p
f0 <- as.formula(formula)
p2 <- p[, attr(terms(f0), "term.labels"), drop=F]
# get 1st level of each variable
lev0 <- lapply(p2, function(col) levels(col)[1])

# join p data and add Time and ref/sample Batch
pj <- cbind(
  Time = c(w.ts, p$ae),
  Batch = factor(rep(c('r', 's'), c(nR, nX))),
  rbind(do.call(cbind, lapply(lev0, rep, times=nR)), p2) # other terms
  )
rownames(pj) <- colnames(Xj)


# Estimate dispersions with samples only
s0 <- pj$Batch=="s"
dds0 <- DESeqDataSetFromMatrix(countData = Xj[,s0],
                               colData = pj[s0,],
                               design = f0)
dds0 <- estimateSizeFactors(dds0)
dds0 <- estimateDispersions(dds0, fitType = "local")
dd <- dispersions(dds0) # store dispersions
```

```
  dd[is.na(dd)] <- 0 # remove NAs

  ## Build full DE model with reference
  # Add Time and ref/sample batch to model formula
  f1 <- update.formula(f0, substitute(
    ~ splines::ns(Time, df = ns.df) + Batch + .,
    list(ns.df=ns.df)
    ))
  dds <-  DESeqDataSetFromMatrix(countData = Xj,
                                 colData = pj,
                                 design = f1)
  dds <- estimateSizeFactors(dds)
  # inject dispersions from sample-only model
  dispersions(dds) <- dd

  dds <- nbinomWaldTest(dds)

  return(dds)
}
```

References

Bulteau, Romain, and Mirko Francesconi. 2022. "Real Age Prediction from the Transcriptome with RAPToR." *Nature Methods*, 1–7.

Miki, Takashi S, Sarah H Carl, and Helge Großhans. 2017. "Two Distinct Transcription Termination Modes Dictated by Promoters." *Genes & Development* 31 (18): 1870–79.

SessionInfo

```
sessionInfo()
```

```
#> R version 4.1.2 (2021-11-01)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 22.04.1 LTS
#>
#> Matrix products: default
#> BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
#> LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.20.so
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>  [3] LC_TIME=C                  LC_COLLATE=en_US.UTF-8
#>  [5] LC_MONETARY=fr_FR.UTF-8    LC_MESSAGES=en_US.UTF-8
#>  [7] LC_PAPER=fr_FR.UTF-8       LC_NAME=C
#>  [9] LC_ADDRESS=C               LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=fr_FR.UTF-8 LC_IDENTIFICATION=C
#>
#> attached base packages:
#> [1] splines   stats4    stats     graphics  grDevices utils     datasets
```

```
#> [8] methods   base
#>
#> other attached packages:
#>  [1] ica_1.0-3                   drosoRef_0.2.0
#>  [3] limma_3.50.3                vioplot_0.4.0
#>  [5] zoo_1.8-11                  sm_2.2-5.7.1
#>  [7] beeswarm_0.4.0              RColorBrewer_1.1-3
#>  [9] viridis_0.6.2               viridisLite_0.4.1
#> [11] ggpubr_0.6.0                ggplot2_3.4.0.9000
#> [13] ROCR_1.0-11                 DESeq2_1.34.0
#> [15] SummarizedExperiment_1.24.0 Biobase_2.54.0
#> [17] MatrixGenerics_1.6.0        matrixStats_0.63.0
#> [19] GenomicRanges_1.46.1        GenomeInfoDb_1.30.1
#> [21] IRanges_2.28.0              S4Vectors_0.32.4
#> [23] BiocGenerics_0.40.0         wormRef_0.5.0
#> [25] RAPToR_1.2.0                BiocStyle_2.22.0
#>
#> loaded via a namespace (and not attached):
#>  [1] colorspace_2.1-0       ggsignif_0.6.4         pryr_0.1.6
#>  [4] XVector_0.34.0         rstudioapi_0.14        hexbin_1.28.2
#>  [7] farver_2.1.1           bit64_4.0.5            AnnotationDbi_1.56.2
#> [10] fansi_1.0.4            codetools_0.2-18       cachem_1.0.6
#> [13] geneplotter_1.72.0     knitr_1.42             jsonlite_1.8.4
#> [16] broom_1.0.3            annotate_1.72.0        png_0.1-8
#> [19] BiocManager_1.30.19    compiler_4.1.2         httr_1.4.4
#> [22] backports_1.4.1        Matrix_1.5-3           fastmap_1.1.0
#> [25] cli_3.6.0              htmltools_0.5.4        tools_4.1.2
#> [28] gtable_0.3.1           glue_1.6.2             GenomeInfoDbData_1.2.7
#> [31] dplyr_1.1.0            Rcpp_1.0.10            carData_3.0-5
#> [34] jquerylib_0.1.4        vctrs_0.5.2            Biostrings_2.62.0
#> [37] nlme_3.1-161           xfun_0.37              stringr_1.5.0
#> [40] rbibutils_2.2.13       lifecycle_1.0.3        rstatix_0.7.2
#> [43] XML_3.99-0.13          zlibbioc_1.40.0        scales_1.2.1
#> [46] parallel_4.1.2         yaml_2.3.7             memoise_2.0.1
#> [49] gridExtra_2.3          sass_0.4.5             stringi_1.7.12
#> [52] RSQLite_2.2.20         highr_0.10             genefilter_1.76.0
#> [55] BiocParallel_1.28.3    Rdpack_2.4             rlang_1.0.6
#> [58] pkgconfig_2.0.3        bitops_1.0-7           evaluate_0.20
#> [61] lattice_0.20-45        purrr_1.0.1            labeling_0.4.2
#> [64] cowplot_1.1.1          bit_4.0.5              tidyselect_1.2.0
#> [67] magrittr_2.0.3         bookdown_0.32          R6_2.5.1
#> [70] magick_2.7.3           generics_0.1.3         DelayedArray_0.20.0
#> [73] DBI_1.1.3              pillar_1.8.1           withr_2.5.0
#> [76] mgcv_1.8-41            survival_3.5-0         KEGGREST_1.34.0
#> [79] abind_1.4-5            RCurl_1.98-1.10        tibble_3.1.8
#> [82] crayon_1.5.2           car_3.1-1              utf8_1.2.3
#> [85] rmarkdown_2.20.1       locfit_1.5-9.7         grid_4.1.2
#> [88] data.table_1.14.6      blob_1.2.3             digest_0.6.31
#> [91] xtable_1.8-4           tidyr_1.3.0            munsell_0.5.0
#> [94] bslib_0.4.2
```