RAPToR - Building References

RAPToR 1.2.0

Romain Bulteau

March 2023

Contents

1 Introduction

This vignette is specifically focused on building RAPToR references, and assumes knowledge of general use of the package (see the main "RAPToR" vignette).

Building references is a key aspect of RAPToR, as samples need an appropriate reference to be staged. References are broadly usable once built, so they are worth the trouble to set up. This vignette, will go through the general workflow of building a reference from selecting an appropriate dataset, to validating a model for interpolation.

Along the explanations, examples will be given using *C. elegans* larval development time-series data published by Aeschimann et al. (2017) and Hendriks et al. (2014), stored in dsaeschimann2017 and dshendriks2014 respectively. Code to create these objects can be found at the end of this vignette.

Further examples are given at the end for broader coverage of reference-building needs.

2 Selecting / Preparing a dataset

We start from a gene expression profiling time-series spanning the developmental stages of interest for your organism. Thankfully, these time-series experiments are increasingly numerous in the literature and many models will already have some kind of useable data. You may also make (or already have) your own time-series on hand.

2.1 Databases

There are a few expression profiling databases to search in and download data from. The most well-known are the Gene Expression Omnibus (GEO) and the Array Express.

Both of these databases have APIs to get their data directly from R (*e.g* the GEOquery package, used in multiple data loading scripts of the RAPToR vignettes).

2.2 What to look out for

Several points of the experimental design should be kept in mind when selecting data for a reference.

- ***Are there replicates ?*** If so, good. This means you can confirm the dynamics in the data are not noise. A slightly sparser time-series with replicates could be a better candidate to build a reference than an experiment with one batch.
- ***Is the time point sampling even ?*** Profiling is still expensive, so time-series sometimes account for dynamic ranges of development (*i.e.* early or fast-changing stages are more sampled). Even sampling is better for spline fitting, but data can still otherwise be used by transforming time values so they are more uniform (see *Example 3*.
- ***What's the developmental range ?*** Usually, the bigger, the better! Aging (as opposed to development) reference need special handling (see *Aging references*, and *Example 4*)
- ***Is the profiling done on whole-organism, dissected parts, single-cells ?*** You should aim for profiling that matches your sample type. Whole-organism reference for whole-organism samples, dissected tissue/organ reference for similar samples. Dissected tissue (or single-cell) samples are often sparser than whole-organism for biological and technical reasons. To account for this, we recommend to filter out genes with median $log(TPM + 1) < 0.5$ across reference samples.

- ***What's the profiling technology ?*** RNA-seq is much, *much* cleaner than microarray data, so it's preferable. There is however no trouble staging RNA-seq samples on microarray references and vice-versa. RNA-seq counts should have some within-sample normalization (*e.g.* TPM) to reflect gene expression accurately across samples.

2.3 ID formatting

Bioinformatics is a field plagued by diverse and fast-changing sets of IDs for genes, transcripts, etc. When you build a reference, you should always convert it to IDs that are **conventional and stable**. We like to use the organism-specific IDs (*e.g*, Wormbase for *C. elegans* : `WBGene00016153`, Flybase for *Drosophila* : `FBgn0010583`).

The ensembl biomart or its associated R package `biomaRt` are a very useful resource to get gene, transcript or probe ID lists for conversion.

Below is a code snippet to get gene IDs for drosophila with `biomaRt`.

```
library(biomaRt)

# setup connection to ensembl
mart <- biomaRt::useMart("ensembl", dataset = "dmelanogaster_gene_ensembl")

# get list of attributes
droso_genes <- biomaRt::getBM(attributes = c("ensembl_gene_id",
                                             "ensembl_transcript_id",
                                             "external_gene_name",
                                             "flybase_gene_id"),
                              mart = mart)

head(droso_genes)
#>   ensembl_gene_id ensembl_transcript_id external_gene_name flybase_gene_id
#> 1     FBgn0015380           FBtr0330209                drl     FBgn0015380
#> 2     FBgn0015380           FBtr0081195                drl     FBgn0015380
#> 3     FBgn0010356           FBtr0088240               Taf5     FBgn0010356
#> 4     FBgn0266023           FBtr0343232     lncRNA:CR44788     FBgn0266023
#> 5     FBgn0250732           FBtr0091512               gfzf     FBgn0250732
#> 6     FBgn0250732           FBtr0334671               gfzf     FBgn0250732
```

When multiple probe or transcript IDs match a single gene ID, we usually sum-aggregate counts and mean-aggregate other expression values (microarray or already-processed RNAseq as TPMs). This is taken care of with the `format_ids()` function.

2.4 Normalize and log expression

It's common practice to normalize expression data (*e.g.* to account for technical bias). You may deal with many different profiling technologies when building references, and may even join different datasets together for a single reference.

To stay as consistent as possible, we apply quantile-normalization on all data regardless of source or type. For this, we use the `normalizeBetweenArrays()` function from `limma`.

We also log-transform the data with $log(X + 1)$.

```
library(RAPToR)
library(limma)

dsaeschimann2017$g <- limma::normalizeBetweenArrays(dsaeschimann2017$g,
                                                    method = "quantile")
dsaeschimann2017$g <- log1p(dsaeschimann2017$g)

dshendriks2014$g <- limma::normalizeBetweenArrays(dshendriks2014$g,
                                                  method = "quantile")
dshendriks2014$g <- log1p(dshendriks2014$g)
```

2.5 Exploring the data

It's good practice to explore the data a little before moving on.

We start with quick look at the data: expression matrix and associated sample (or phenotypic) data.

```
dsaeschimann2017$g[1:5,1:3]
#>                 let.7.n2853._18hr let.7.n2853._20hr let.7.n2853._22hr
#> WBGene00007063         2.1206604          2.469532          2.373273
#> WBGene00007064         2.1621558          2.260804          2.661102
#> WBGene00007065         2.7763061          2.847833          2.727037
#> WBGene00003525         0.9434159          2.466223          2.609585
#> WBGene00007067         1.0787531          1.081964          1.350796

head(dsaeschimann2017$p, n = 5)
#>                        title geo_accession           organism_ch1
#> GSM2113587 let.7.n2853._18hr    GSM2113587 Caenorhabditis elegans
#> GSM2113588 let.7.n2853._20hr    GSM2113588 Caenorhabditis elegans
#> GSM2113589 let.7.n2853._22hr    GSM2113589 Caenorhabditis elegans
#> GSM2113590 let.7.n2853._24hr    GSM2113590 Caenorhabditis elegans
#> GSM2113591 let.7.n2853._26hr    GSM2113591 Caenorhabditis elegans
#>                  strain time in development:ch1 age
#> GSM2113587 let-7(n2853)                18 hours  18
#> GSM2113588 let-7(n2853)                20 hours  20
#> GSM2113589 let-7(n2853)                22 hours  22
#> GSM2113590 let-7(n2853)                24 hours  24
#> GSM2113591 let-7(n2853)                26 hours  26
```

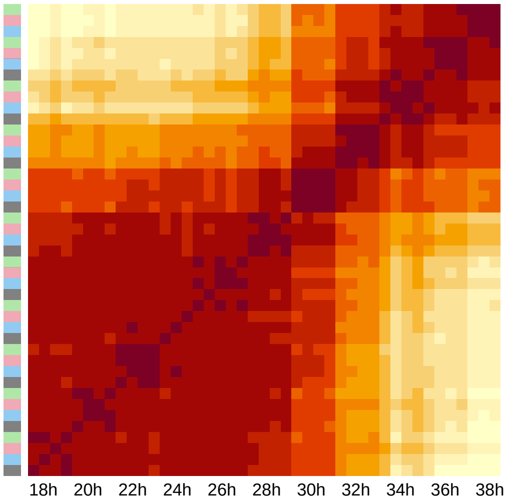
2.5.1 Correlation

Outliers in time series data can be revealed using sample-sample correlation heatmaps or boxplots. This also shows the clear correlation between samples of similar development.

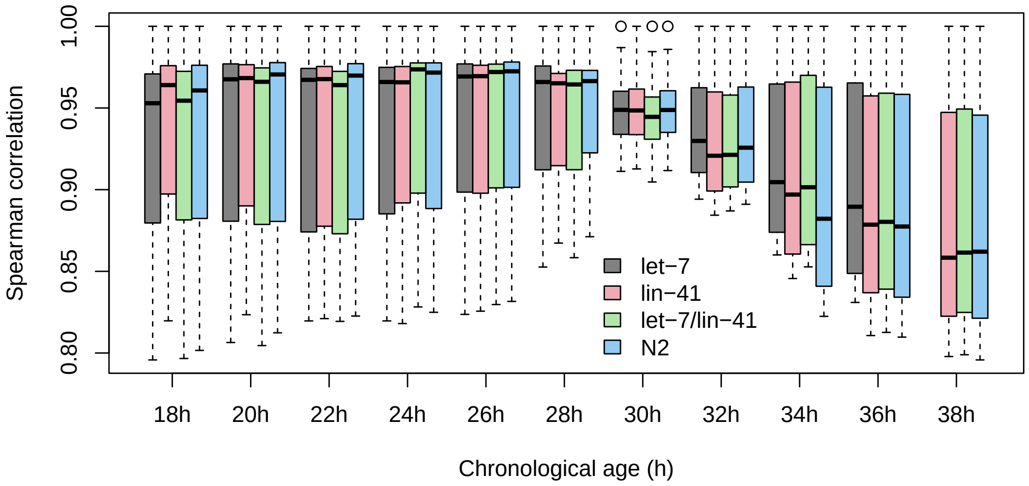
```
cor_dsaeschimann2017 <- cor(dsaeschimann2017$g, method = "spearman")
```

```
ord <- order(dsaeschimann2017$p$age) # order by development
heatmap(cor_dsaeschimann2017[ord, ord], Colv = NA, Rowv = NA,
        scale = "none", keep.dendro = F, margins = c(1,1),
        # see Example 3 for transp() function.
        RowSideColors = transp(as.numeric(dsaeschimann2017$p$strain[ord])),
        labRow = "", labCol = "")
```

```
# add time labels
par(xpd = T)
mtext(text = paste0(unique(dsaeschimann2017$p$age), 'h'), side = 1, line = 4,
      at = seq(-.1,1.05, l = 11))
```



```
boxplot(cor_dsaeschimann2017~interaction(dsaeschimann2017$p$strain,
                                          dsaeschimann2017$p$age),
        col = transp(1:4), # see Example 3 for transp() function.
        xaxt = "n", at = seq(1,44, l = 55)[c(T,T,T,T,F)],
        ylab = "Spearman correlation", xlab = "Chronological age (h)")
#add time labels
axis(side = 1, at = seq(2,42, l = 11),
     labels = paste0(unique(dsaeschimann2017$p$age), 'h'))
legend(23,.87, fill = transp(1:4),
       legend = c("let-7", "lin-41", "let-7/lin-41", "N2"),
       bty = "n")
```



Everything looks good in this case.

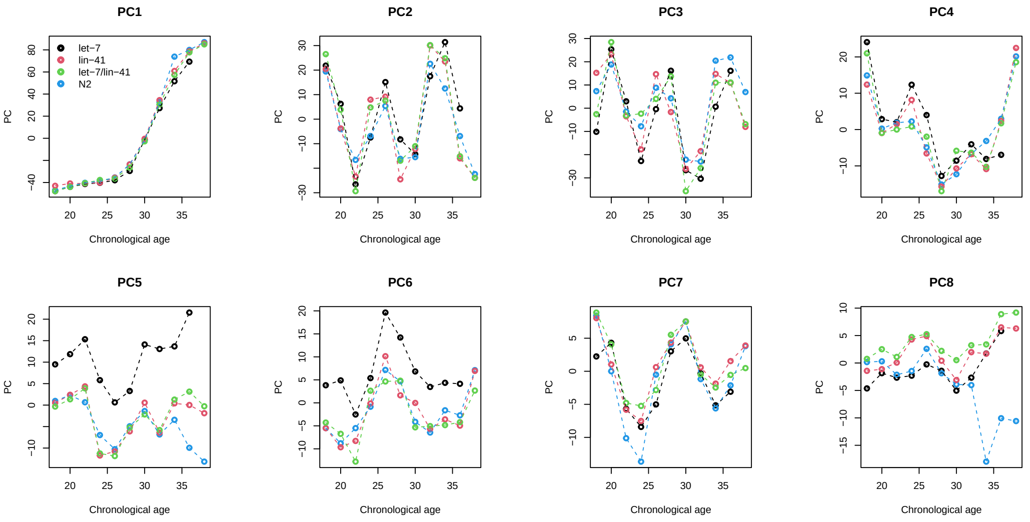
2.5.2 Plotting components

Dimension reductions such as PCA or ICA (Principal or Independent Component Analysis) yield components that summarize the gene expression landscape and dynamics (Alter, Brown, and Botstein (2000)). Plotting these components with respect to time is a good way to grasp general dynamics in the data.

For PCA, we want to perform a non-scaled, centered PCA. Centering is done gene-wise, not sample-wise (hence the matrix rotation below).

```
pca_dsaeschimann2017 <- stats::prcomp(t(dsaeschimann2017$g), rank = 25,
                                      center = TRUE, scale = FALSE)
```

```
par(mfrow = c(2,4), pty='s')

# for components 1:8
invisible(sapply(1:8, function(i){
  # plot points colored by strain
  plot(dsaeschimann2017$p$age, pca_dsaeschimann2017$x[,i],
       lwd = 2, col = dsaeschimann2017$p$strain,
       xlab = "Chronological age", ylab = "PC", main = paste0("PC", i))
  # connect the dots per strain
  sapply(levels(dsaeschimann2017$p$strain), function(l){
    s <- which(dsaeschimann2017$p$strain == l)
    points(dsaeschimann2017$p$age[s], pca_dsaeschimann2017$x[s,i],
           col = which(levels(dsaeschimann2017$p$strain) == l),
           type = 'l', lty = 2)
  })
  if(i == 1) # add legend on first panel
    legend("topleft", bty = 'n',
           legend = c("let-7", "lin-41", "let-7/lin-41", "N2"),
           pch = c(rep(1, 4)), lty = c(rep(NA, 4)),
           col = c(1:4), lwd = 3)
}))
```

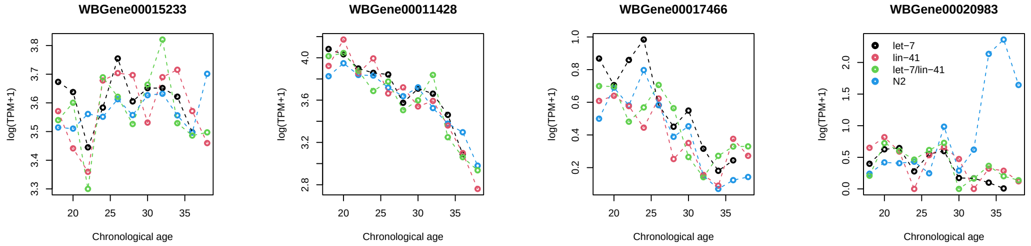


In this *C. elegans* larval development data, we see very consistent dynamics across different strains. Also, PC2 and PC3 capture an oscillatory dynamic which is characteristic of molting in *C. elegans*.

2.5.3 Plotting random genes

We can also plot a few random genes, to get a grasp on the noise in the data.

```
# select random genes
set.seed(10)
gtp <- sample(nrow(dsaeschimann2017$g), size = 4)

par(mfrow = c(1,4), pty='s')
invisible(sapply(gtp, function(i){
  plot(dsaeschimann2017$p$age, dsaeschimann2017$g[i,],
       lwd = 2, col = dsaeschimann2017$p$strain,
       xlab = "Chronological age", ylab = "log(TPM+1)",
       main = rownames(dsaeschimann2017$g)[i])
  sapply(levels(dsaeschimann2017$p$strain), function(l){
    s <- which(dsaeschimann2017$p$strain == l)
    points(dsaeschimann2017$p$age[s], dsaeschimann2017$g[i,s],
           col = which(levels(dsaeschimann2017$p$strain) == l),
           type = 'l', lty = 2)
  })
  if(i == gtp[4])
    legend("topleft", bty = 'n',
           legend = c("let-7", "lin-41", "let-7/lin-41", "N2"),
           pch = c(rep(1, 4)), lty = c(rep(NA, 4)),
           col = c(1:4), lwd = 3)
}))
```



3 The gene expression interpolation model (GEIM)

Increasing the resolution of a profiling time series is a very unbalanced regression problem. We want to predict tens of thousands of dependent variables (genes) with a few independent variables (time, batch, . . .).

3.1 Fitting models in component space

In order to fit a large number of output variable (genes), we project the data in a linear dimensionally-reduced space to interpolate at component-level, before re-projecting the data back to gene-level. We do this with Principal Components or Independant Components (Independant Component Analysis).

Both PCA and ICA perform the same type of linear transformation on the data (they just optimize different criteria). We get the following :

$$X_{(m\times n)} = G_{(m\times c)}S^T_{(n\times c)}$$

with X, the matrix of m genes by n samples, G the gene loadings (m genes by c components) and S^T the sample scores (n samples by c components). When performing PCA (or ICA) on gene expression data, S is what's usually plotted (e.g. PC1 vs. PC2) to see how samples are grouped in the component space. It's what we plotted earlier in the section on observing data, for instance.

Alter, Brown, and Botstein (2000) demonstrated that singular value decomposition of gene expression data can be taken as "eigengenes", giving a global picture of the expression landscape and dynamics with a few components. GEIMs use this property. We fit a model on the columns of S^T (eigengenes), predict in the component space, and reconstruct the gene expression data by a matrix product with the gene loadings.

We propose 2 model types : Generalized Additive Models (GAMs, the default) and Generalized Linear Models (GLMs). GAMs use `gam()` from the `mgcv` package, and GLMs use the `glm()` function of the `stats` core package.

A standard R formula will be specified for the model. This formula can include the tools and functions generally used with `gam()` or `glm()`, most notably the variety of polynomial or smoothing splines implemented through the `s()` function of `mgcv` for GAMs. GLMs can also use splines from the `splines` core package, such as `ns()` for natural cubic splines.

3.2 The GEIM interface

Gene Expression Interpolation Models (GEIMs), are built with the `ge_im()` function, which outputs a `geim` object. This function inputs 3 key arguments :

- `X` : the gene expression matrix of your time-series (genes as rows, samples as columns)
- `p` : a dataframe of phenotypic data, samples as rows. This should include the age/time variable and any other covariates you want to include in the model (*e.g* batch, strain)
- `formula` : the model formula. This should be a standard R formula, using terms found in `p`. **It must start with `X~`**.

Another important argument is the **n**umber of **c**omponents to interpolate on, `nc`.

For example, using the `dsaeschimann2017` dataset we could build the following model.

```
m_dsaeschimann2017 <- ge_im(X = dsaeschimann2017$g,
                            p = dsaeschimann2017$p,
                            formula = "X ~ s(age, bs = 'ts') + strain",
                            nc = 32)
```

Additional parameters are detailed in the documentation of the function `?ge_im()`.

Note that a single model formula is specified and applied to all the components, but models are fitted independently on each components.

Model predictions can be generated with the `predict()` function, as for any standard R model.

3.3 Defining the appropriate model and parameters

3.3.1 Model type

There are 5 types of GEIMs:

- A GAM on PCA components (`method="gam",dim_red="pca"`) (default)
- A GLM on PCA components (`method="glm",dim_red="pca"`)
- A GAM on ICA components (`method="gam",dim_red="ica"`)
- A GLM on ICA components (`method="glm",dim_red="ica"`)
- A gene-by-gene linear model directly on the gene expression matrix (`method="limma"`)

Our default GEIM is fitting GAMs on PCA components, which is a robust choice when applying a smoothing spline to the data.

When using GAMs, there can be no interaction between terms (by definition). It is possible to include a `by` argument to the `s()` function, which essentially corresponds to separate model fits on each level of the specified group variable (and thus requires enough data to do so).

PCA and ICA interpolation usually yield near-identical results (ICA components tend to be more biologically meaningful or interpretable). ICA tends to outperform PCA when the data is very noisy (by design, since ICA essentially does signal extraction). It is however slower, especially if `nc` is large.

With the last option (`"limma"`), a model is fit per gene with the `lmFit()` function of the `limma` package. This approach makes no effort to reduce the dimensionality of the problem (`dim_red` and `nc` arguments are ignored) which means there is no information loss or bias introduced by dimension reduction. It is however very sensitive to noise.

3.3.2 Model performance

The `mperf()` function computes multiple indices to evaluate model performance.

In the formulas below, X corresponds to the input gene expression matrix (m genes as rows, n samples as columns), $\hat{X}$ to the model predictions. x_i corresponds to row i of matrix X and $x_i^{(j)}$ to sample j of that row. This notation is derived from the general regression problem, where X^T corresponds to the set of m dependant variables to predict.

- `aCC` : average Correlation Coefficient.

$$aCC = \frac{1}{m}\sum_{i=1}^{m} CC = \frac{1}{m}\sum_{i=1}^{m}\frac{\sum_{j=1}^{n}(x_i^{(j)} - \bar{x}_i)(\hat{x}_i^{(j)} - \bar{\hat{x}}_i)}{\sqrt{\sum_{j=1}^{n}(x_i^{(j)} - \bar{x}_i)^2(\hat{x}_i^{(j)} - \bar{\hat{x}}_i)^2}}$$

- `aRE` : average Relative Error.

$$a\delta = \frac{1}{m}\sum_{i=1}^{m}\delta = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n}\sum_{j=1}^{n}\frac{|x_i^{(j)} - \hat{x}_i^{(j)}|}{x_i^{(j)}}$$

- `MSE` : Mean Squared Error.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n}\sum_{j=1}^{n}(x_i^{(j)} - \hat{x}_i^{(j)})^2$$

- `aRMSE` : average Root MSE.

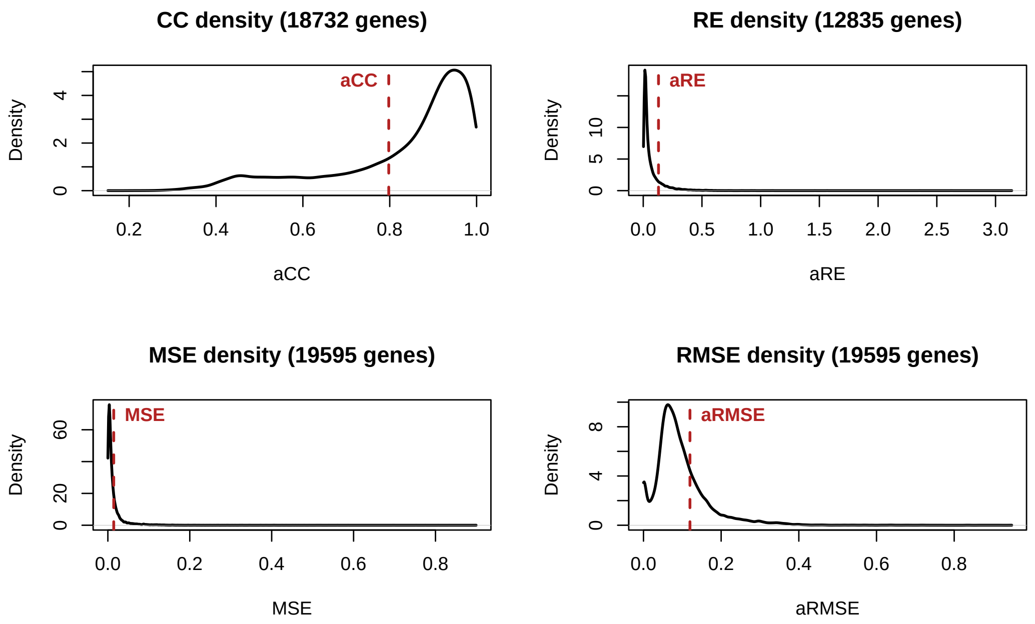
$$aRMSE = \frac{1}{m}\sum_{i=1}^{m} RMSE = \frac{1}{m}\sum_{i=1}^{m}\sqrt{\frac{\sum_{j=1}^{n}(x_i^{(j)} - \hat{x}_i^{(j)})^2}{n}}$$

Note that these indices are computed and averaged *with respect to variables (genes)*, *not observations*. `mperf()` outputs either the overall (averaged) index, or values per-gene (`global` parameter).

```
# global values
g_mp <- mperf(dsaeschimann2017$g, predict(m_dsaeschimann2017), is.t = T)
g_mp
#> $aCC
#> [1] 0.7977299
#>
#> $aRE
#> [1] 0.1301014
#>
#> $MSE
#> [1] 0.01431891
#>
#> $aRMSE
#> [1] 0.1196617

# per gene values
ng_mp <- mperf(dsaeschimann2017$g, predict(m_dsaeschimann2017), is.t = T,
               global = F)
ng_mp <- lapply(ng_mp, na.omit) # remove NAs (eg. 0 variance genes)
ng_mp$aRE <- ng_mp$aRE[ng_mp$aRE < Inf] # remove Inf values (/0)
```

Model performance is reflected by the distributions of prediction indices over all genes.

```
par(mfrow = c(2,2))
invisible(sapply(names(ng_mp), function(idx){
  rg <- range(na.omit(ng_mp[[idx]]))
  # label position
  if(idx == "aCC"){
    pos <- 2
  } else {
    pos <- 4
  }
  # estimate density curve
  d <- density(na.omit(ng_mp[[idx]]), from = rg[1], to = rg[2])

  plot(d, main = paste0(gsub("a", "", idx, fixed = T),
                        " density (", length(ng_mp[[idx]]), " genes)"),
       xlab = idx, lwd = 2)
  # display global value
  abline(v = g_mp[[idx]], lty = 2, lwd = 2, col = "firebrick")
  text(g_mp[[idx]], .9*max(d$y), pos = pos, labels = idx,
       font = 2, col = "firebrick")
}))
```



This data is quite clean, so model fits are very good across the board.

3.3.3 Number of components

By default, the number of components to interpolate is set to the number of samples. However, we recommend setting a cutoff on explained variance of PCA (or ICA) components.

For example, on the `dsaeschimann2017` dataset, we set the threshold at 99% :

```
nc <- sum(summary(pca_dsaeschimann2017)$importance[3,] < .99) + 1
nc
#> [1] 32
```

This threshold must be set in accordance with the noise in the data. For example, in very noisy data, would you consider that 99% of the variance in the dataset corresponds to meaningful information ?

You can also define `nc` by plotting your components and stopping after the components stop capturing meaningful variation (dynamics) with respect to time/age. We define components with 'intelligible dynamics' with respect to time as those where a model fit explains > 0.5 of the deviance (noisy components with no dynamics have poor fits).

In very noisy data, you may have to keep very few components (or even a single component) for the interpolation.

3.3.4 Comparing formulas

Choosing from different splines (and/or parameters) can be done with cross-validation (CV). The `ge_imCV()` function inputs the `X`, `p` and a `formula_list` to test, as well as other potential CV parameters (*e.g.* training set size).

The default training/validation set ratio is `cv.s=0.8`, so 80% of the data is used to build the model and the remainder for validation. When including (factor) covariates in the model, the training set is built such that all groups are proportionately represented in the training set (based on terms of the first formula in the list). The number of CV repeats is defined by `cv.n`.

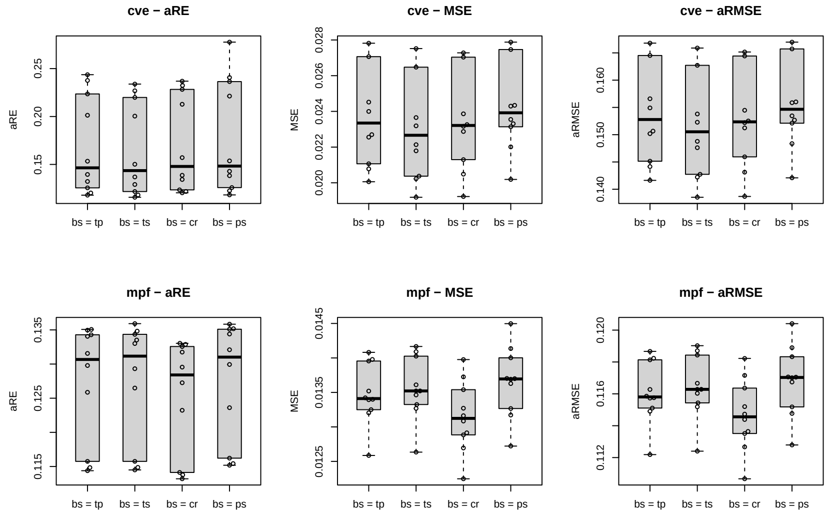
The model type (GAM/GLM and PCA/ICA) is fixed for all formulas in one call of `ge_imCV()`.

`ge_imCV()` returns model performance indices from `mperf()` (excluding `aCC` to reduce computing time). Indices are computed on the validation set (CV Error, cve) *and* on the training set (Model PerFormance, mpf).

Below we choose between 4 available GAM smooth terms on the `dsaeschimann2017` GEIM.

```
smooth_methods <- c("tp", "ts", "cr", "ps")
flist <- as.list(paste0("X ~ s(age, bs = \'", smooth_methods, "\') + strain"))
cat(unlist(flist), sep = '\n')
#> X ~ s(age, bs = 'tp') + strain
#> X ~ s(age, bs = 'ts') + strain
#> X ~ s(age, bs = 'cr') + strain
#> X ~ s(age, bs = 'ps') + strain



set.seed(2)
cv_dsaeschimann2017 <- ge_imCV(X = dsaeschimann2017$g,
                               p = dsaeschimann2017$p,
                               formula_list = flist,
                               cv.n = 10, nc = nc,
                               nb.cores = 4)
#> CV on 4 models. cv.n = 10 | cv.s = 0.8
#>
#> ...Building training sets
#> ...Setting up cluster
#> ...Running CV
#> ...Cleanup and formatting
```

```
plot(cv_dsaeschimann2017, names = paste0("bs = ", smooth_methods), outline = F,
     swarmargs = list(cex = .8), boxwex=.5)
```



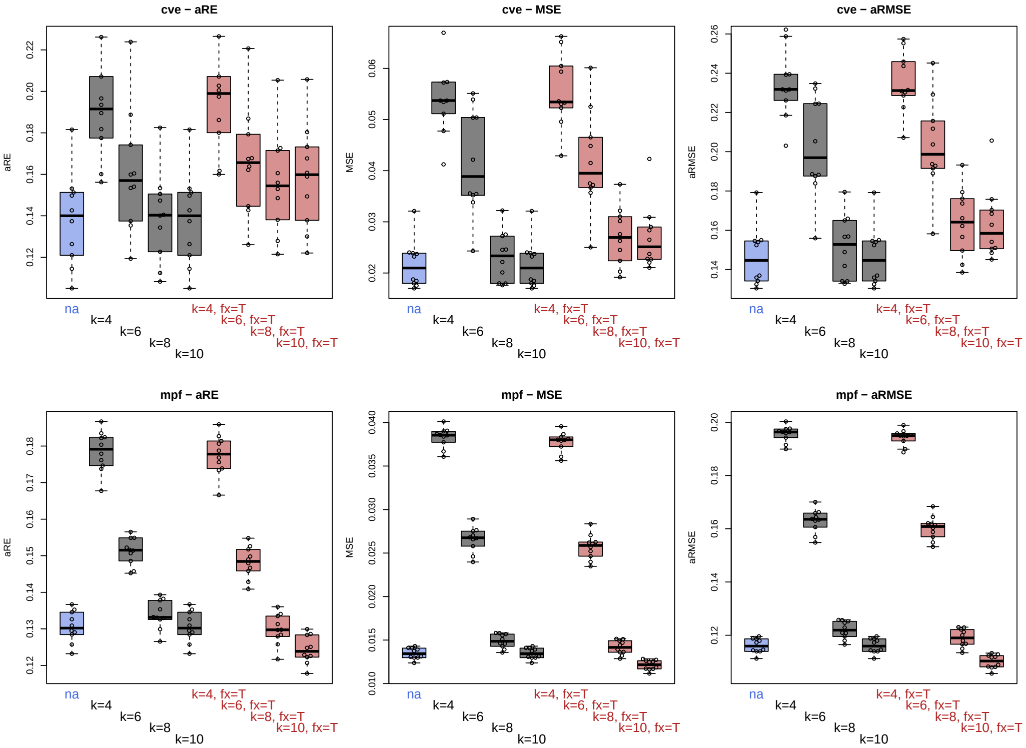
From the plots above, we can see the different splines perform similarly. All could work. We choose ts (a thin-plate regression spline), as it minimizes CV error without much impact on model performance.

Extra spline parameters can also be specified to the model. With s(), you can give the spline's basis dimension (≃ number of knots) to use with the k parameter. By default, the spline is a *penalized spline*, so it will not necessarily use k knots, but it will stay below that value. By setting fx=TRUE, a spline with k basis dimension is forced. Note this fits a spline of dimension k on *all* components, whereas the penalized spline will adjust. The s() or choose.k documentation gives further information.

In our experience, the parameter estimation done by gam() is usually sufficient and rarely requires tweaking. Below are examples of spline parameter tweaking with the dsaeschimann2017 data.

```
ks <- c(4,6,8,10)
flistk <- as.list(c(
  "X ~ s(age, bs =  'cr') + strain",
  paste0("X ~ s(age, bs =  'cr', k = ", ks , ") + strain"),
  paste0("X ~ s(age, bs =  'cr', k = ", ks , ", fx=TRUE) + strain")
  ))
cat(unlist(flistk), sep = '\n')
#> X ~ s(age, bs =  'cr') + strain
#> X ~ s(age, bs =  'cr', k = 4) + strain
#> X ~ s(age, bs =  'cr', k = 6) + strain
#> X ~ s(age, bs =  'cr', k = 8) + strain
#> X ~ s(age, bs =  'cr', k = 10) + strain
#> X ~ s(age, bs =  'cr', k = 4, fx=TRUE) + strain
#> X ~ s(age, bs =  'cr', k = 6, fx=TRUE) + strain
#> X ~ s(age, bs =  'cr', k = 8, fx=TRUE) + strain
#> X ~ s(age, bs =  'cr', k = 10, fx=TRUE) + strain

cv_dsaeschimann2017k <- ge_imCV(X = dsaeschimann2017$g,
                                p = dsaeschimann2017$p,
                                formula_list = flistk,
                                cv.n = 10, nc = nc,
                                nb.cores = 4)
#> CV on 9 models. cv.n = 10 | cv.s = 0.8
#>
#> ...Building training sets
#> ...Setting up cluster
#> ...Running CV
#> ...Cleanup and formatting
```

```
par(mar = c(7,4,3,1))
plot(cv_dsaeschimann2017k,
     names = c("na", paste0("k=", ks, rep(c("", ", fx=T"), each = 4))),
     outline = F,
     col = transp(c("royalblue", rep(c(1, "firebrick"), each = 4))),
     tcol = c("royalblue", rep(c(1, "firebrick"), each = 4)),
     names.arrange = 5, swarmargs = list(cex = .8))
```



This confirms the parameters estimated by GAM (in blue) are optimal.

3.4 Building a Reference object

A ref object can be built from a GEIM using make_ref(), specifying interpolation resolution and relevant metadata.

On our dsaeschimann2017 example :

```
r_dsaeschimann2017 <- make_ref(
  m = m_dsaeschimann2017,        # geim
  n.inter = 100,                 # interpolation resolution
  t.unit = "h past egg-laying (25°C)", # time unit
  cov.levels = list("strain"= "N2"),  # covariate lvls to use for interpolation
  metadata = list("organism"= "C. elegans", # any metadata
                  "profiling"= "whole-organism, bulk",
                  "technology"= "RNAseq"))
```

As any R model, GEIMs also have a predict() function, which can used to manually output predictions either in component space or at the gene level. Doing so manually instead of through make_ref() can be useful for a deeper look at the model.

```
# first generate the new predictor data
n.inter <- 100 # nb of new timepoints
newdat <- data.frame(
  age = seq(min(dsaeschimann2017$p$age),
            max(dsaeschimann2017$p$age), l = n.inter),
  strain = rep("N2", n.inter) # we want to predict as N2
  )
```

```
head(newdat)
#>        age strain
#> 1 18.00000     N2
#> 2 18.20202     N2
#> 3 18.40404     N2
#> 4 18.60606     N2
#> 5 18.80808     N2
#> 6 19.01010     N2


# predict at gene level
pred_m_dsaeschimann2017 <- predict(m_dsaeschimann2017,
                                   newdata = newdat)
# predict at component level
pred_m_dsaeschimann2017_comp <- predict(m_dsaeschimann2017,
                                        newdata = newdat, as.c = TRUE)
```

3.5 Validating the reference

After building a reference, we check interpolation results by:

- Checking model fits on components (plots)
- Staging the samples on their own interpolated data, or better (if possible) stage another independent time-series on your reference for external validation.

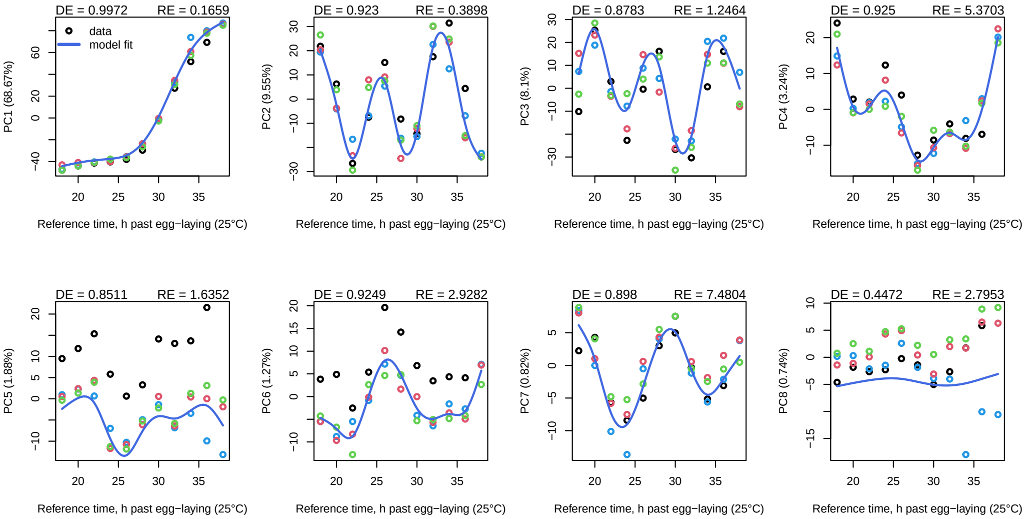
3.5.1 Checking model predictions against components

Checking predictions against components allows you to immediately see if some dynamics get mishandled by the model, or if there is over fitting. It's acceptable to have slight offsets, no fit is perfect.

Plotting a model and a reference object (or equivalent metadata) shows component interpolation, with deviance explained (DE) and relative error (RE) for each component (this information is also returned by the plot function). DE can be used to define components with "intelligible" dynamics (*w.r.t.* time), when $DE > 0.5$. In noisy data, this distinction can be useful to remove components which do not reflect meaningful developmental variation (but rather noise).

Predictions of the first few components from dsaeschimann2017 are plotted below.

```
par(mfrow = c(2,4))
fit_vals <- plot(m_dsaeschimann2017, r_dsaeschimann2017, ncs=1:8,
                 col = dsaeschimann2017$p$strain, col.i = 'royalblue')
```



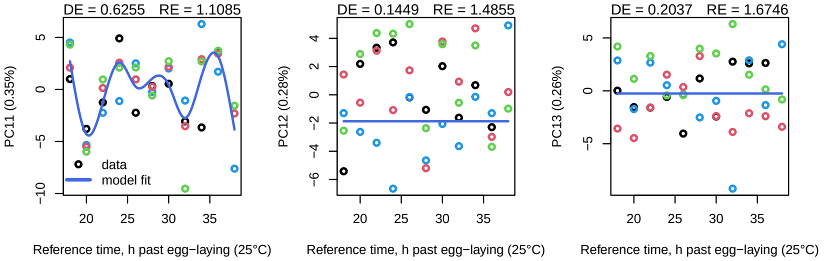
Of note, we are predicting model values as `N2` (lightblue). While all strains are shown on the plots, some model parameters depend on the selected `N2` strain.

The fit values are also returned by the plot function.

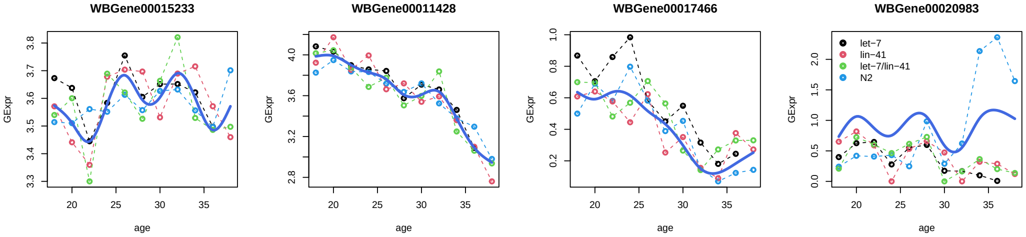
```
head(fit_vals)
#>     component.var.exp        r2 deviance.expl relative.err
#> PC1           0.68673 0.9972253     0.9972253    0.1658575
#> PC2           0.09546 0.9230361     0.9230361    0.3898380
#> PC3           0.08096 0.8783276     0.8783276    1.2464485
#> PC4           0.03242 0.9249808     0.9249808    5.3703222
#> PC5           0.01875 0.8510695     0.8510695    1.6352080
#> PC6           0.01274 0.9249475     0.9249475    2.9281560
```

You may notice some noisy components get "flattened", with a null model fitted. These components can be left in or removed as they generally have little to no impact on interpolation at the gene level (representing a minuscule part of total variance in the data). This can actually get rid of unwanted variation.

```
par(mfrow = c(1,3), pty='s')
fit_vals <- plot(m_dsaeschimann2017, r_dsaeschimann2017, ncs=11:13,
                 col = dsaeschimann2017$p$strain, col.i = 'royalblue',
                 l.pos = 'bottomleft')
```



The interpolation should translate well at the gene level, on the full expression matrix.

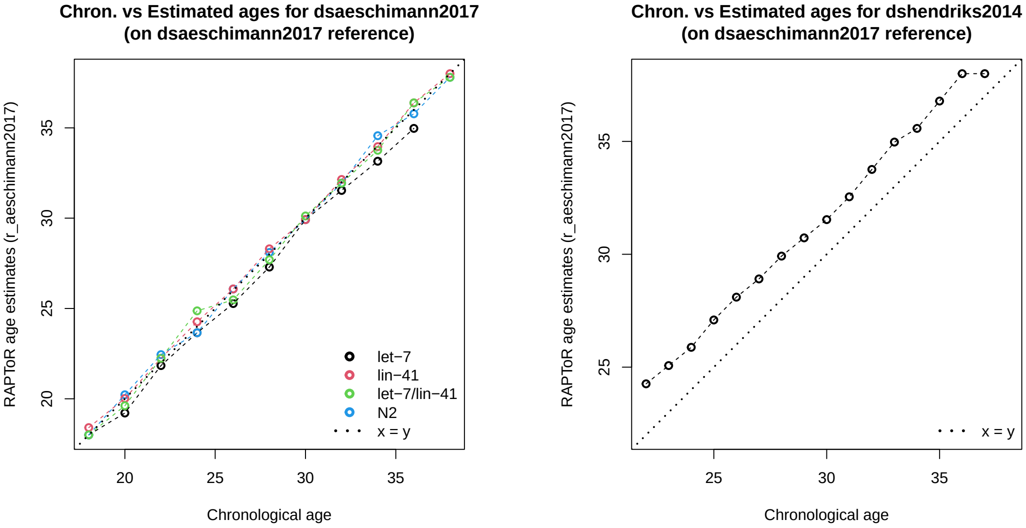


3.5.2 Staging samples

Staging the samples used to build the reference on their interpolated version is a good first test. Then, staging another time-series from the literature on your reference is the best validation, if such data exists. This external validation confirms the interpolated dynamics indeed correspond to development processes and not a dataset-specific effect (which is unlikely, but not impossible).

We can use `dshendriks2014` for external validation.

```
ae_test_dsaeschimann2017 <- ae(dsaeschimann2017$g, r_dsaeschimann2017)
ae_test_dshendriks2014 <- ae(dshendriks2014$g, r_dsaeschimann2017)
```



3.6 About aging references

Unlike development, where gene expression changes are robust across individuals (and to an extent across species), expression changes along aging are much more subtle, more variable and noisier across experiments. Individuals can age at different rates, in different ways, and the process of aging itself is also still poorly understood. This makes it difficult for RAPToR to work with aging data "as is", and we recommend to strengthen the aging signal by restricting genes to an informative set.

Empirically, we have found that genes with monotonous trends along aging make for a robust choice. Genes whose expression correlates with age values above a given threshold can thus be selected to build a reference from a single component. This strategy is what we use in our article (Bulteau and Francesconi (2022)).

A more robust approach is to look for genes that change consistently along aging between multiple independent time-series. This can for example be done with an ICA joining multiple datasets, selecting genes with high loadings on components which behave the same across all included datasets. Of course, this requires multiple independent aging time-series for an organism of interest (the more, the better), which is rarely found.

4 Reference-Building examples

Here are a few examples of reference building and validation on different organisms.

- *C. elegans* larval development
- *D. melanogaster* embryonic development
- *Danio rerio* embryonic development with uneven sampling

4.1 Example 1 - *C. elegans* larval development

4.1.1 Data

In this example, we use two *C. elegans* RNAseq time-series:

1. A high-resolution time series of late larval development published by Hendriks et al. (2014), `dshendriks2014`, used to build the reference. (Accession : GSE52861)
2. A time series of larval development in 4 different strains published by Aeschimann et al. (2017), `dsaeschimann2017`, used for external validation. (Accession : GSE80157)

The data is the same used in the vignette above, but we flip which dataset is the reference and which is used for external validation. Code to generate `dsaeschimann2017` and `dshendriks2014` can be found at the end of this section

4.1.2 Normalization & exploration

We start by normalizing the data.

```
library(RAPToR)
library(limma)

dshendriks2014$g <- limma::normalizeBetweenArrays(dshendriks2014$g,
                                                   method = "quantile")
dshendriks2014$g <- log1p(dshendriks2014$g)

dsaeschimann2017$g <- limma::normalizeBetweenArrays(dsaeschimann2017$g,
                                                     method = "quantile")
dsaeschimann2017$g <- log1p(dsaeschimann2017$g)
```

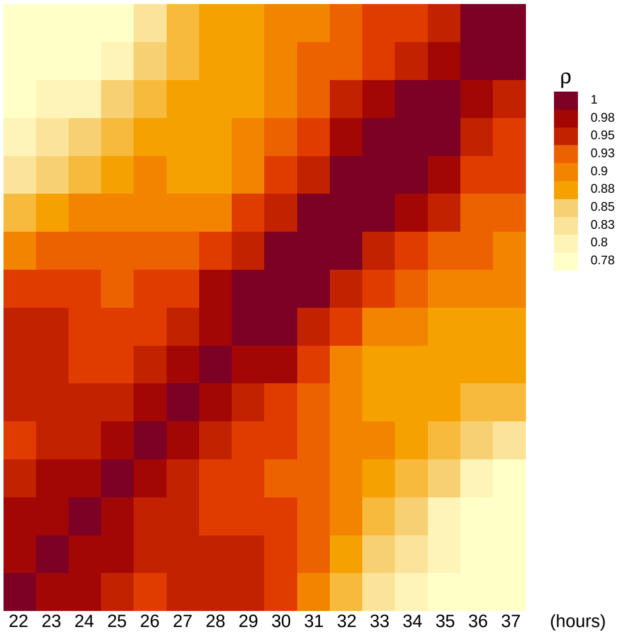
Check the contents of the expression matrix and pheno data.

```
dshendriks2014$g[1:5,1:3]
#>                 contDevA_N2_21h contDevA_N2_22h contDevA_N2_23h
#> WBGene00007063       1.4124620       1.5261100       1.4106924
#> WBGene00007064       1.8814221       2.0860505       2.2545866
#> WBGene00007065       2.7669727       2.6197855       2.5342036
#> WBGene00003525       0.3651526       0.9841291       1.7629401
#> WBGene00007067       1.0067865       0.9129110       0.7664744
```

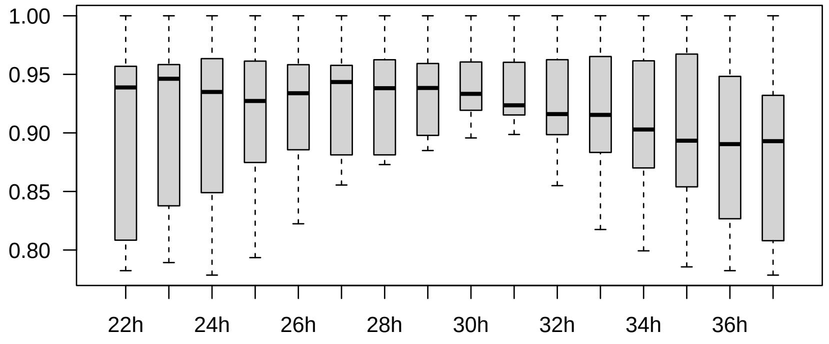
```
head(dshendriks2014$p, n = 5)
#>                       title geo_accession time in development:ch1 age
#> GSM1277118 contDevA_N2_21h    GSM1277118                22 hours  22
#> GSM1277119 contDevA_N2_22h    GSM1277119                23 hours  23
#> GSM1277120 contDevA_N2_23h    GSM1277120                24 hours  24
#> GSM1277121 contDevA_N2_24h    GSM1277121                25 hours  25
#> GSM1277122 contDevA_N2_25h    GSM1277122                26 hours  26
```

Correlation between the samples.

```
cor_dshendriks <- cor(dshendriks2014$g, method = "spearman")
ord <- order(dshendriks2014$p$age)
heatmap(cor_dshendriks[ord, ord],
        Colv = NA, Rowv = NA, scale = "none",
        keep.dendro = F, margins = c(1,5),
        labRow = "", labCol = "")
# add time labels
par(xpd = T)
mtext(text = paste0(dshendriks2014$p$age),
      side = 1, line = 4.1, at = seq(-.17,.86, l = 16), cex = .8)

# add color legend
l.values <- seq(min(cor_dshendriks), max(cor_dshendriks), l = 10)
image(x = c(.95,1), y = seq(0.6,1, l = 10), useRaster = T,
      z = matrix(l.values, ncol = 10),
      col = hcl.colors(12, "YlOrRd", rev = TRUE), add = T)
text(.975, 1, pos = 3, labels = expression(rho), font = 2)
text(1, y = seq(0.6,1, l = 10), pos = 4,
     labels = round(l.values, 2), cex = .6)

mtext(at = 1.0, line = 4.1, side = 1, text = "(hours)", cex = .8)
```



```
boxplot(cor_dshendriks, names = paste0(dshendriks2014$p$age,'h'),
        boxwex=.5, las=1)
```

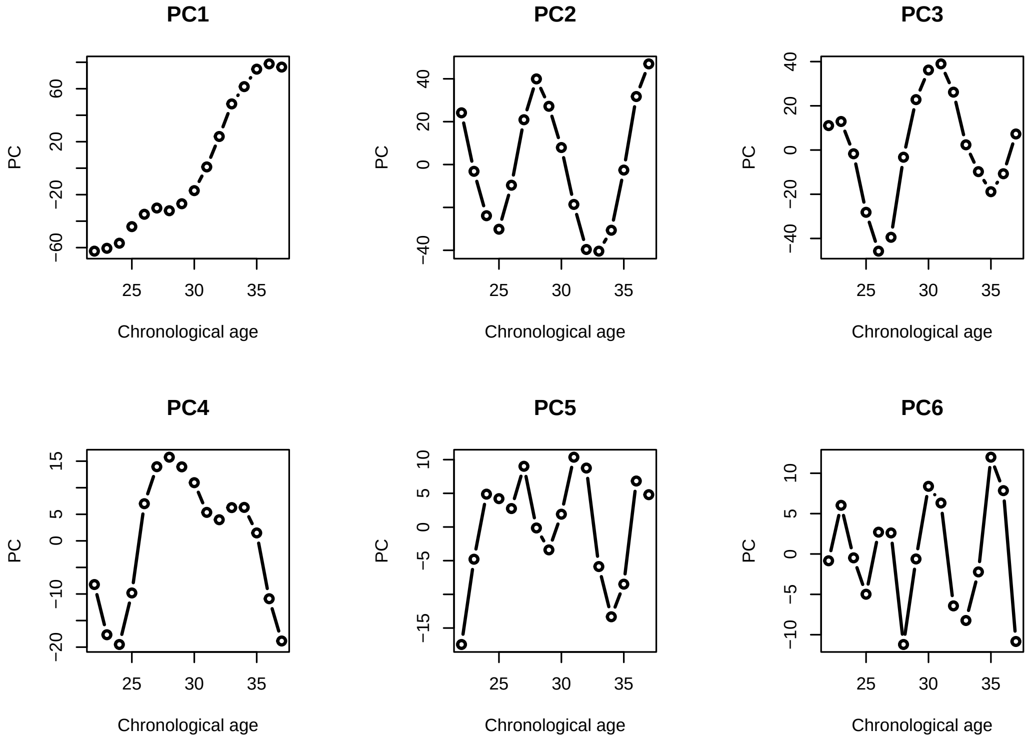


Everything looks good, no outliers.

Now, plotting principal components.

```
pca_dshendriks <- summary(stats::prcomp(t(dshendriks2014$g), rank = 25,
                                        center = TRUE, scale = FALSE))
```

```
par(mfrow = c(2,3), pty='s')
invisible(sapply(1:6, function(i){
  plot(dshendriks2014$p$age, pca_dshendriks$x[,i], lwd = 2, type = 'b',
       xlab = "Chronological age", ylab = "PC", main = paste0("PC", i))
}))
```



4.1.3 Model fitting

We keep enough components to explain 99% of the variance in the data, since it is very clean.

```
nc <- sum(pca_dshendriks$importance[3,] < .99) + 1
nc
#> [1] 10
```

We then fit a GAM model with a cubic spline on age on PCA components.

```
m_dshendriks2014 <- ge_im(X = dshendriks2014$g, p = dshendriks2014$p,
                          formula = "X ~ s(age, bs = 'cr')", nc = nc)
```

4.1.4 Validation

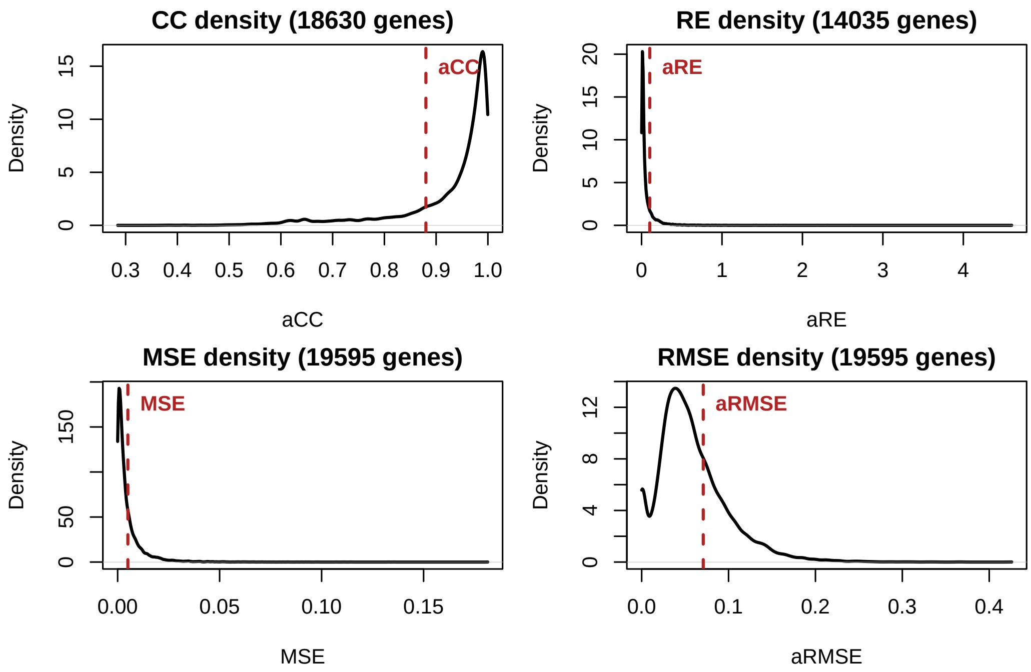
First, check global model performance.

```
# global model performance
mp_hendriks <- mperf(dshendriks2014$g, predict(m_dshendriks2014), is.t = T)
print(do.call(cbind, mp_hendriks))
#>            aCC       aRE          MSE      aRMSE
#> [1,] 0.8801989 0.1018511 0.005021943 0.07086567
```

And then per gene.

```
ng_mp_hendriks <-  mperf(dshendriks2014$g, predict(m_dshendriks2014),
                         is.t = T, global = F)
# remove NAs (eg. 0 variance genes) and Inf values (/0)
ng_mp_hendriks <- lapply(ng_mp_hendriks, na.omit)
ng_mp_hendriks$aRE <- ng_mp_hendriks$aRE[ng_mp_hendriks$aRE < Inf]
```

```
par(mfrow = c(2,2), mar=c(4,4,2,1))
invisible(sapply(names(ng_mp_hendriks), function(idx){
  rg <- range(na.omit(ng_mp_hendriks[[idx]]))
  # estimate density curve
  d <- density(na.omit(ng_mp_hendriks[[idx]]), from = rg[1], to = rg[2])
  plot(d, main = paste0(gsub("a", "", idx, fixed = T),
                        " density (", length(ng_mp_hendriks[[idx]]), " genes)"),
       xlab = idx, lwd = 2)
  # add global value
  abline(v = mp_hendriks[[idx]], lty = 2, lwd = 2, col = "firebrick")
  text(mp_hendriks[[idx]], .9*max(d$y), pos = 4, labels = idx,
       font = 2, col = "firebrick")
}))
```



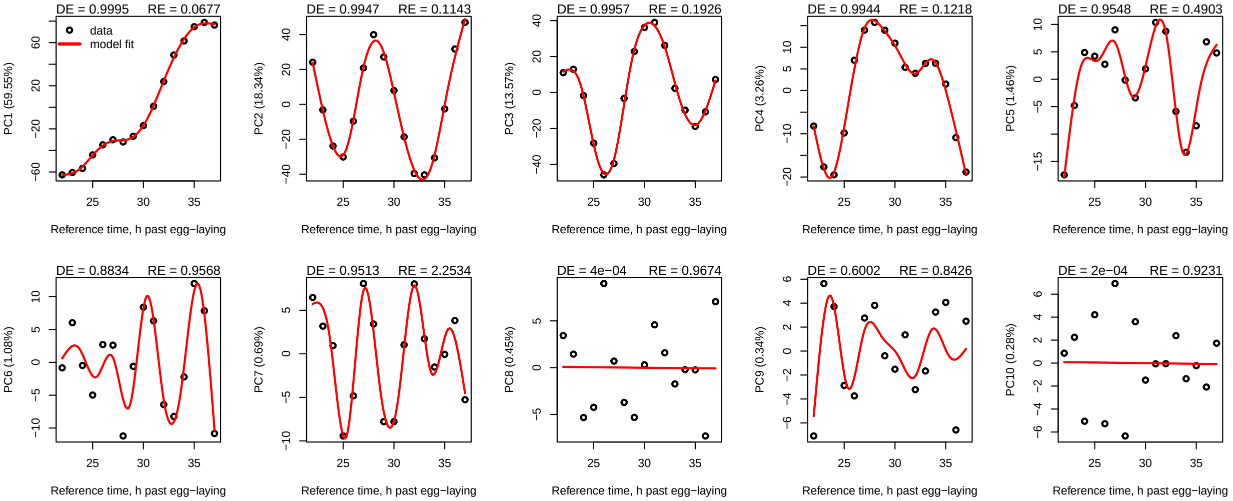
Excellent fits.

Then, build the `ref` object.

```
r_dshendriks2014 <- make_ref(
  m = m_dshendriks2014,
  n.inter = 100,                  # interpolation resolution
  t.unit = "h past egg-laying",   # time unit
  metadata = list("organism" = "C. elegans", # any metadata
                  "profiling" = "whole-organism, bulk",
                  "technology" = "RNAseq"))
```

Plot model predictions on components.

```
par(mfrow = c(2,5), pty='s', mar = c(4,4,3,1))
plot(m_dshendriks2014, r_dshendriks2014, ncs = 1:10)
```

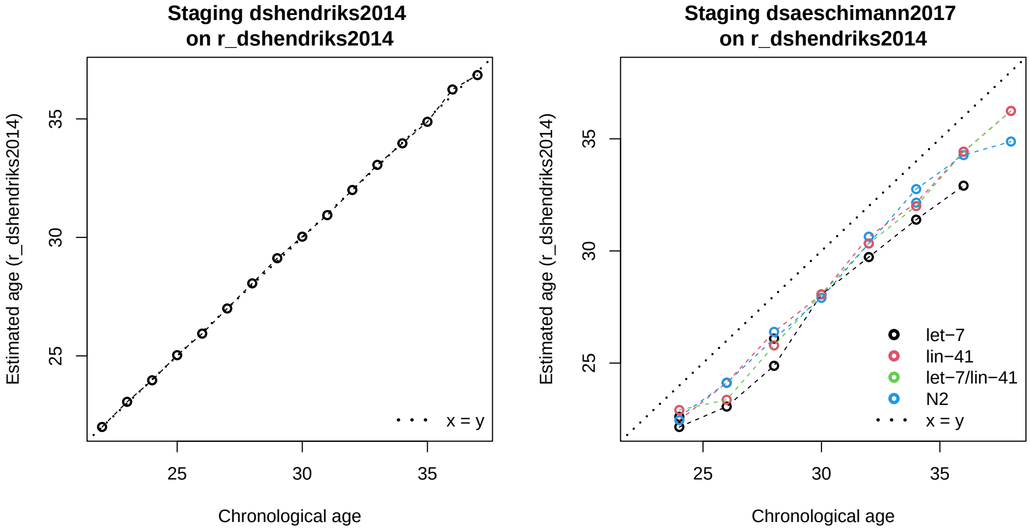


No overfitting, good fits on the components explaining most of the variance. Some components flattened, but minimal associated variance.

Finally, we stage the samples from the reference, as well as the external validation data (`dsaeschimann2017`). We exclude the earliest `dsaeschimann2017` samples (under 22h) from staging since they are outside the span covered by the reference

```
ae_dshendriks2014 <- ae(dshendriks2014$g, r_dshendriks2014)

too_young <- dsaeschimann2017$p$age <= 22
ae_dsaeschimann2017 <- ae(dsaeschimann2017$g[,!too_young], r_dshendriks2014)
```

```
par(mfrow = c(1,2), mar = c(4,5,3,1))
# dshendriks2014
dshendriks2014$p$age_est <- ae_dshendriks2014$age.estimates[,1]
rg <- range(c(dshendriks2014$p$age_est, dshendriks2014$p$age))
plot(age_est~age, data = dshendriks2014$p,
     xlab = "Chronological age", ylab = "Estimated age (r_dshendriks2014)",
     xlim = rg, ylim = rg,
     main = "Staging dshendriks2014
on r_dshendriks2014", lwd = 2)
points(age_est~age, data = dshendriks2014$p, type = 'l', lty = 2)
abline(a = 0, b = 1, lty = 3, lwd = 2) # x=y
legend("bottomright", legend = "x = y", lwd=3, col=1, lty = 3, bty='n')

# dsaeschimann2017
dsaeschimann2017$p[!too_young, "age_est"] <-
  ae_dsaeschimann2017$age.estimates[,1]
rg <- range(c(dsaeschimann2017$p$age_est[!too_young], dsaeschimann2017$p$age[!too_young]))
plot(age_est~age, data = dsaeschimann2017$p[!too_young,],
     xlab = "Chronological age", ylab = "Estimated age (r_dshendriks2014)",
     xlim = rg, ylim = rg,
     main = "Staging dsaeschimann2017
on r_dshendriks2014",
     lwd = 2, col = factor(dsaeschimann2017$p$strain))
invisible(sapply(levels(dsaeschimann2017$p$strain), function(l){
  s <- dsaeschimann2017$p$strain == l & !too_young
  points(age_est~age, data = dsaeschimann2017$p[s,],
         type = 'l', lty = 2,
         col = which(l==levels(factor(dsaeschimann2017$p$strain))))
}))
abline(a = 0, b = 1, lty = 3, lwd = 2)
legend("bottomright", bty='n',
       legend = c("let-7", "lin-41", "let-7/lin-41", "N2", "x = y"),
       lwd=3, col=c(1:4, 1), pch = c(1,1,1,1,NA), lty = c(rep(NA, 4), 3))
```



4.1.5 Code to generate objects

Required packages and variables:

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

requireNamespace("wormRef", quietly = T)
requireNamespace("utils", quietly = T)
requireNamespace("GEOquery", quietly = T) # bioconductor
requireNamespace("Biobase", quietly = T)  # bioconductor
```

Note : set `data_folder` 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)))
}
```

To build dshendriks2014, *C. elegans* late larval development time series from Hendriks et al. (2014)

```
geo_dshendriks2014 <- "GSE52861"

g_url_dshendriks2014 <- GEOquery::getGEOSuppFiles(geo_dshendriks2014,
                                                  makeDirectory = FALSE,
                                                  fetch_files = FALSE)
g_file_dshendriks2014 <- paste0(data_folder, "dshendriks2014.txt.gz")
utils::download.file(url = as.character(g_url_dshendriks2014$url[2]),
                     destfile = g_file_dshendriks2014)

X_dshendriks2014 <- read.table(gzfile(g_file_dshendriks2014),
                               h=T, sep = '\t', stringsAsFactors = F,
                               row.names = 1)

# convert to tpm & wb_id
X_dshendriks2014 <- X_dshendriks2014[
  rownames(X_dshendriks2014)%in%wormRef::Cel_genes$wb_id,]
X_dshendriks2014 <- raw2tpm(
  rawcounts = X_dshendriks2014,
  genelengths = wormRef::Cel_genes$transcript_length[
    match(rownames(X_dshendriks2014), wormRef::Cel_genes$wb_id)])


# pheno data
P_dshendriks2014 <- Biobase::pData(GEOquery::getGEO(geo_dshendriks2014,
                                                    getGPL = F)[[1]])

# filter relevant fields/samples
P_dshendriks2014 <- P_dshendriks2014[
  (P_dshendriks2014$`strain:ch1` == 'N2') &
  (P_dshendriks2014$`growth protocol:ch1` == 'Continuous'), ]

P_dshendriks2014 <- P_dshendriks2014[, c("title", "geo_accession",
                                         "time in development:ch1")]

# get age from sample name
P_dshendriks2014$age <- as.numeric(
  sub('(\\d+)\\shours', '\\1', P_dshendriks2014$`time in development:ch1`))


# formatting
P_dshendriks2014$title <- gsub('RNASeq_polyA_', '',
                               gsub('hr', 'h',
                                    gsub('-', '.', fixed = T,
                                         as.character(P_dshendriks2014$title))))
colnames(X_dshendriks2014) <- gsub('RNASeq_polyA_','',
                                   colnames(X_dshendriks2014))
X_dshendriks2014 <- X_dshendriks2014[, P_dshendriks2014$title]

# save data
```

```
dshendriks2014 <- list(g = X_dshendriks2014, p = P_dshendriks2014)
save(dshendriks2014, file = paste0(data_folder, "dshendriks2014.RData"),
     compress = "xz")

# cleanup
file.remove(g_file_dshendriks2014)
rm(geo_dshendriks2014, g_url_dshendriks2014, g_file_dshendriks2014,
   X_dshendriks2014, P_dshendriks2014)
```

To build `dsaeschimann2017`, *C. elegans* larval development time series of 4 strains from Aeschimann et al. (2017)

```
geo_dsaeschimann2017 <- "GSE80157"

g_url_dsaeschimann2017 <- GEOquery::getGEOSuppFiles(geo_dsaeschimann2017,
                                                    makeDirectory = FALSE,
                                                    fetch_files = FALSE)
g_file_dsaeschimann2017 <- paste0(data_folder, "dsaeschimann2017.txt.gz")
utils::download.file(url = as.character(g_url_dsaeschimann2017$url[2]),
                     destfile = g_file_dsaeschimann2017)

X_dsaeschimann2017 <- read.table(gzfile(g_file_dsaeschimann2017),
                                 h=T, sep = '\t', stringsAsFactors = F,
                                 row.names = 1)

# convert to tpm & wb_id
X_dsaeschimann2017 <- X_dsaeschimann2017[
  rownames(X_dsaeschimann2017) %in% wormRef::Cel_genes$wb_id,]

X_dsaeschimann2017 <- raw2tpm(
  rawcounts = X_dsaeschimann2017,
  genelengths = wormRef::Cel_genes$transcript_length[
    match(rownames(X_dsaeschimann2017), wormRef::Cel_genes$wb_id)])

# pheno data
P_dsaeschimann2017 <- Biobase::pData(GEOquery::getGEO(geo_dsaeschimann2017,
                                                      getGPL = F)[[1]])
P_dsaeschimann2017[,10:34] <- NULL
P_dsaeschimann2017[, 3:8] <- NULL

colnames(P_dsaeschimann2017)[4] <- "strain"
P_dsaeschimann2017$strain <- factor(P_dsaeschimann2017$strain)
P_dsaeschimann2017$title <- make.names(P_dsaeschimann2017$title)

# get age from sample name
P_dsaeschimann2017$age <- as.numeric(
  sub('(\\d+)\\shours', '\\1', P_dsaeschimann2017$`time in development:ch1`))

# formatting
colnames(X_dsaeschimann2017) <- gsub('RNASeq_riboM_', '',
                                     colnames(X_dsaeschimann2017), fixed = T)
P_dsaeschimann2017$title <- gsub('RNASeq_riboM_', '',
                                 P_dsaeschimann2017$title, fixed = T)
X_dsaeschimann2017 <- X_dsaeschimann2017[, P_dsaeschimann2017$title]

# save data
dsaeschimann2017 <- list(g = X_dsaeschimann2017, p = P_dsaeschimann2017)

save(dsaeschimann2017, file = paste0(data_folder, "dsaeschimann2017.RData"),
     compress = "xz")

# cleanup
file.remove(g_file_dsaeschimann2017)
rm(geo_dsaeschimann2017, g_url_dsaeschimann2017, g_file_dsaeschimann2017,
   X_dsaeschimann2017, P_dsaeschimann2017)
```

4.2 Example 2 - *D. melanogaster* embryonic development

4.2.1 Data

In this example, we build a reference for *Drosophila melanogaster* embryo development. We use the following time-series:

1. A bulk embryo development time-series, part of the modENCODE project and published by Graveley et al. (2011), `dsgraveley2011`, used to build the reference. (Data from fruitfly.org)
2. A high-resolution time-series of single embryos published by Levin et al. (2016), `dslevin2016dmel`, used for external validation. (Accession : GSE60471)

Code to generate `dsgraveley2011` and `dslevin2016dmel` can be found at the end of this section

4.2.2 Normalization & exploration

We start by normalizing the data.

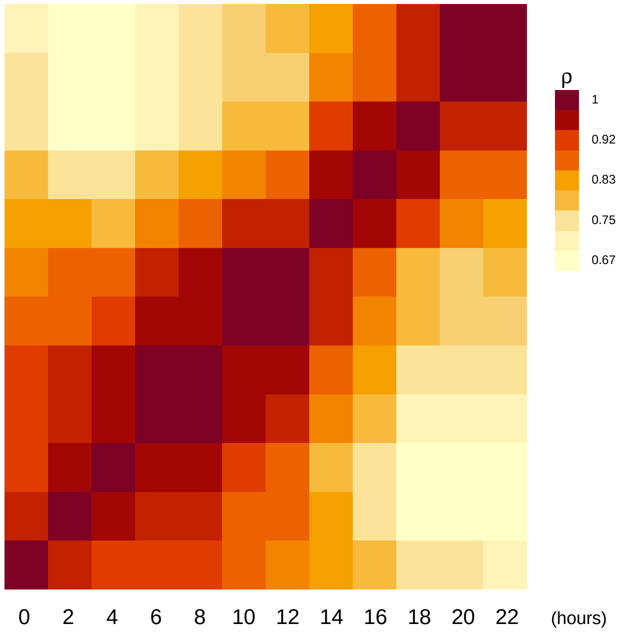
```
library(RAPToR)
library(limma)

dsgraveley2011$g <- limma::normalizeBetweenArrays(dsgraveley2011$g,
                                                  method = "quantile")
dsgraveley2011$g <- log1p(dsgraveley2011$g)

dslevin2016dmel$g <- limma::normalizeBetweenArrays(dslevin2016dmel$g,
                                                   method = "quantile")
dslevin2016dmel$g <- log1p(dslevin2016dmel$g)
```

Check the contents of the expression matrix and pheno data.

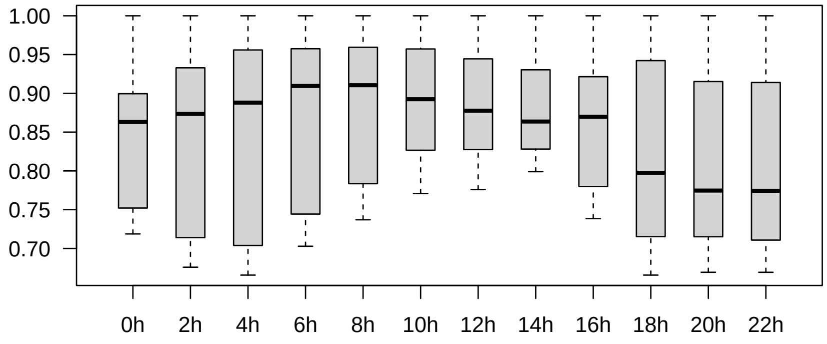
```
dsgraveley2011$g[1:5, 1:5]
#>               em0.2hr   em2.4hr   em4.6hr   em6.8hr  em8.10hr
#> FBgn0000003 3.9688855 4.2758251 3.3179427 4.5643881 4.6995131
#> FBgn0000008 1.2963291 0.9221436 0.6952683 0.6472183 0.7443754
#> FBgn0000014 0.5099048 0.9511344 1.3960100 1.8617929 1.8433359
#> FBgn0000015 0.2435604 0.6427720 1.0519592 1.1089745 1.0195901
#> FBgn0000017 1.7970494 2.0885522 1.3390407 1.5331338 1.6763588

head(dsgraveley2011$p, n = 5)
#>      sname age
#> 1  em0.2hr   0
#> 2  em2.4hr   2
#> 3  em4.6hr   4
#> 4  em6.8hr   6
#> 5 em8.10hr   8
```

Correlation between the samples for outliers.

```
cor_dsgraveley2011 <- cor(dsgraveley2011$g, method = "spearman")
ord <- order(dsgraveley2011$p$age)

heatmap(cor_dsgraveley2011[ord, ord], Colv = NA, Rowv = NA,
        scale = "none", keep.dendro = F, margins = c(2,5),
        labRow = "", labCol = "")
# add time label
par(xpd = T)
mtext(text = dsgraveley2011$p$age, side = 1,
      line = 4, at = seq(-.16,.85, l = 12))
mtext(at = 1, line = 4, side = 1, text = "(hours)", cex = .8)

# add color legend
l.values <- seq(min(cor_dsgraveley2011), max(cor_dsgraveley2011), l = 9)
image(x = c(.95,1), y = seq(0.6,1, l = 9), useRaster = T,
      z = matrix(l.values, ncol = 9),
      col = hcl.colors(12, "YlOrRd", rev = TRUE), add = T)
text(.975, 1, pos = 3, labels = expression(rho), font = 2)
text(1, y = seq(0.6,1, l = 9)[c(T,F)], pos = 4,
     labels = round(l.values[c(T,F)], 2), cex = .6)
```



```
boxplot(cor_dsgraveley2011, names = paste0(dsgraveley2011$p$age,'h'),
        boxwex=.5, las=1)
```

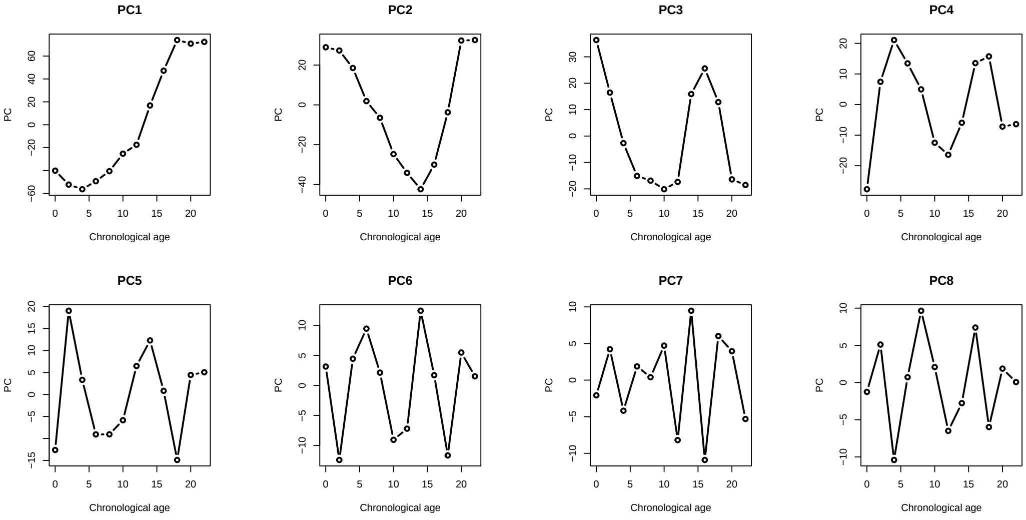


No outliers.

Plotting principal components.

```
pca_dsgraveley2011 <- summary(stats::prcomp(t(dsgraveley2011$g), rank = 12,
                                            center = TRUE, scale = FALSE))
```

```
par(mfrow = c(2,4), pty = 's')
invisible(sapply(seq_len(8), function(i){
  plot(dsgraveley2011$p$age, pca_dsgraveley2011$x[,i], type = 'b', lwd = 2,
       xlab = "Chronological age", ylab = "PC", main = paste0("PC", i))
}))
```



4.2.3 Model fitting

We keep enough components to explain 99% of the variance in the data.

```
nc <- sum(pca_dsgraveley2011$importance[3,] < .99) + 1
nc
#> [1] 8
```

We then fit a GAM on PCA components with a cubic spline on age.

```
m_dsgraveley2011 <- ge_im(X = dsgraveley2011$g,
                          p = dsgraveley2011$p,
                          formula = "X ~ s(age, bs = 'cr')", nc = nc)
```

4.2.4 Validation

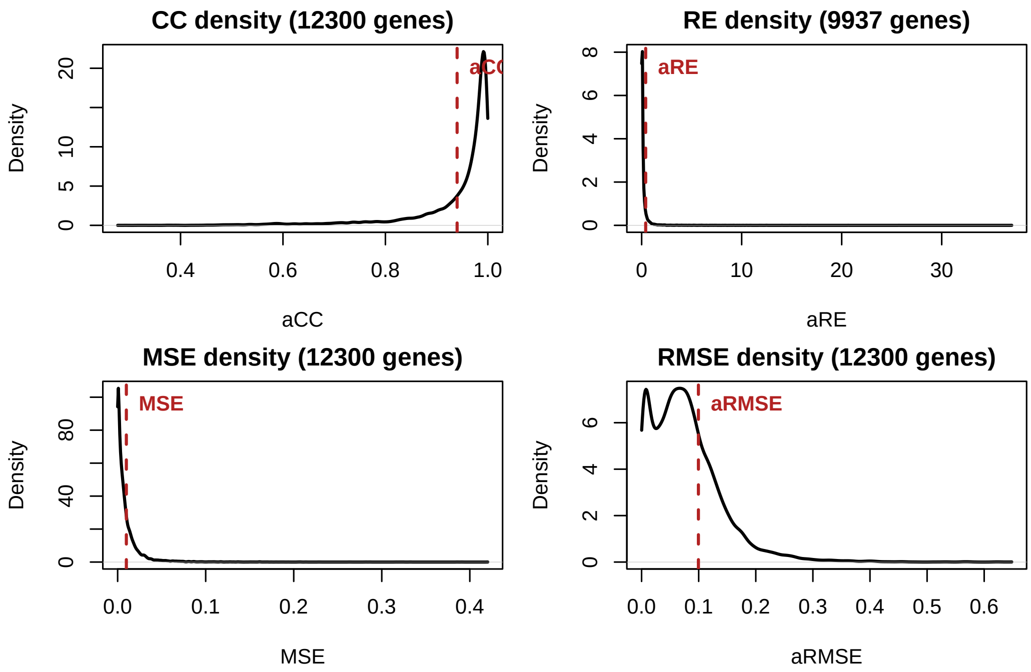
First, check global model performance.

```
# global model performance
mp_grav <- mperf(dsgraveley2011$g, predict(m_dsgraveley2011), is.t = T)
print(do.call(cbind, mp_grav))
#>            aCC       aRE         MSE      aRMSE
#> [1,] 0.9400274 0.4040618 0.009908293 0.09954041
```

And then per gene.

```
ng_mp_grav <-  mperf(dsgraveley2011$g, predict(m_dsgraveley2011),
                     is.t = T, global = F)
# remove NAs (eg. 0 variance genes) and Inf values (/0)
ng_mp_grav <- lapply(ng_mp_grav, na.omit)
ng_mp_grav$aRE <- ng_mp_grav$aRE[ng_mp_grav$aRE < Inf]
```

```
par(mfrow = c(2,2), mar=c(4,4,2,1))
invisible(sapply(names(ng_mp_grav), function(idx){
  rg <- range(na.omit(ng_mp_grav[[idx]]))
  # estimate density curve
  d <- density(na.omit(ng_mp_grav[[idx]]), from = rg[1], to = rg[2])
  plot(d, main = paste0(gsub("a", "", idx, fixed = T),
                        " density (", length(ng_mp_grav[[idx]]), " genes)"),
       xlab = idx, lwd = 2)
  # add global value
  abline(v = mp_grav[[idx]], lty = 2, lwd = 2, col = "firebrick")
  text(mp_grav[[idx]], .9*max(d$y), pos = 4, labels = idx,
       font = 2, col = "firebrick")
}))
```



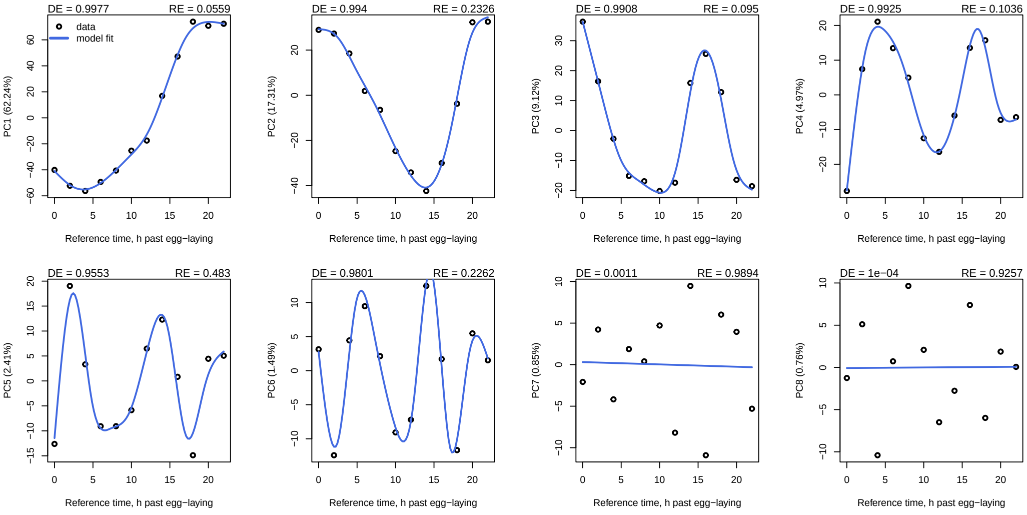
Excellent fits.

Then, build the `ref` object.

```
r_dsgraveley2011 <- make_ref(
  m = m_dsgraveley2011,
  n.inter = 100,                # interpolation resolution
  t.unit = "h past egg-laying", # time unit
  metadata = list("organism" = "D. melanogaster", # any metadata
                  "profiling" = "whole-organism, bulk",
                  "technology" = "RNAseq"))
```

Plot model predictions on components.

```
par(mfrow = c(2,4), pty='s', mar = c(4,4,3,1))
plot(m_dsgraveley2011, r_dsgraveley2011, col.i = 'royalblue')
```



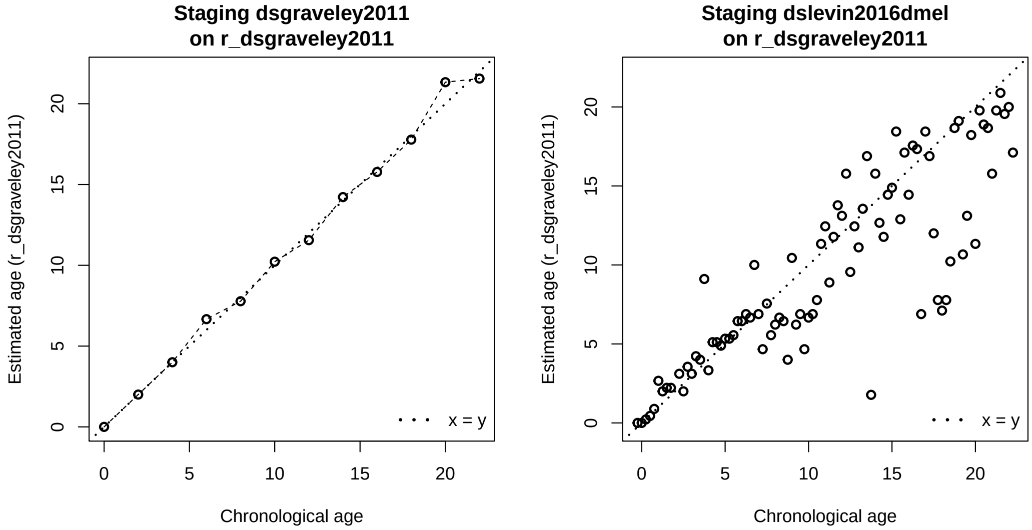
No overfitting, good fits on the components explaining most of the variance. Some components flattened, but minimal associated variance.

Finally, we stage the samples from the reference, as well as the external validation data (`dslevin2016dmel`).

```
ae_dsgraveley2011 <- ae(dsgraveley2011$g, r_dsgraveley2011)
#> Bootstrap set size is 4100
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
ae_dslevin2016dmel <- ae(dslevin2016dmel$g, r_dsgraveley2011)
#>                 nb.genes
#> refdata            12300
#> samp               15072
#> intersect.genes    12096
#> Bootstrap set size is 4032
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
```

```
par(mfrow = c(1,2), mar = c(4,5,3,1))
# dsgraveley2011
dsgraveley2011$p$age_est <- ae_dsgraveley2011$age.estimates[,1]
rg <- range(c(dsgraveley2011$p$age_est, dsgraveley2011$p$age))
plot(age_est~age, data = dsgraveley2011$p,
     xlab = "Chronological age", ylab = "Estimated age (r_dsgraveley2011)",
     xlim = rg, ylim = rg, lwd = 2,
     main = "Staging dsgraveley2011
on r_dsgraveley2011")
```

```
points(age_est~age, data = dsgraveley2011$p, type = 'l', lty = 2)
abline(a = 0, b = 1, lty = 3, lwd = 2) # x=y
legend("bottomright", legend = "x = y", lwd=3, col=1, lty = 3, bty='n')

# dslevin2016dmel
dslevin2016dmel$p$age_est <- ae_dslevin2016dmel$age.estimates[,1]
rg <- range(c(dslevin2016dmel$p$age_est , dslevin2016dmel$p$age))
plot(age_est~age, data = dslevin2016dmel$p,
     xlab = "Chronological age", ylab = "Estimated age (r_dsgraveley2011)",
     xlim = rg, ylim = rg, lwd = 2,
     main = "Staging dslevin2016dmel
on r_dsgraveley2011")
abline(a = 0, b = 1, lty = 3, lwd = 2) #x=y
legend("bottomright", legend = "x = y", lwd=3, col=1, lty = 3, bty='n')
```

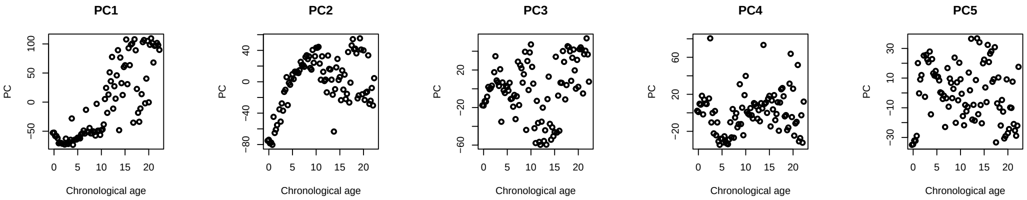


We see that the validation data age estimates vary from their chronological age, particularly in later stages. However, this is not due to errors in the reference or staging, but instead to the fact this data is single-embryo profiling. Indeed, inter-individual variation in developmental speed results in striking differences with expected age, where it would otherwise be averaged out in bulk data.

The dynamics of the `dslevin2016dmel` data further confirm this.

```
pca_dslevin2016dmel <- stats::prcomp(t(dslevin2016dmel$g), rank = 20,
                                     center = TRUE, scale = FALSE)
```

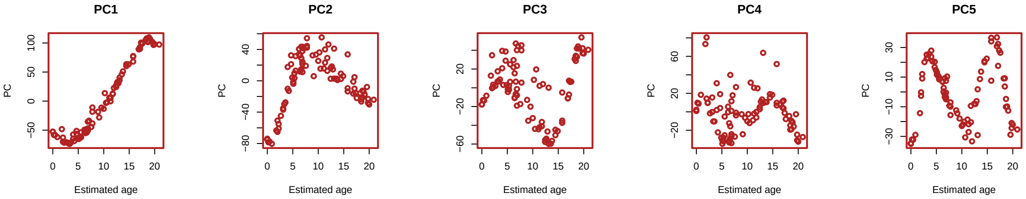
```
par(mfrow = c(1,5), pty='s')
invisible(sapply(1:5, function(i){
    plot(dslevin2016dmel$p$age, pca_dslevin2016dmel$x[,i], lwd = 2,
         xlab = "Chronological age", ylab = "PC", main = paste0("PC", i))
}))
```



Chronological age specified for the samples is inaccurate, and does not properly reflect developmental gene expression dynamics, as evidenced by noisy components.

Inferred age however, clearly restores the dynamics cleanly.

```
par(mfrow = c(1,5), pty='s')
invisible(sapply(1:5, function(i){
    plot(dslevin2016dmel$p$age_est, pca_dslevin2016dmel$x[,i], lwd = 2,
         xlab = "Estimated age", ylab = "PC", main = paste0("PC", i),
         col = 'firebrick')
  box(col = 'firebrick', lwd=2)
}))
```



This example shows how inter-individual variation in developmental speed makes it difficult to profile high-temporal-resolution time series. Furthermore, the temporal resolution difference between the reference and validation data also demonstrates the effectiveness of gene expression interpolation.

4.2.5 Code to generate objects

Required packages and variables:

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

requireNamespace("utils", quietly = T)
requireNamespace("GEOquery", quietly = T) # bioconductor
requireNamespace("Biobase", quietly = T)  # bioconductor
requireNamespace("biomaRt", quietly = T)  # bioconductor
```

Note : set `data_folder` *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)))
}
```

Get *Drosophila* gene ID table from ensembl.

```
mart <- biomaRt::useMart("ensembl", dataset = "dmelanogaster_gene_ensembl")
droso_genes <- biomaRt::getBM(attributes = c("ensembl_gene_id",
                                             "ensembl_transcript_id",
                                             "external_gene_name",
                                             "transcript_end",
                                             "transcript_start"),
                              mart = mart)
droso_genes$transcript_length <-
  droso_genes$transcript_end - droso_genes$transcript_start
droso_genes <- droso_genes[,c(1:3,6)]
colnames(droso_genes)[1:3] <- c("fb_id", "transcript_id", "gene_name")

rm(mart)
```

To build dsgraveley2011, bulk *D. melanogaster* embryo time series from Graveley et al. (2011)

```
g_url_dsgraveley2011 <- paste0(
  "ftp://ftp.fruitfly.org/pub/download/modencode_expression_scores/",
  "Celniker_Drosophila_Annotation_20120616_1428_allsamps_",
  "MEAN_gene_expression.csv.gz")
g_file_dsgraveley2011 <- paste0(data_folder, "dsgraveley2011.csv.gz")
utils::download.file(g_url_dsgraveley2011, destfile = g_file_dsgraveley2011)


X_dsgraveley2011 <- read.table(gzfile(g_file_dsgraveley2011),
                               sep = ',', row.names = 1, h = T)

# convert gene ids to FBgn
X_dsgraveley2011 <- RAPToR::format_ids(X_dsgraveley2011, droso_genes,
                                       from = "gene_name", to = "fb_id")

# select embryo time series samples
X_dsgraveley2011 <- X_dsgraveley2011[,1:12]

P_dsgraveley2011 <- data.frame(
  sname = colnames(X_dsgraveley2011),
  age = as.numeric(gsub("em(\\d+)\\.\\d+hr", "\\1",
                        colnames(X_dsgraveley2011))),
  stringsAsFactors = FALSE)

dsgraveley2011 <- list(g = X_dsgraveley2011, p = P_dsgraveley2011)

save(dsgraveley2011,
     file = paste0(data_folder, "dsgraveley2011.RData"), compress = "xz")

# cleanup
file.remove(g_file_dsgraveley2011)
```

```
rm(g_url_dsgraveley2011, g_file_dsgraveley2011,
   X_dsgraveley2011, P_dsgraveley2011)
```

To build `dslevin2016dmel`, single-embryo *D. melanogaster* embryo time series from Levin et al. (2016)

```
geo_dslevin2016dmel <- "GSE60471"
g_url_dslevin2016dmel <- GEOquery::getGEOSuppFiles(geo_dslevin2016dmel,
                                                   makeDirectory = FALSE,
                                                   fetch_files = FALSE)
g_file_dslevin2016dmel <- paste0(data_folder, "dslevin2016dmel.txt.gz")
utils::download.file(url = as.character(g_url_dslevin2016dmel$url[3]),
                     destfile = g_file_dslevin2016dmel)

X_dslevin2016dmel <- read.table(gzfile(g_file_dslevin2016dmel), h = T,
                                sep = '\t', as.is = T, row.names = 1,
                                comment.char = "")

# filter poor quality samples
cm_dslevin2016dmel <- RAPToR::cor.gene_expr(X_dslevin2016dmel,
                                            X_dslevin2016dmel)
f_dslevin2016dmel <- which(0.6 > apply(cm_dslevin2016dmel, 1,
                                       quantile, probs = .99))
X_dslevin2016dmel <- X_dslevin2016dmel[, -f_dslevin2016dmel]

# convert to tpm & FBgn
X_dslevin2016dmel <- X_dslevin2016dmel[
  rownames(X_dslevin2016dmel)%in%droso_genes$fb_id,]
X_dslevin2016dmel <- raw2tpm(
  rawcounts = X_dslevin2016dmel,
  genelengths = droso_genes$transcript_length[
    match(rownames(X_dslevin2016dmel), droso_genes$fb_id)])

# pheno data
P_dslevin2016dmel <- Biobase::pData(GEOquery::getGEO(geo_dslevin2016dmel,
                                                     getGPL = F)[[1]])

# filter relevant fields/samples
P_dslevin2016dmel <- P_dslevin2016dmel[,
  c("title", "geo_accession", "time (minutes cellularization stage):ch1")]
colnames(P_dslevin2016dmel)[3] <- "time"
P_dslevin2016dmel$title <- as.character(P_dslevin2016dmel$title)

P_dslevin2016dmel <- P_dslevin2016dmel[
  P_dslevin2016dmel$title %in% colnames(X_dslevin2016dmel),]
X_dslevin2016dmel <- X_dslevin2016dmel[, P_dslevin2016dmel$title]

# formatting
P_dslevin2016dmel$title <- gsub('Metazome_Drosophila_timecourse_', '',
                                P_dslevin2016dmel$title)
colnames(X_dslevin2016dmel) <- P_dslevin2016dmel$title
P_dslevin2016dmel$age <- as.numeric(P_dslevin2016dmel$time) / 60
```

```
# save data
dslevin2016dmel <- list(g = X_dslevin2016dmel, p = P_dslevin2016dmel)
save(dslevin2016dmel, file = paste0(data_folder, "dslevin2016dmel.RData"), compress = "xz")

# cleanup
file.remove(g_file_dslevin2016dmel)
rm(geo_dslevin2016dmel, g_url_dslevin2016dmel, g_file_dslevin2016dmel,
   X_dslevin2016dmel, P_dslevin2016dmel,
   cm_dslevin2016dmel, f_dslevin2016dmel)
```

4.3 Example 3 - Uneven sampling of *D. rerio* embryos

4.3.1 Data

This example uses two *Danio rerio* (zebrafish) embryo development time-series:

1. A time-series of zebrafish embryonic development (with uneven time sampling) published by White et al. (2017), `dswhite2017`, used to build the reference. (Data accessible from the publication)
2. A high-resolution time-series of embryonic development published by Levin et al. (2016), `dslevin2016zeb`, used for external validation. (Accession : GSE60619)

Code to generate `dswhite2017` and `dslevin2016zeb` can be found at the end of this section

The reference data (`dswhite2017`) has uneven time sampling, as can often be the case to account for differences in dynamic ranges of expression: later time points are more sparse. We can apply a transformation on the time values (using `asinh()` in this case) so they become more uniform and thus avoid interpolation bias.

4.3.2 Normalization & Quick look

We start by normalizing the data.

```
library(RAPToR)
library(limma)

dswhite2017$g <- limma::normalizeBetweenArrays(dswhite2017$g,
                                               method = "quantile")
dswhite2017$g <- log1p(dswhite2017$g)

dslevin2016zeb$g <- limma::normalizeBetweenArrays(dslevin2016zeb$g,
                                                  method = "quantile")
dslevin2016zeb$g <- log1p(dslevin2016zeb$g)
```

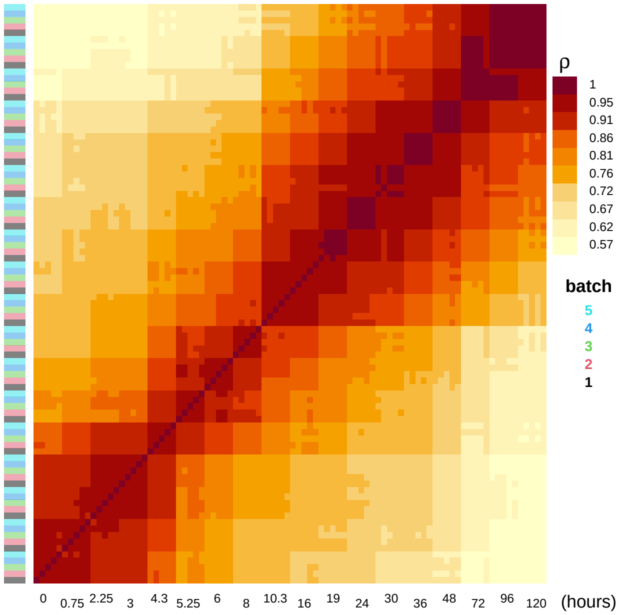
Check the contents of the expression matrix and pheno data.

```
dswhite2017$g[1:5, 1:4]
#>                    zmp_ph133_B zmp_ph133_D zmp_ph133_E zmp_ph133_F
#> ENSDARG00000000001    2.192007    2.019082    1.929426    2.031762
#> ENSDARG00000000002    1.149510    1.188959    0.900076    1.185358
#> ENSDARG00000000018    2.456661    2.534134    2.224970    2.364784
#> ENSDARG00000000019    4.432509    4.529970    4.608232    4.533400
#> ENSDARG00000000068    4.406696    4.460862    4.267657    4.294028
```

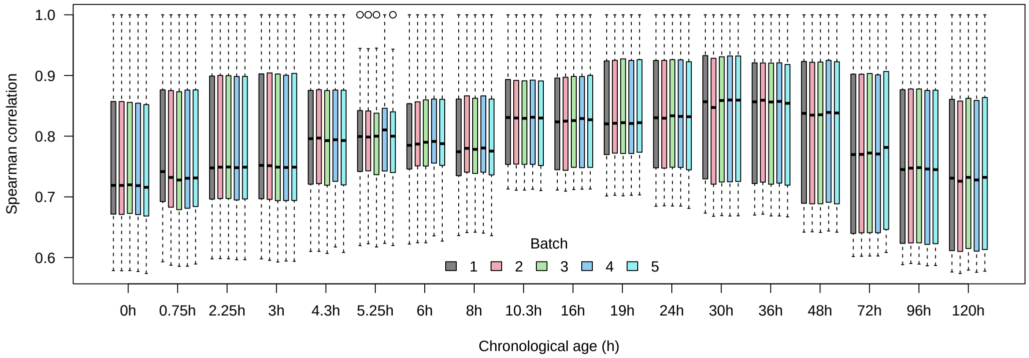
```
head(dswhite2017$p, n = 5)
#>        sample accession_number         stage stageName sampleName age
#> 1 zmp_ph133_B       ERS1079239 Zygote:1-cell    1-cell   1-cell-1   0
#> 2 zmp_ph133_D       ERS1079240 Zygote:1-cell    1-cell   1-cell-2   0
#> 3 zmp_ph133_E       ERS1079241 Zygote:1-cell    1-cell   1-cell-3   0
#> 4 zmp_ph133_F       ERS1079243 Zygote:1-cell    1-cell   1-cell-4   0
#> 5 zmp_ph133_G       ERS1079244 Zygote:1-cell    1-cell   1-cell-5   0
#>   batch
#> 1     1
#> 2     2
#> 3     3
#> 4     4
#> 5     5
```

Correlation between samples for outliers.

```
cor_dswhite2017 <- cor(dswhite2017$g, method = "spearman")
ord <- order(dswhite2017$p$age)
heatmap(cor_dswhite2017[ord, ord], Colv = NA, Rowv = NA,
        scale = "none", keep.dendro = F, margins = c(1,5),
        RowSideColors = transp(as.numeric(dswhite2017$p$batch[ord])),
        labRow = "", labCol = "")
par(xpd = T)
mtext(text = unique(dswhite2017$p$age), side = 1, line = c(3.8, 4),
      at = seq(-.12, .915, l = length(unique(dswhite2017$p$age))), cex = .6)

# color key
l.values <- seq(min(cor_dswhite2017), max(cor_dswhite2017), l = 10)
image(x = c(.95,1), y = seq(0.6,1, l = 10), useRaster = T,
      z = matrix(l.values, ncol = 10),
      col = hcl.colors(12, "YlOrRd", rev = TRUE), add = T)

text(.975, 1, pos = 3, labels = expression(rho), font = 2)
text(1, y = seq(0.6,1, l = 10), pos = 4,
     labels = round(l.values, 2), cex = .6)

xlp <- 1.025
batch_legend <- as.character(1:5)
text(xlp, .5, labels = "batch", font = 2, cex = .8, adj = .5)
text(xlp, seq(.3,.48, l = 5), labels = batch_legend, adj = 1, pos = 1,
     col = levels(dswhite2017$p$batch), font = 2, cex = .7)

mtext(at = xlp, line = 4, side = 1, text = "(hours)", cex = .8)
```



```
boxplot(cor_dswhite2017~interaction(dswhite2017$p$batch,
                                    dswhite2017$p$age),
        col = transp(1:5), # see 'Code to generate objects' for transp()
        xaxt = "n", at = seq(1,90, l = 90*(6/5))[c(T,T,T,T,T,F)], boxwex=.5,
        ylab = "Spearman correlation", xlab = "Chronological age (h)", las=1)
#add time labels
axis(side = 1, at = seq(2.5,87.5, l = 90/5),
     labels = paste0(unique(dswhite2017$p$age), 'h'))
legend('bottom', fill = transp(1:5), title = "Batch",
       legend = c(1:5), horiz = T, bty = "n")
```

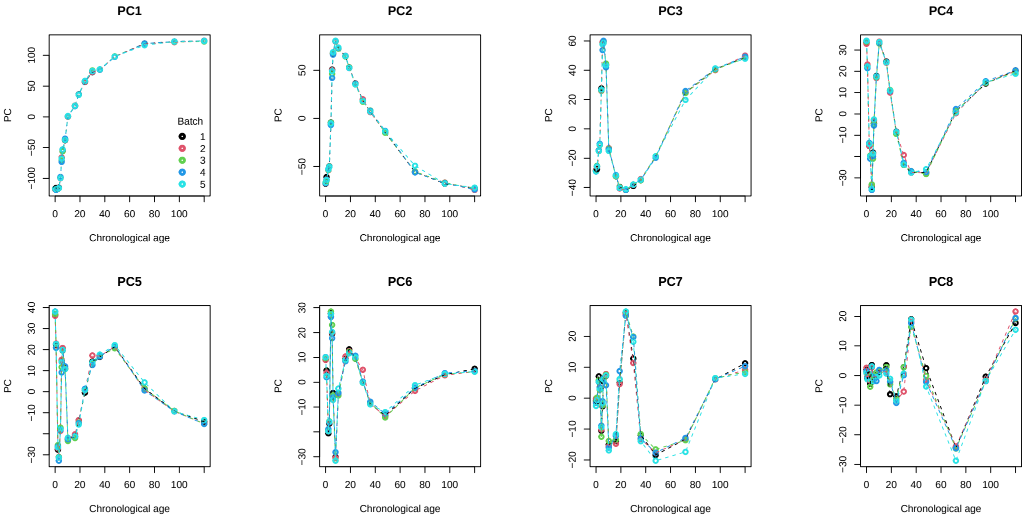


No outliers.

Plotting principal components.

```
pca_dswhite2017 <- stats::prcomp(t(dswhite2017$g), rank = 25,
                                 center = TRUE, scale = FALSE)
```

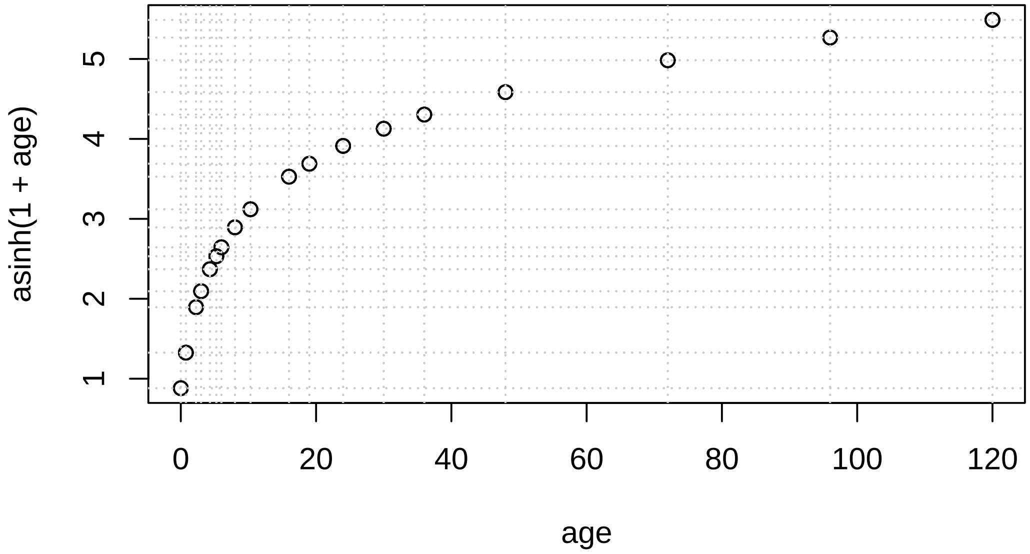
```
par(mfrow = c(2,4), pty='s')
invisible(sapply(1:8, function(i){
  plot(dswhite2017$p$age, pca_dswhite2017$x[,i],
       lwd = 2, col = dswhite2017$p$batch,
       xlab = "Chronological age", ylab = "PC", main = paste0("PC", i))
  # add dotted lines
  sapply(seq_along(levels(dswhite2017$p$batch)), function(l){
    s <- which(dswhite2017$p$batch == levels(dswhite2017$p$batch)[l])
    points(dswhite2017$p$age[s], pca_dswhite2017$x[s,i], col = l,
           type = 'l', lty = 2)
  })
  # legend
  if(i == 1)
    legend("bottomright", bty = 'n', legend = batch_legend, title = "Batch",
           pch = c(rep(1, 5)), lty = c(rep(NA, 5)), col = c(1:5), lwd = 3)
}))
```



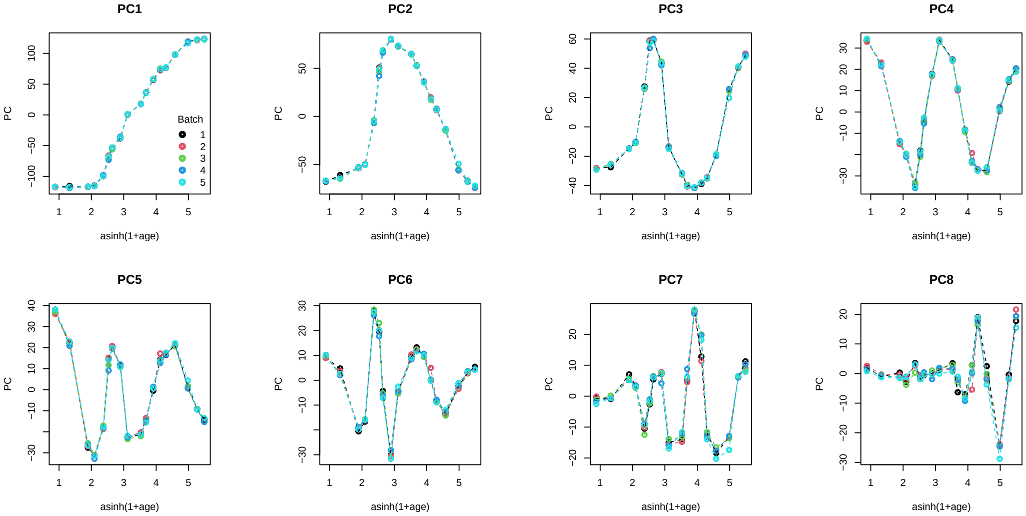
Sampling is sparser towards the end of the time series, and expression dynamics are also "wider". Because of this, fitting splines along age on these components poorly fit the earlier time points.

To bypass this, we can use `asinh(age)`, which has a similar effect to a logarithm to stretch the age values closer to a uniform scale. We also add an intercept to avoid overstretching the first few time points. The relationship between `age` and `asinh(1+age)` is shown below.

```
plot(asinh(1+age)~age, data=dswhite2017$p)
# add a grid
abline(v=dswhite2017$p$age, h=asinh(1+dswhite2017$p$age),
       col = 'grey80', lty=3)
```



We see that values are more evenly spread along the y axis, which is the intended effect. This effect also applies on the component dynamics, which will have better spline fits.



The transformation will be specified in the model formula so that we can predict new points from time values on the `age` scale (otherwise, we would need to predict with using transformed values such that they correspond to a uniform time scale when transformed back).

4.3.3 Model fitting

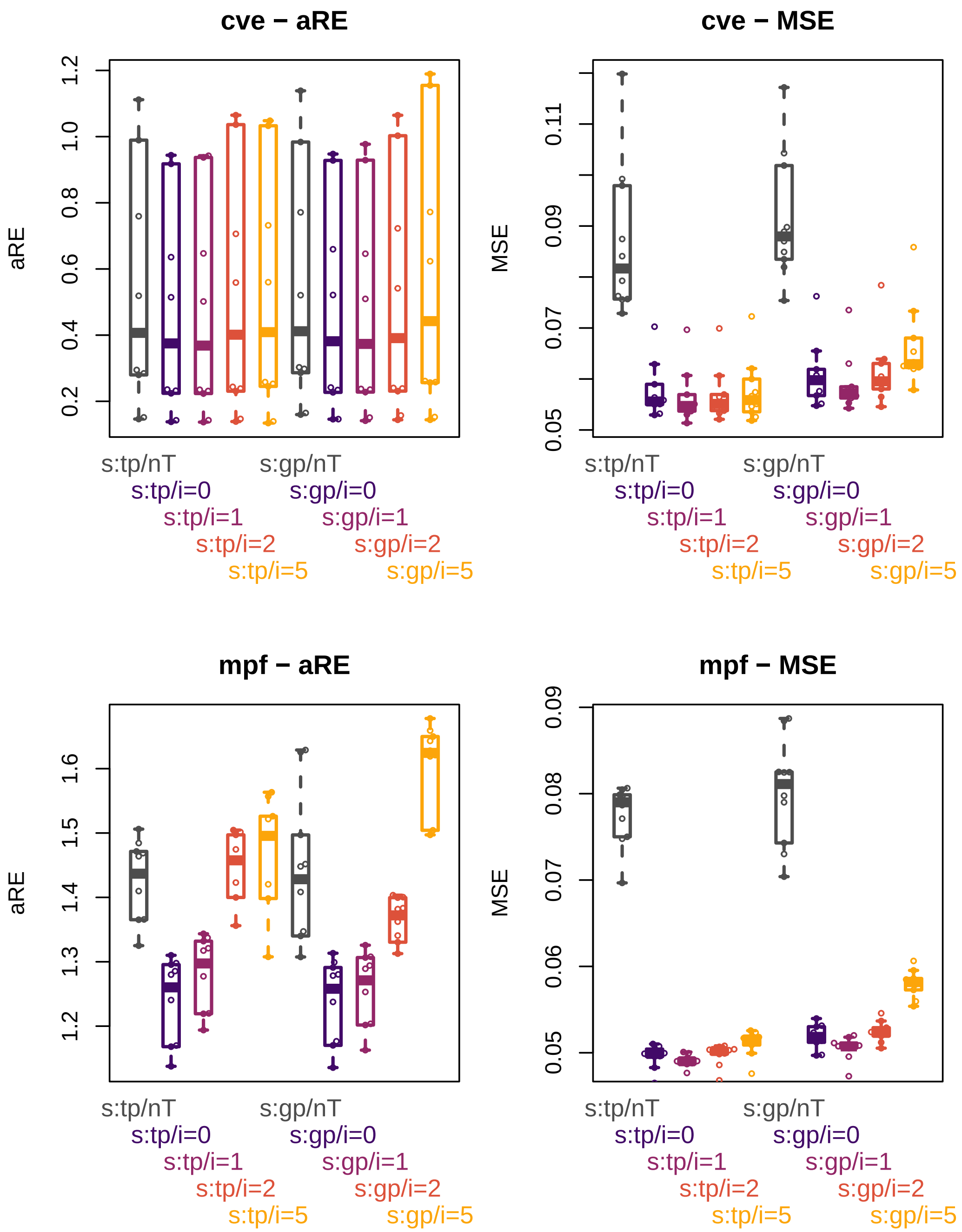
We keep enough components to explain 99% of the variance in the data.

```
nc <- sum(summary(pca_dswhite2017)$importance[3,] < .99) + 1
nc
#> [1] 67
```

We will fit a GAM on PCA components. The type of spline to fit as well as the value of the intercept in `asinh()` will be determined using cross-validation. Since sample batch is clearly absent from the components, we exclude it for a more parsimonious model.

```
set.seed(3)
# intercept values to try
intercepts <- c(0,1,2,5)

# list of formulas to test
flist <- as.list(paste0(
  "X ~ s(",
  c("age", # age without transformation
    paste0("asinh(", intercepts, "+age)")), # asinh with intercept
  ", bs = '",
  rep(c("tp", "gp"), e=1+length(intercepts)), # 2 different splines
  "', k=9)"))
# print formula list
cat(unlist(flist), sep = '\n')
#> X ~ s(age, bs = 'tp', k=9)
#> X ~ s(asinh(0+age), bs = 'tp', k=9)
#> X ~ s(asinh(1+age), bs = 'tp', k=9)
#> X ~ s(asinh(2+age), bs = 'tp', k=9)
#> X ~ s(asinh(5+age), bs = 'tp', k=9)
#> X ~ s(age, bs = 'gp', k=9)
#> X ~ s(asinh(0+age), bs = 'gp', k=9)
#> X ~ s(asinh(1+age), bs = 'gp', k=9)
#> X ~ s(asinh(2+age), bs = 'gp', k=9)
#> X ~ s(asinh(5+age), bs = 'gp', k=9)

cv_dswhite2017 <- ge_imCV(X = dswhite2017$g, p = dswhite2017$p,
                          formula_list = flist, nc = nc,
                          cv.n = 10, nb.cores = 6)
#> CV on 10 models. cv.n = 10 | cv.s = 0.8
#>
#> ...Building training sets
#> ...Setting up cluster
#> ...Running CV
#> ...Cleanup and formatting
```

```
par(mar = c(7,4,3,1))
cpal <- c("grey30", "#420A68FF",
          "#932667FF", "#DD513AFF", "#FCA50AFF") # color palette
plot(cv_dswhite2017, to_plot = c("aRE", "MSE"),
     names = paste0("s:", rep(c("tp", "gp"), e=1+length(intercepts)),
                    "/", c("nT", paste0("i=", intercepts))),
     col = NA, lwd=2, names.arrange = 5, boxwex = .5, outline=F,
     border = cpal, tcol = cpal, swarmargs = list(cex = .5, col = cpal))
```



We first see that with no `asinh()` transformation on time (grey, `nT` boxes), model performance and CV error are worse than on transformed data. Next, we note that than thin plate splines (`tp`) perform slightly better than Gaussian process splines (`gp`) from MSE. Finally, it seems that an intercept of 1 minimizes the overall CV error and model MSE.

We select `s:tp/i=1`, a thin plate spline with transformed age and intercept of 1.

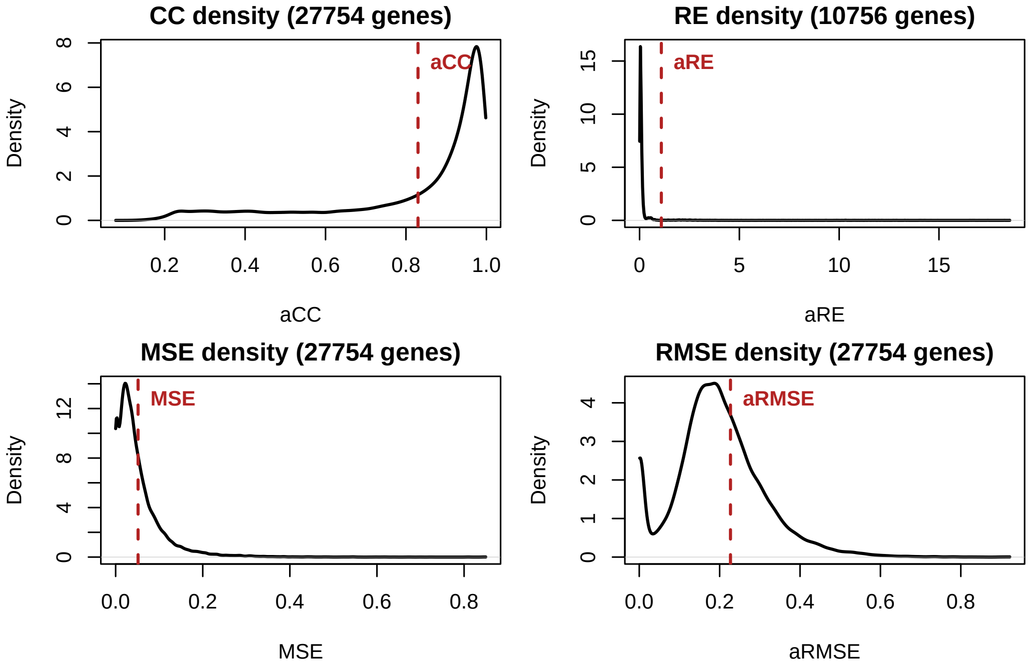
```
m_dswhite2017 <- ge_im(X = dswhite2017$g,
                       p = dswhite2017$p,
                       formula = "X ~ s(asinh(1+age), bs = 'gp', k=9)",
                         nc = nc)
```

```
#>             aCC      aRE       MSE     aRMSE
#> [1,] 0.8299206 1.090387 0.0515087 0.2269553
```

And then per gene.

```
ng_mp_white <-  mperf(dswhite2017$g, predict(m_dswhite2017),
                     is.t = T, global = F)
# remove NAs (eg. 0 variance genes) and Inf values (/0)
ng_mp_white <- lapply(ng_mp_white, na.omit)
ng_mp_white$aRE <- ng_mp_white$aRE[ng_mp_white$aRE < Inf]
```

```
par(mfrow = c(2,2), mar=c(4,4,2,1))
invisible(sapply(names(ng_mp_white), function(idx){
  rg <- range(na.omit(ng_mp_white[[idx]]))
  # estimate density curve
  d <- density(na.omit(ng_mp_white[[idx]]), from = rg[1], to = rg[2])
  plot(d, main = paste0(gsub("a", "", idx, fixed = T),
                        " density (", length(ng_mp_white[[idx]]), " genes)"),
       xlab = idx, lwd = 2)
  # add global value
  abline(v = mp_white[[idx]], lty = 2, lwd = 2, col = "firebrick")
  text(mp_white[[idx]], .9*max(d$y), pos = 4, labels = idx,
       font = 2, col = "firebrick")
}))
```



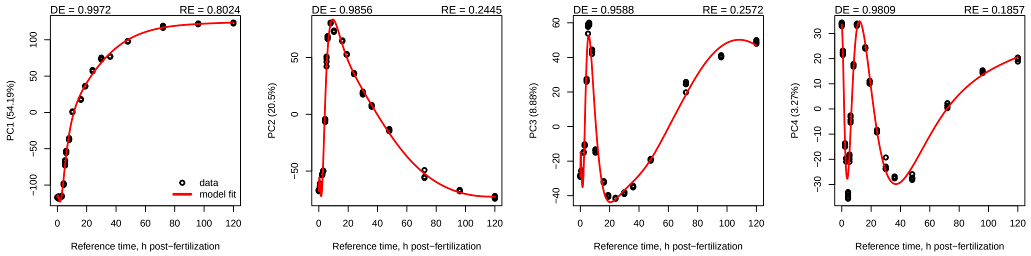
Very good fits.

Then, build the `ref` object.

```
# make a 'ref' object'
r_dswhite2017 <- make_ref(
  m_dswhite2017,
  n.inter = 500,
  t.unit = "h post-fertilization",     # time unit
  metadata = list("organism" = "D. rerio", # any metadata
                  "profiling" = "whole-organism, single-embryo",
                  "technology" = "RNAseq",
                  "note" = "asinh(1+age) interpolation"))
```

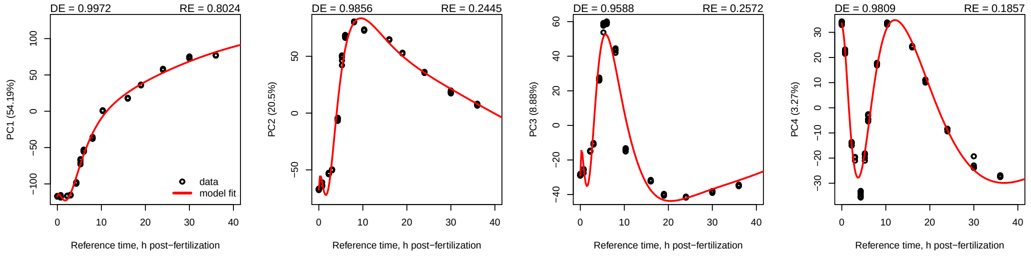
Plot component interpolation.

```
par(mfrow = c(1,4), pty='s')
plot(m_dswhite2017, r_dswhite2017, ncs=1:4, l.pos = 'bottomright')
```



We can zoom in on the early time points to ensure they are indeed properly fit.

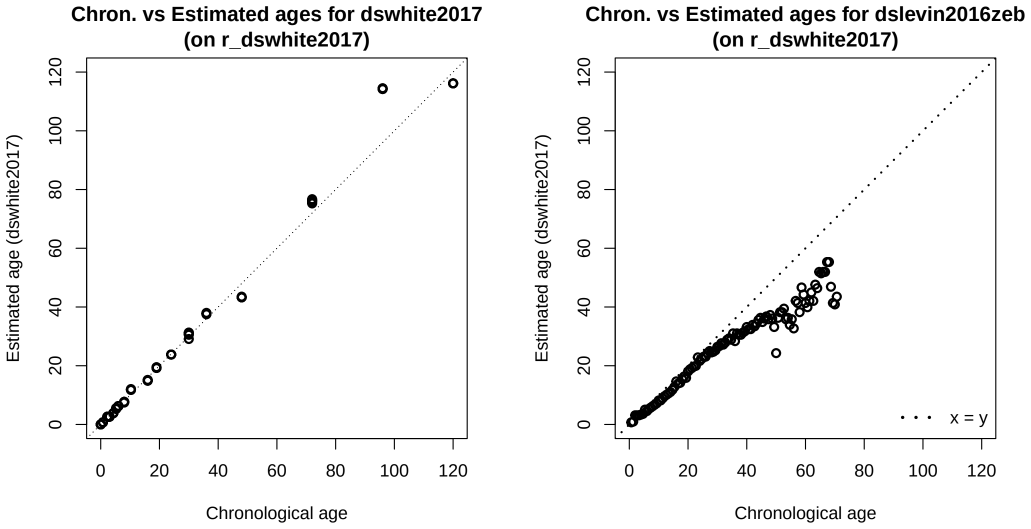
```
par(mfrow = c(1,4), pty='s')
plot(m_dswhite2017, r_dswhite2017, ncs=1:4, l.pos = 'bottomright', xlim = c(0,40))
```



Looks OK.

```
# stage samples
ae_dswhite2017 <- ae(dswhite2017$g, r_dswhite2017)
ae_dslevin2016zeb <- ae(dslevin2016zeb$g, r_dswhite2017)
#>                    nb.genes
#> refdata               27754
#> samp                  21429
#> intersect.genes       21429
```

```
par(mfrow = c(1,2), mar = c(4,5,3,1), pty='s')
rg <- range(c(ae_dswhite2017$age.estimates[,1], dswhite2017$p$age))
plot(ae_dswhite2017$age.estimates[,1]~dswhite2017$p$age,
     xlab = "Chronological age", ylab = "Estimated age (dswhite2017)",
     xlim = rg, ylim = rg,
     main = "Chron. vs Estimated ages for dswhite2017
(on r_dswhite2017)",
     lwd = 2)
abline(a = 0, b = 1, lty = 3, lwd = 1)

plot(ae_dslevin2016zeb$age.estimates[,1]~dslevin2016zeb$p$age,
     xlab = "Chronological age", ylab = "Estimated age (dswhite2017)",
     xlim = rg, ylim = rg,
     main = "Chron. vs Estimated ages for dslevin2016zeb
(on r_dswhite2017)", lwd = 2)
abline(a = 0, b = 1, lty = 3, lwd = 2)

legend("bottomright", legend = "x = y", lwd=3, col=1, lty = 3, bty='n')
```



4.3.4 Model fitting without age transformation

We'll build the same model without considering uneven sampling, for comparison. This is purposely not the optimal model, and will show interpolation issues to look out for.

```
m_dswhite2017_nT <- ge_im(X = dswhite2017$g,
                          p = dswhite2017$p,
                          formula = "X ~ s(age, bs = 'tp', k=9)",
                          nc = nc)
```

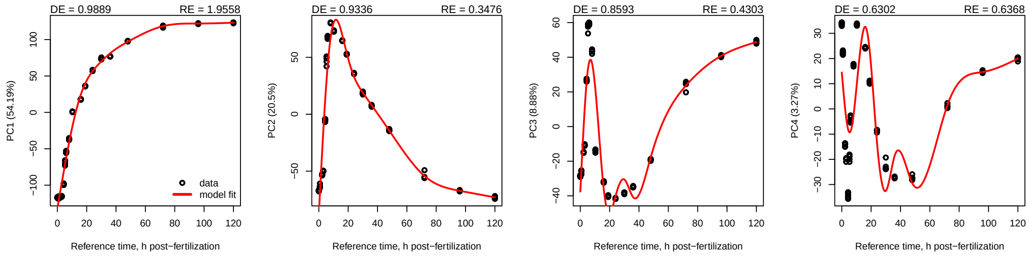
```
#>            aCC       aRE      MSE    aRMSE
#> [1,] 0.7776173 0.5979607 2.672777 1.634863
```

Build a ref object

```
# make a 'ref' object'
r_dswhite2017_nT <- make_ref(
  m_dswhite2017_nT,
  n.inter = 500,
  t.unit = "h post-fertilization",     # time unit
  metadata = list("organism" = "D. rerio", # any metadata
                  "profiling" = "whole-organism, single-embryo",
                  "technology" = "RNAseq",
                  "note" = "Not the best model"))
```

Plot component interpolation.

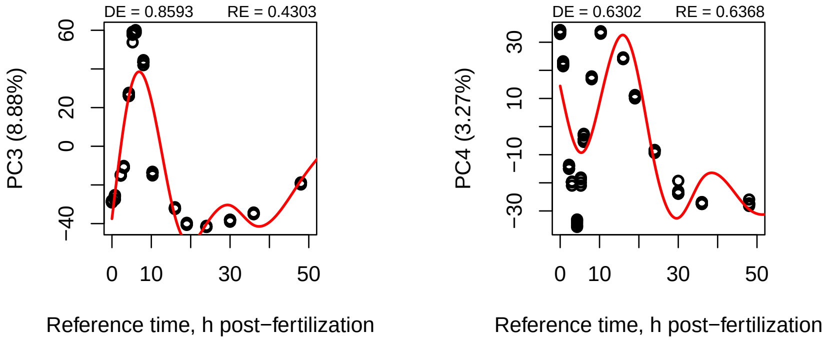
```
par(mfrow = c(1,4), pty='s')
plot(m_dswhite2017_nT, r_dswhite2017_nT, ncs=1:4, l.pos = 'bottomright')
```



We can see that the model has trouble predicting the dynamics accurately.

PC3 and PC4 have flattened early dynamics, and overfitting around 40hpf.

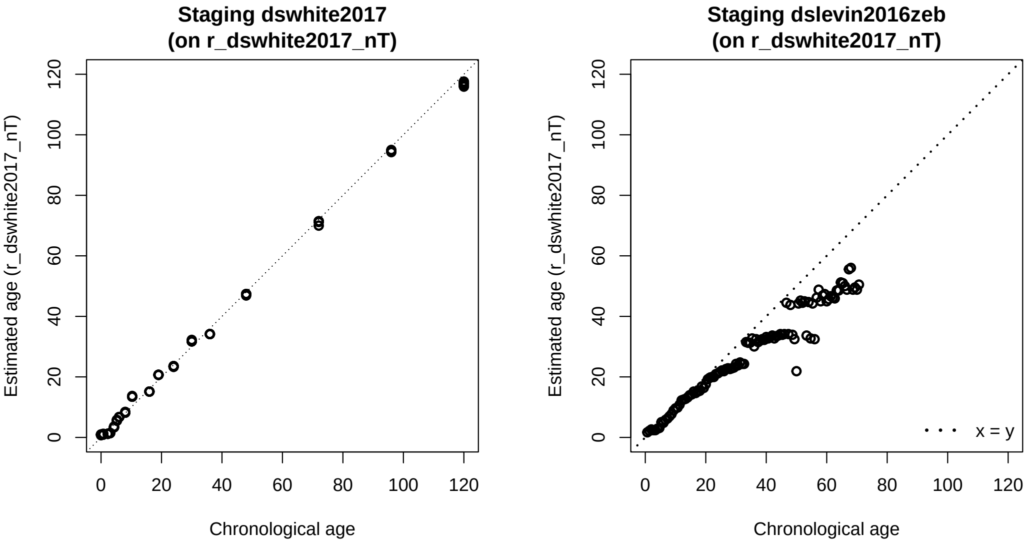
```
par(mfrow = c(1,2), pty='s')
plot(m_dswhite2017_nT, r_dswhite2017_nT, ncs=3:4, show.legend = F,
     xlim = c(0,50))
```



Consequences can be seen on when staging external data (*i.e.* the validation data), *but not necessarily when staging the reference data* as we'll see below. Validating references with independent/external data is thus highly recommended.

```
ae_dswhite2017_nT <- ae(dswhite2017$g, r_dswhite2017_nT)
ae_dslevin2016zeb_nT <- ae(dslevin2016zeb$g, r_dswhite2017_nT)
#>                   nb.genes
#> refdata              27754
#> samp                 21429
#> intersect.genes      21429
```

```
par(mfrow = c(1,2), mar = c(4,5,3,1), pty='s')
rg <- range(c(ae_dswhite2017_nT$age.estimates[,1], dswhite2017$p$age))
plot(ae_dswhite2017_nT$age.estimates[,1]~dswhite2017$p$age,
     xlab = "Chronological age", ylab = "Estimated age (r_dswhite2017_nT)",
     xlim = rg, ylim = rg,
     main = "Staging dswhite2017
(on r_dswhite2017_nT)",
     lwd = 2)
abline(a = 0, b = 1, lty = 3, lwd = 1)

plot(ae_dslevin2016zeb_nT$age.estimates[,1]~dslevin2016zeb$p$age,
     xlab = "Chronological age", ylab = "Estimated age (r_dswhite2017_nT)",
     xlim = rg, ylim = rg,
     main = "Staging dslevin2016zeb
(on r_dswhite2017_nT)", lwd = 2)
abline(a = 0, b = 1, lty = 3, lwd = 2)

legend("bottomright", legend = "x = y", lwd=3, col=1, lty = 3, bty='n')
```



The “gaps” or “steps” in the validation data estimates are caused by interpolation bias (overfitting and poorly fit dynamics mentioned above). These model errors create local “unlikely/unrealistic” gene expression areas in the interpolation, which do not correlate with the samples of corresponding age.

Gaps will most often be in-between original time points of the reference data, meaning the effect can’t be seen by staging reference samples. Validation data of sufficient temporal resolution will however have clear ‘blank’ ranges of age estimates, around which are clustered the samples of corresponding age.

The sub-optimal model we used had clear red flags on component plots, but interpolation bias can be much more subtle. In such cases, only staging external data will reveal the problem.

4.3.5 Code to generate objects

Required packages and variables:

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

requireNamespace("utils", quietly = T)
requireNamespace("GEOquery", quietly = T) # bioconductor
requireNamespace("Biobase", quietly = T)  # bioconductor
```

Note : set `data_folder` 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)))
}
```

```
requireNamespace("biomaRt", quietly = TRUE)

mart <- biomaRt::useMart("ensembl", dataset = "drerio_gene_ensembl")
zeb_genes <- biomaRt::getBM(attributes = c("ensembl_gene_id", "transcript_length"),
                            mart = mart)
rm(mart)
```

To build `dswhite2017`, *D. rerio* embryo time series from White et al. (2017)

```
p_url_dswhite2017 <- paste0("http://europepmc.org/articles/PMC5690287/",
                            "bin/elife-30860-supp1.tsv")
g_url_dswhite2017 <- apste0("http://europepmc.org/articles/PMC5690287/",
                            "bin/elife-30860-supp2.tsv")
g_file_dswhite2017 <- paste0(data_folder, "dswhite2017.tsv")
utils::download.file(g_url_dswhite2017, destfile = g_file_dswhite2017)

X_dswhite2017 <- read.table(g_file_dswhite2017, h = T, sep  ="\t",
                            as.is = T, quote = "\"")
rownames(X_dswhite2017) <- X_dswhite2017$Gene.ID
X_dswhite2017 <- X_dswhite2017[,-(1:8)]

# convert to tpm & ensembl_id
X_dswhite2017 <- X_dswhite2017[
  rownames(X_dswhite2017)%in%zeb_genes$ensembl_gene_id,]
X_dswhite2017 <- raw2tpm(
  rawcounts = X_dswhite2017,
  genelengths = zeb_genes$transcript_length[
    match(rownames(X_dswhite2017), zeb_genes$ensembl_gene_id)])

# pheno data
P_dswhite2017 <- read.table(p_url_dswhite2017, h = T, sep = "\t", as.is = T)
P_dswhite2017 <- P_dswhite2017[P_dswhite2017$sequencing == "RNASeq",
                               c("sample", "accession_number", "stage",
                                 "stageName", "sampleName")]

# timings of stages (from White et al. eLife (2017)).
# in hours post-fertilization
timepoints <- data.frame(stage = unique(P_dswhite2017$stageName),
                         hours_pf = c(0, .75, 2.25, 3, 4.3, 5.25, 6, 8, 10.3,
                                      16, 19, 24, 30, 36, 48, 72, 96, 120),
                         stringsAsFactors = F, row.names = "stage")
P_dswhite2017$age <- timepoints[P_dswhite2017$stageName, "hours_pf"]

# formatting
P_dswhite2017$batch <- factor(gsub(".*-(\\d)$", "\\1",
                                   P_dswhite2017$sampleName))
X_dswhite2017 <- X_dswhite2017[, P_dswhite2017$sample]

# save data
dswhite2017 <- list(g = X_dswhite2017, p = P_dswhite2017)
save(dswhite2017, file = paste0(data_folder, "dswhite2017.RData"),
     compress = "xz")

# cleanup
file.remove(g_file_dswhite2017)
rm(p_url_dswhite2017, g_url_dswhite2017, g_file_dswhite2017,
   X_dswhite2017, P_dswhite2017, timepoints)
```

To build dslevin2016zeb, *D. rerio* embryo time series from Levin et al. (2016)

```
geo_dslevin2016zeb <- "GSE60619"

g_url_dslevin2016zeb <- GEOquery::getGEOSuppFiles(geo_dslevin2016zeb,
                                                  makeDirectory = FALSE,
                                                  fetch_files = FALSE)
g_file_dslevin2016zeb <- paste0(data_folder, "dslevin2016zeb.txt.gz")
utils::download.file(url = as.character(g_url_dslevin2016zeb$url[2]),
                     destfile = g_file_dslevin2016zeb)

X_dslevin2016zeb <- read.table(gzfile(g_file_dslevin2016zeb), h = T,
                               sep = '\t', as.is = T, row.names = 1,
                               comment.char = "")

# convert to tpm & ensembl_id
X_dslevin2016zeb <- X_dslevin2016zeb[
  rownames(X_dslevin2016zeb)%in%zeb_genes$ensembl_gene_id,]
X_dslevin2016zeb <- raw2tpm(
  rawcounts = X_dslevin2016zeb,
  genelengths = zeb_genes$transcript_length[
    match(rownames(X_dslevin2016zeb), zeb_genes$ensembl_gene_id)])


# pheno data
P_dslevin2016zeb <- Biobase::pData(GEOquery::getGEO(geo_dslevin2016zeb,
                                                    getGPL = F)[[1]])

# filter relevant fields/samples
P_dslevin2016zeb <- P_dslevin2016zeb[, c("title", "geo_accession",
                                         "time (min after fertilization):ch1")]
colnames(P_dslevin2016zeb)[3] <- "time"
P_dslevin2016zeb$title <- as.character(P_dslevin2016zeb$title)
```

```
P_dslevin2016zeb <- P_dslevin2016zeb[
  P_dslevin2016zeb$title %in% colnames(X_dslevin2016zeb),]
X_dslevin2016zeb <- X_dslevin2016zeb[, P_dslevin2016zeb$title]

# formatting
P_dslevin2016zeb$title <- gsub('Metazome_ZF_timecourse_', '',
                               P_dslevin2016zeb$title)
colnames(X_dslevin2016zeb) <- P_dslevin2016zeb$title

P_dslevin2016zeb$age <- as.numeric(P_dslevin2016zeb$time) / 60

# save data
dslevin2016zeb <- list(g = X_dslevin2016zeb, p = P_dslevin2016zeb)
save(dslevin2016zeb, file = paste0(data_folder, "dslevin2016zeb.RData"),
     compress = "xz")

# cleanup
file.remove(g_file_dslevin2016zeb)
rm(geo_dslevin2016zeb, g_url_dslevin2016zeb, g_file_dslevin2016zeb,
   X_dslevin2016zeb, P_dslevin2016zeb)
```

Function to make a color transparent (`transp()`)

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

Back to top

References

Aeschimann, Florian, Pooja Kumari, Hrishikesh Bartake, Dimos Gaidatzis, Lan Xu, Rafal Ciosk, and Helge Großhans. 2017. "LIN41 Post-Transcriptionally Silences mRNAs by Two Distinct and Position-Dependent Mechanisms." *Molecular Cell* 65 (3): 476–89.

Alter, Orly, Patrick O Brown, and David Botstein. 2000. "Singular Value Decomposition for Genome-Wide Expression Data Processing and Modeling." *Proceedings of the National Academy of Sciences* 97 (18): 10101–6.

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

Graveley, Brenton R, Angela N Brooks, Joseph W Carlson, Michael O Duff, Jane M Landolin, Li Yang, Carlo G Artieri, et al. 2011. "The Developmental Transcriptome of Drosophila Melanogaster." *Nature* 471 (7339): 473.

Hendriks, Gert-Jan, Dimos Gaidatzis, Florian Aeschimann, and Helge Großhans. 2014. "Extensive Oscillatory Gene Expression During c. Elegans Larval Development." *Molecular Cell* 53 (3): 380–92.

Levin, Michal, Leon Anavy, Alison G Cole, Eitan Winter, Natalia Mostov, Sally Khair, Naftalie Senderovich, et al. 2016. "The Mid-Developmental Transition and the Evolution of Animal Body Plans." *Nature* 531 (7596): 637.

White, Richard J, John E Collins, Ian M Sealy, Neha Wali, Christopher M Dooley, Zsofia Digby, Derek L Stemple, et al. 2017. "A High-Resolution mRNA Expression Time Course of Embryonic Development in Zebrafish." *Elife* 6: e30860.

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] stats     graphics  grDevices utils     datasets  methods   base
#> 
#> other attached packages:
#> [1] beeswarm_0.4.0     RColorBrewer_1.1-3 biomaRt_2.50.3     wormRef_0.5.0
#> [5] BiocStyle_2.22.0   limma_3.50.3       RAPToR_1.2.0
#> 
#> loaded via a namespace (and not attached):
#>  [1] Biobase_2.54.0         httr_1.4.4             sass_0.4.5
#>  [4] bit64_4.0.5            jsonlite_1.8.4         viridisLite_0.4.1
#>  [7] splines_4.1.2          bslib_0.4.2            Rdpack_2.4
#> [10] assertthat_0.2.1       highr_0.10             BiocManager_1.30.19
#> [13] stats4_4.1.2           BiocFileCache_2.2.1    blob_1.2.3
#> [16] GenomeInfoDbData_1.2.7 yaml_2.3.7             progress_1.2.2
#> [19] pillar_1.8.1           RSQLite_2.2.20         lattice_0.20-45
#> [22] glue_1.6.2             digest_0.6.31          pryr_0.1.6
#> [25] XVector_0.34.0         rbibutils_2.2.13       htmltools_0.5.4
#> [28] Matrix_1.5-3           XML_3.99-0.13          pkgconfig_2.0.3
#> [31] magick_2.7.3           bookdown_0.32          zlibbioc_1.40.0
#> [34] tibble_3.1.8           KEGGREST_1.34.0        mgcv_1.8-41
#> [37] generics_0.1.3         IRanges_2.28.0         ellipsis_0.3.2
#> [40] cachem_1.0.6           BiocGenerics_0.40.0    cli_3.6.0
#> [43] magrittr_2.0.3         crayon_1.5.2           memoise_2.0.1
#> [46] evaluate_0.20          fansi_1.0.4            nlme_3.1-161
#> [49] xml2_1.3.3             ica_1.0-3              tools_4.1.2
#> [52] data.table_1.14.6      prettyunits_1.1.1      hms_1.1.2
#> [55] lifecycle_1.0.3       stringr_1.5.0         S4Vectors_0.32.4
#> [58] AnnotationDbi_1.56.2  Biostrings_2.62.0     compiler_4.1.2
#> [61] jquerylib_0.1.4       GenomeInfoDb_1.30.1   rlang_1.0.6
#> [64] grid_4.1.2            RCurl_1.98-1.10       rstudioapi_0.14
#> [67] rappdirs_0.3.3        bitops_1.0-7          rmarkdown_2.20.1
#> [70] codetools_0.2-18      curl_5.0.0            DBI_1.1.3
#> [73] R6_2.5.1              knitr_1.42            dplyr_1.1.0
#> [76] utf8_1.2.3            fastmap_1.1.0         bit_4.0.5
#> [79] filelock_1.0.2        stringi_1.7.12        parallel_4.1.2
#> [82] Rcpp_1.0.10           vctrs_0.5.2           png_0.1-8
#> [85] tidyselect_1.2.0      dbplyr_2.3.0          xfun_0.37
```