RAPToR - Real Age Prediction from Transcriptome staging On Reference

RAPToR 1.2.0

Romain Bulteau

March 2023

Contents

RAPToR is a computational framework to estimate the real age of biological samples from gene expression. This is especially useful for fast-developing organisms – such as *C. elegans* worms, flies, or zebrafish – where many factors substantially impact developmental speed, and thus unintended developmental variation between samples could obscure or confound the effect of variables of interest.

With RAPToR and the inferred age of your samples, you can

- precisely estimate the effect of perturbations on developmental timing (including in a tissue-specific way),
- increase power in differential expression analyses,
- quantify differential expression due to uncontrolled development and,
- recover perturbation-specific effects on gene expression even when completely confounded by development.

Please cite our article (Bulteau and Francesconi (2022)) if you use RAPToR in your research:

- Bulteau, R., Francesconi, M. Real age prediction from the transcriptome with RAPToR. *Nat Methods* (2022). (https://doi.org/10.1038/s41592-022-01540-0)

1 Quickstart

Given an expression matrix X of transcripts per million (TPM) with samples as columns and genes as rows.

```
library(RAPToR)
library(wormRef) # reference data package

# quick look at expression matrix
X[1:5, 1:3]
#>                 contDevA_N2_21h contDevA_N2_22h contDevA_N2_23h
#> WBGene00007063       2.7298009        2.928176       2.3023107
#> WBGene00007064       5.2372118        6.161017       7.0817623
#> WBGene00007065      14.5928869       11.448793       9.7072795
#> WBGene00003525       0.2554172        1.182322       3.8705742
#> WBGene00007067       1.3860572        1.025319       0.6974292
```

Load an appropriate reference.

```
ref <- prepare_refdata("Cel_larv_YA", "wormRef", 600)
```

Stage samples.

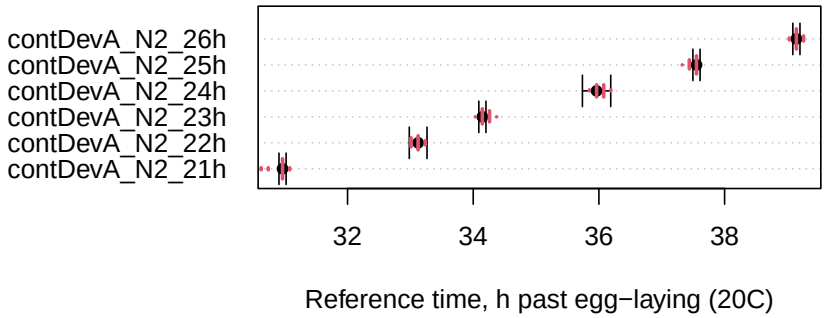
```
ae_X <- ae(X, ref)
#>                 nb.genes
#> refdata            19953
#> samp               19595
#> intersect.genes    19525
#> Bootstrap set size is 6508
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
```

Show output.

```
print(ae_X)
#> RAPToR ae object
#> ---
#> Call:    ae(samp = X, refdata = ref)
#> Reference metadata:
#>  organism: C. elegans
#>  profiling: whole-organism, bulk
#>  technology: RNAseq
#>
#> Time unit: h past egg-laying (20C)
#>
#>                 age.estimate     lb     ub cor.score
#> contDevA_N2_21h       30.966 30.909 31.022     0.982
#> contDevA_N2_22h       33.124 32.983 33.265     0.981
#> contDevA_N2_23h       34.146 34.089 34.202     0.982
#> contDevA_N2_24h       35.963 35.738 36.188     0.979
#> contDevA_N2_25h       37.553 37.496 37.610     0.980
#> contDevA_N2_26h       39.143 39.086 39.200     0.978
```

Plot age estimates.

```
plot(ae_X)
```



Extract age estimates.

```
ae_X$age.estimates[,1]
#> contDevA_N2_21h contDevA_N2_22h contDevA_N2_23h contDevA_N2_24h
#>        30.96566        33.12353        34.14568        35.96284
#> contDevA_N2_25h contDevA_N2_26h
#>        37.55286        39.14287
```

2 About RAPToR

2.1 Why use RAPToR ?

In gene expression data, unknown and unintended developmental variation among biological samples can obscure and confound the effect of variables of interest. As many factors influence growth speed (including experimental conditions), synchronizing samples can be challenging but failing to do has a strong impact on gene expression.

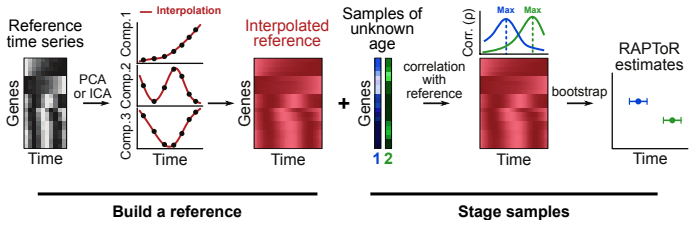
Aware of this problem, studies with large scale developmental profiling generally re-order or rank the samples post-profiling with methods such as BLIND (Anavy et al. (2014)) that combine dimension-reduction (PCA, Diffusion Map, ...) and a trajectory-finding method. Unfortunately, this only works with hundreds of samples and/or time-series designs and thus doesn't apply to the overwhelming majority of expression profiling studies.

Overcoming this limitation, RAPToR provides a way to precisely determine the real age of *single samples* from their expression profile.

2.2 How does it work ?

RAPToR is a 2-step process:

1. A reference gene expression time-series is interpolated to build a near-continuous, **high-temporal-resolution reference** (a number of which are included in associated data-packages, see below).
2. A **correlation profile** against this reference is dressed for of each of your samples, and the timing of the correlation peak is the estimated age. **Bootstrapping on genes** then gives a confidence interval of the estimates.



2.3 What type of data can be used ?

RAPToR can stage samples from any genome-wide gene expression profiling method : RNA-seq (preferably as TPM), microarray, ...

Note that the references provided in the data-packages are $log(X + 1)$ of expression values, so applying this transformation to your data is important if comparing expression changes with the reference (but is not required for staging).

Data must not be gene-centered, as this destroys the relationship between gene levels within a sample.

2.4 General structure of the package

The main package (RAPToR) holds the functions needed for staging samples and building references.

To provide a quick and easy way to infer the age of samples, we pre-built several references for commonly used organisms from available data in the literature. References are voluminous for an R package, so they are stored in separate "data-packages". For example, wormRef stores the *C. elegans* references.

References available in a data-package are listed with the list_refs() function (you must have the data-package installed for this, wormRef can be installed from here.)

```
list_refs("wormRef")
#>          name              organism                description
#> 1  Cel_embryo Caenorhabditis elegans       Embryonic development
#> 2  Cel_larval Caenorhabditis elegans          Larval development
#> 3 Cel_larv_YA Caenorhabditis elegans Larval to adult development
#> 4    Cel_YA_1 Caenorhabditis elegans              YA development
#> 5    Cel_YA_2 Caenorhabditis elegans              YA development
#>                                range
#> 1 1-cell(-50) to 840 min past 4-cell
#> 2      0 to 55 h post-hatching (20C)
#> 3      5 to 76 h post-hatching (20C)
#> 4   41.5 to 72 h post-hatching (20C)
#> 5     48 to 87 h post-hatching (20C)
```

The data-packages currently available are listed in the README of RAPToR's github repo. You can also build your own reference data-packages following guidelines given in the vignette on this topic.

3 Detailed usage example

In this part, we show how to stage two *C. elegans* time-series datasets using pre-built references from `wormRef` (which must be installed to use its references, see here for installation).

1. A larval development time-series of 4 different strains published by Aeschimann et al. (2017), hereafter called `dsaeschimann2017`.
2. A high-resolution time-series of late larval development published by Hendriks et al. (2014), hereafter called `dshendriks2014`

Both datasets are available on GEO (GSE80157, GSE52861), and code to create the `dsaeschimann2017` and `dshendriks2014` objects can be found at the end of this vignette.

Here is a quick look at the data

```
dsaeschimann2017$g[1:5,1:3]
#>                let.7.n2853._18hr let.7.n2853._20hr let.7.n2853._22hr
#> WBGene00007063          7.501850         10.988212          10.45480
#> WBGene00007064          8.023767          8.655388          14.21012
#> WBGene00007065         15.919452         16.875057          15.23932
#> WBGene00003525          1.416181         10.938876          13.42202
#> WBGene00007067          1.765342          1.775650           2.77224

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
```

```
dshendriks2014$g[1:5,1:3]
#>                contDevA_N2_21h contDevA_N2_22h contDevA_N2_23h
#> WBGene00007063       2.7298009        2.928176       2.3023107
#> WBGene00007064       5.2372118        6.161017       7.0817623
#> WBGene00007065      14.5928869       11.448793       9.7072795
#> WBGene00003525       0.2554172        1.182322       3.8705742
#> WBGene00007067       1.3860572        1.025319       0.6974292

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
```

We quantile-normalize and log the expression data (this is not required for staging, but is useful for plotting the expression data).

```
library(limma)

dsaeschimann2017$g <- limma::normalizeBetweenArrays(dsaeschimann2017$g,
                                                    method = "quantile")
dsaeschimann2017$g <- log1p(dsaeschimann2017$g) # log1p(x) = log(x + 1)

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

You may need to convert probe IDs or gene IDs of your data to match those of the reference series. All the references included in the `wormRef` data-package use WormBase Gene IDs (*e.g.* `WBGene00016153`). To help with this conversion, we provide the `format_ids()` function and gene ID reference tables are included in the data-packages (built directly from `biomaRt` queries).

Transcript-level data is aggregated into gene-level expression in our references for a broader usage (since it is possible to compute gene-level expression from transcripts, but not the other way around). RNASeq expression data should ideally be sum-aggregated at the count level, and other types of data – such as microarray, or RNASeq TPM (if counts are unavailable) – are mean-aggregated.

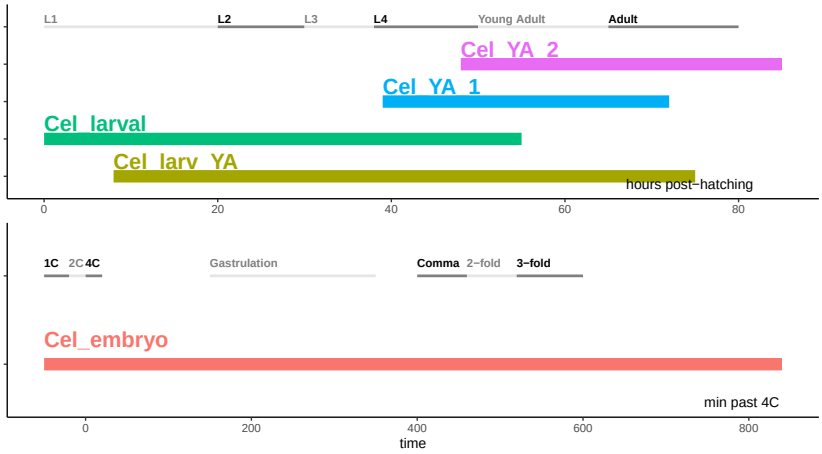
3.1 Choosing and loading a reference

`RAPToR` estimates sample age using correlation with a reference time-series, meaning you must select the proper reference to compare your samples with.

If we haven't built references for your favorite organism yet, you can take a look at the *Building your own references* section in this vignette for a quick-start guide or the dedicated reference-building vignette for an in-depth explanation. For species without available time-series data (*e.g.* non-model), samples can still be staged using ortholog genes and a reference from a related specie.

To determine if an existing data-package reference covers the appropriate development span for your samples, use `list_refs()` like above, or the `plot_refs()` function which shows the references available in the data-package (in this case, `wormRef`) along with landmark developmental stages.

```
plot_refs("wormRef")
```



The `Cel_larval` reference is appropriate for both example datasets since their samples range from mid-larval to early young adult. We load it using `prepare_refdata()`, specifying an interpolation resolution through either `n.inter` (number of interpolated points) or `by.inter` (interpolation step in reference time units). In the interest of lightening the computational load (each interpolated timepoint is compared to the samples) you can choose smaller `n.inter` or larger `by.inter` values. We find having `n.inter` > 500 gets the most out of interpolation.

```
r_larv <- prepare_refdata("Cel_larval", "wormRef", n.inter = 600)
```

Note that age estimates will be given in the time unit and scale of the chosen reference (here, hours post-hatching at 20°C).

```
r_larv
#> RAPToR reference object
#> ---
#>  interpGE:   ( 18718 x 600 )
#>  time:    [ 0 - 55.00455 ] h past egg-laying (20C), by 0.0918273 steps
#>
#>  Metadata:
#>  organism: C. elegans
#>  profiling: whole-organism, bulk
#>  technology: RNAseq
#>
#>  GEIM:
#>   GAM fit on 40 PCA components:
#>   X ~ s(age, bs = 'ds') + cov
#>   with covariate levels cov: 0.20
#> ---
```

3.2 Age estimation

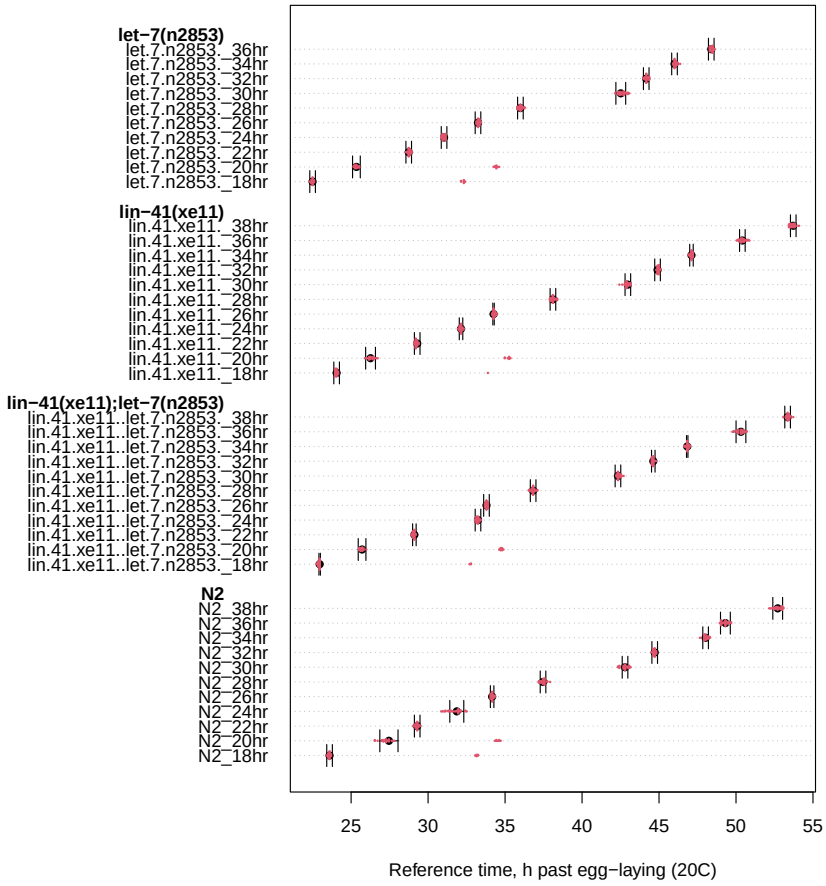
All that is left to do is run the `ae()` (**a**ge **e**stimation) function.

```
ae_dsaeschimann2017 <- ae(dsaeschimann2017$g, # input gene expression matrix
                          r_larv)             # reference object
#> Bootstrap set size is 6239
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
#>                 nb.genes
#> refdata            18718
#> samp               19595
#> intersect.genes    18718
```

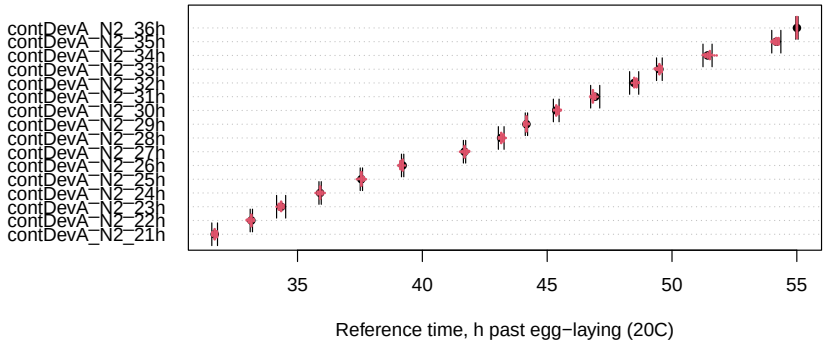
```
ae_dshendriks2014 <- ae(dshendriks2014$g, # input gene expression matrix
                        r_larv)           # reference object
#> Bootstrap set size is 6239
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
#>                 nb.genes
#> refdata            18718
#> samp               19595
#> intersect.genes    18718
```

Let's look at the results.

```
plot(ae_dsaeschimann2017, groups = dsaeschimann2017$p$strain,
     lmar = 14, g.line = 3)
```




```
plot(ae_dshendriks2014)
```



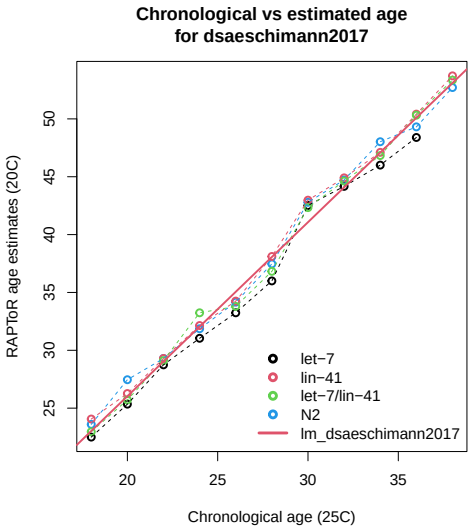
At 25°C, *C. elegans* worms develop 1.5 times faster than worms at 20°C. Since the worms we staged were grown at 25°C, but the reference – and thus, the inferred age – corresponds to 20°C development, we can see this 1.5 factor by fitting a simple linear model between chronological and estimated age.

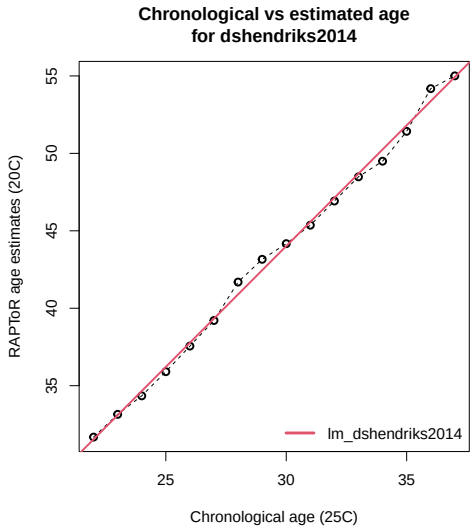
```
# extract age estimates
dsaeschimann2017$p$age_est <- ae_dsaeschimann2017$age.estimates[,1]
dshendriks2014$p$age_est <- ae_dshendriks2014$age.estimates[,1]

lm_dsaeschimann2017 <- lm(age_est ~ age, data = dsaeschimann2017$p)
summary(lm_dsaeschimann2017)$coefficients
```

```
#>               Estimate Std. Error  t value     Pr(>|t|)
#> (Intercept) -4.049955 0.66919171 -6.05201 3.648829e-07
#> age          1.504035 0.02351908 63.94959 1.068742e-42

lm_dshendriks2014 <-  lm(age_est ~ age, data = dshendriks2014$p)
summary(lm_dshendriks2014)$coefficients
#>               Estimate Std. Error   t value     Pr(>|t|)
#> (Intercept) -2.774130 0.74277707 -3.734808 2.219069e-03
#> age          1.559579 0.02487699 62.691619 1.489212e-18
```





3.3 Understanding the output

The output of `ae()` is an `ae` object containing (among other things) the age estimate and confidence intervals from booststrapping (age estimates on random gene subsets).

General information can be accessed via `summary()`.

```
summary(ae_dshendriks2014)
#>
#> Age estimate
#>   Time unit: h past egg-laying (20C)
#>   Span:      23.324
#>   Range: [ 31.68 , 55.005 ]
#> ---
#>                 age.estimate     lb     ub
#> contDevA_N2_21h       31.680 31.566 31.794
#> contDevA_N2_22h       33.150 33.104 33.196
#> contDevA_N2_23h       34.343 34.161 34.525
#> contDevA_N2_24h       35.904 35.859 35.950
#> contDevA_N2_25h       37.557 37.511 37.603
#> contDevA_N2_26h       39.210 39.164 39.256
#> contDevA_N2_27h       41.690 41.644 41.736
#> contDevA_N2_28h       43.159 43.045 43.273
#> contDevA_N2_29h       44.169 44.123 44.215
```

```
#> contDevA_N2_30h       45.363 45.249 45.477
#> contDevA_N2_31h       46.924 46.742 47.106
#> contDevA_N2_32h       48.485 48.303 48.667
#> contDevA_N2_33h       49.495 49.381 49.609
#> contDevA_N2_34h       51.423 51.241 51.605
#> contDevA_N2_35h       54.178 53.996 54.360
#> contDevA_N2_36h       55.005 54.959 55.050
```

Age estimates and their confidence intervals are accessible through `$age.estimates`, a table with

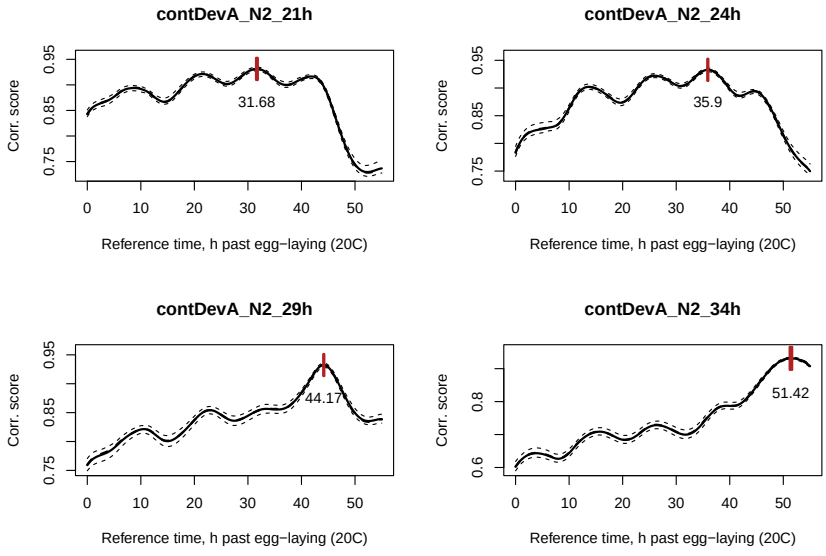
- `age.estimate`, age estimate for the sample.
- `lb`, `ub`, the lower and upper bounds of the bootstrap confidence interval (Median Absolute Deviation of bootstrapped estimated).
- `cor.score`, the correlation score between the sample and reference at the age estimate.

```
head(ae_dshendriks2014$age.estimates)
#>                 age.estimate       lb       ub cor.score
#> contDevA_N2_21h     31.68042 31.56643 31.79440 0.9312136
#> contDevA_N2_22h     33.14966 33.10374 33.19557 0.9310279
#> contDevA_N2_23h     34.34341 34.16135 34.52547 0.9346601
#> contDevA_N2_24h     35.90448 35.85856 35.95039 0.9328182
#> contDevA_N2_25h     37.55737 37.51145 37.60328 0.9330641
#> contDevA_N2_26h     39.21026 39.16434 39.25617 0.9359745
```

Estimates and confidence intervals can be displayed in the form of a 'dotchart' with the default `plot()` function (as done above).

The `ae` object also holds record of correlation scores of each samples with the full reference span (for bootstrap estimates as well). These correlation profiles can be plotted with `plot_cor()`

```
par(mfrow=c(2,2))
plot_cor(ae_dshendriks2014, subset = c(1,4,9,14))
```



Red bars correspond to the estimate confidence interval, and the 95% interval of bootstrap correlation with the reference is shown as black dotted lines. The sample age estimate is displayed below the interval.

4 Building your own references

This section gives a broad overview, on reference-building, but note there is a vignette entirely dedicated to it..

Interpolating on time-series expression data is the key to get the high-temporal-resolution references we need for RAPToR. When using our pre-built references, interpolation is done internally by the prepare_refdata() function, using the functions described below.

To build a reference, you will require a time-series of gene expression data for your favorite organism. We will use the dsaeschimann2017 data from the previous examples to illustrate the process.

4.1 The gene expression interpolation model

Gene expression interpolation models (GEIMs) are built with the ge_im() function. This function takes as input 3 key arguments :

- X : your time-series gene expression matrix (genes as rows, samples as columns)
- p : a dataframe of phenotypic data, samples as rows *in the same order as X columns*. This should include the age/time variable and any other covariate(s) 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, which may include elements (such as splines) from chosen model type (see below). **It must start with X**.

For example, using dsaeschimann2017, we build the following model:

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

In order to model a large number of output variable (genes), our strategy is to project the data in a dimensionally-reduced space and interpolate there before re-projecting the data back to genes. 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.

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. We use this property through GEIMs. 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 implemented 2 model types : Generalized Additive Models (GAMs, the default) and Generalized Linear Models (GLMs). GAMs use the `gam()` function of the `mgcv` package, and GLMs use the `glm()` function of the `stats` core package. The specified model formula can include the usual tools of `gam()` or `glm()`, most notably the variety of polynomial or smoothing splines implemented through the `s()` function for GAMs.

If you are unfamiliar with the `mgcv` package or GLMs in R, we recommend you look at their documentation (especially `s()` for GAMs) to understand the terms included in formulas to model non-linear dynamics.

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

4.2 Defining the appropriate model formula and parameters

4.2.1 Model type

Our default GEIM fits GAMs on PCA components, which is a robust choice when applying a smoothing spline to the data. PCA and ICA interpolation usually yield near-identical results (but ICA components tend to be more biologically meaningful or interpretable).

4.2.2 Parameter estimation

The number of components to use for the interpolation is by default set to the number of samples. However, we recommend setting a cutoff on explained variance of PCA components. For example, on the `dsaeschimann2017` dataset, we set a threshold at 99% :

```
pca_dsaeschimann2017 <- stats::prcomp(t(dsaeschimann2017$g),
                                      center = TRUE, scale = FALSE,
                                      rank = 25)
nc <- sum(summary(pca_dsaeschimann2017)$importance[3,] < .99) + 1
nc
#> [1] 32
```

This threshold should be set with respect to 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 or dynamics ? One can also keep only components that have '*intelligible dynamics*' with respect to time, defined as those where a model fit explains > 0.5 of the deviance (noisy components with no dynamics have poor fits).

Choosing from different splines (and/or parameters) can be done with Cross-Validation (CV) through the `ge_imCV()` function, which inputs `X`, `p` and a `formula_list` to test. Other parameters on the CV itself can also be given (*e.g.* training set size).

Below is an example to choose among available spline types for the `dsaeschimann2017` GEIM.

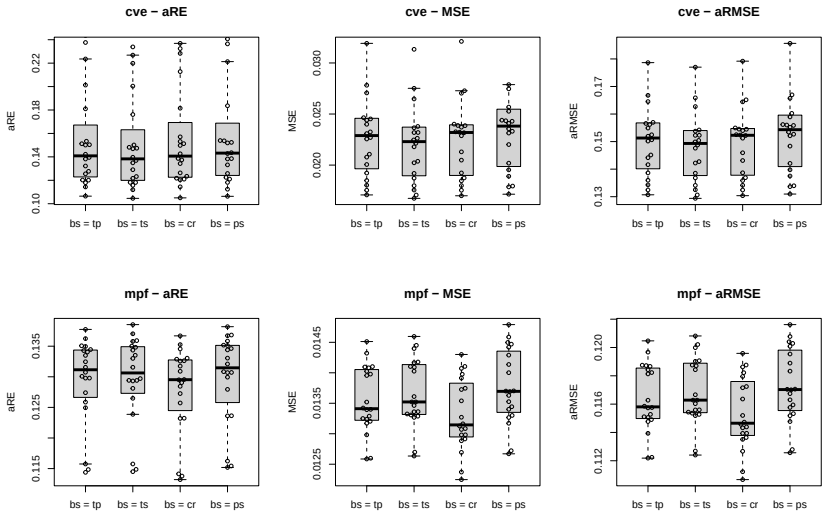
```
smooth_methods <- c("tp", "ts", "cr", "ps")
flist <- as.list(paste0("X ~ s(age, bs = \'", smooth_methods, "\') + strain"))

cat(unlist(flist), sep='\n') # print formulas to test
#> X ~ s(age, bs = 'tp') + strain
#> X ~ s(age, bs = 'ts') + strain
#> X ~ s(age, bs = 'cr') + strain
#> X ~ s(age, bs = 'ps') + strain
```

```
cv_dsaeschimann2017 <- ge_imCV(X = dsaeschimann2017$g,
                               p = dsaeschimann2017$p,
                               formula_list = flist,
                               cv.n = 20, nc = nc,
                               nb.cores = 3)
#> CV on 4 models. cv.n = 20 | cv.s = 0.8
#>
#> ...Building training sets
#> ...Setting up cluster
#> ...Running CV
#> ...Cleanup and formatting
```

Plot the results.

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



`ge_imCV()` computes multiple indices of model performance : the average Correlation Coefficient (`aCC`), the average Relative Error (`aRE`), Mean Squared Error (`MSE`) and average Root MSE (`aRMSE`). These indices all compare model predictions with the true data, and they are computed both using the validation set (CV Error, cve) *and* on the training set (Model PerFormance, mpf).

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

4.3 Building a reference object from a model

A `ref` object is built from a GEIM using `make_ref()`, specifying interpolation resolution and relevant metadata:

```
r_dsaeschimann2017 <- make_ref(
  m = m_dsaeschimann2017,
  n.inter = 100,                  # interpolation resolution
  t.unit = "h past egg-laying",   # 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")
  )
```

4.4 Validating the interpolation

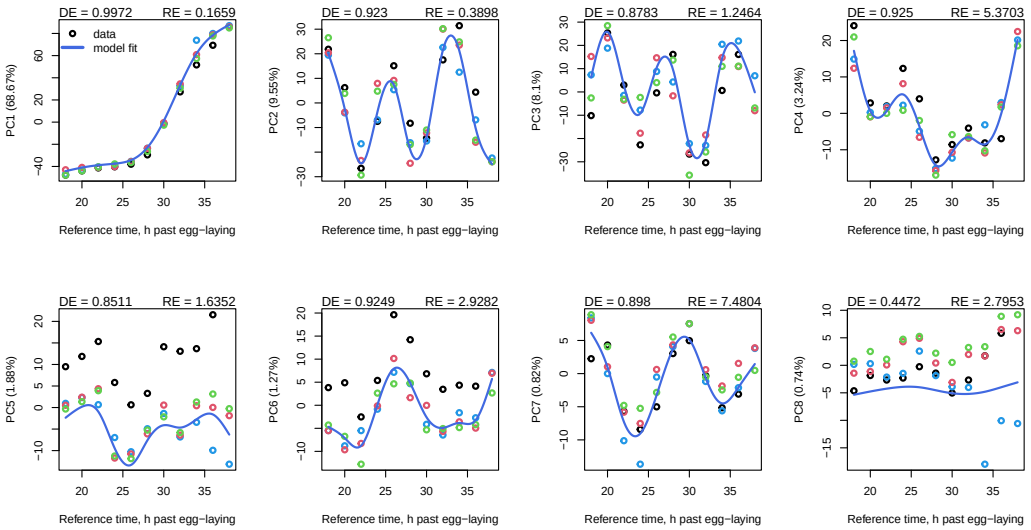
After building a reference, we check interpolation results by:

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

We can do both with our example data, using `dshendriks2014` for external validation.

Plotting the model and reference object (or equivalent metadata) shows component interpolation, with deviance explained (DE) and relative error (RE) for each component.

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



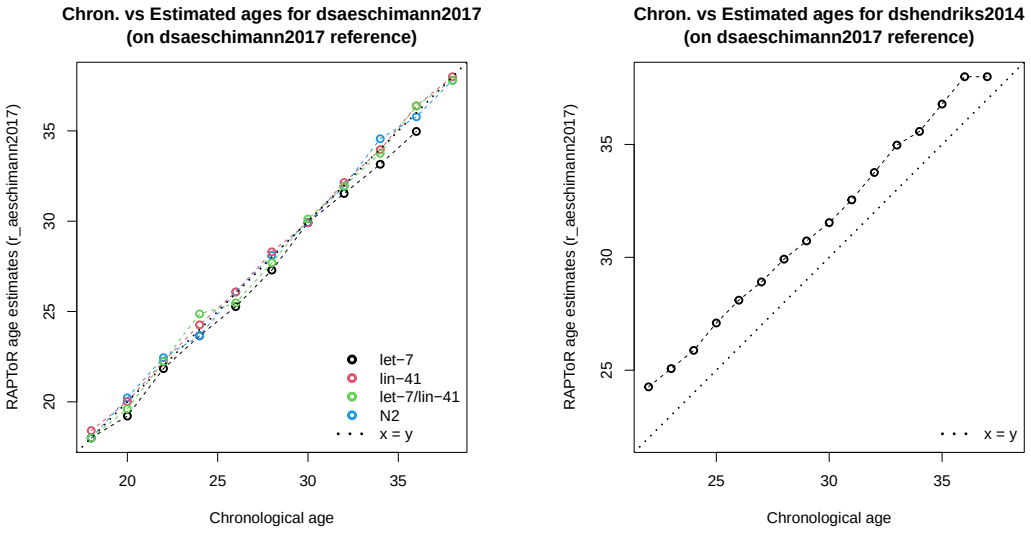
Of note, we are predicting model values as `N2` (lightblue). While all strains are shown on the plots, some model parameters depend on the selected `N2` strain.

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
```

Staging the samples and external validation dataset.

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



Staging results are excellent, validating the reference which can now be used to stage any samples within its developmental span.

5 Staging with a prior

When few genes are available for staging (e.g. for tissue-specific staging), it may be appropriate to give RAPToR a prior to help age estimation. In the `ae()` function, priors are parameters for gaussian distributions of time (for each sample). The correlation peaks are then ranked according to the prior's density. The correlation profile is unaffected by the prior, only the choice of the correlation peak is.

This implies that with a prior which is completely off, the estimate may also be wrong ; use with care.

Priors must be given *in the time scale of the reference*, so beware of growth speed difference with temperature or different time origins (fertilization, egg-laying, hatching...). For example, the `dshendriks2014` *C. elegans* data used above is grown at 25°C, resulting in a different growth speed than the reference which is at 20°C development.

We insist on careful use due to these possible differences and biases introduced by the prior.

A first run without priors will give a general idea of the difference between the chronological age of samples and their developmental age in the reference time scale.

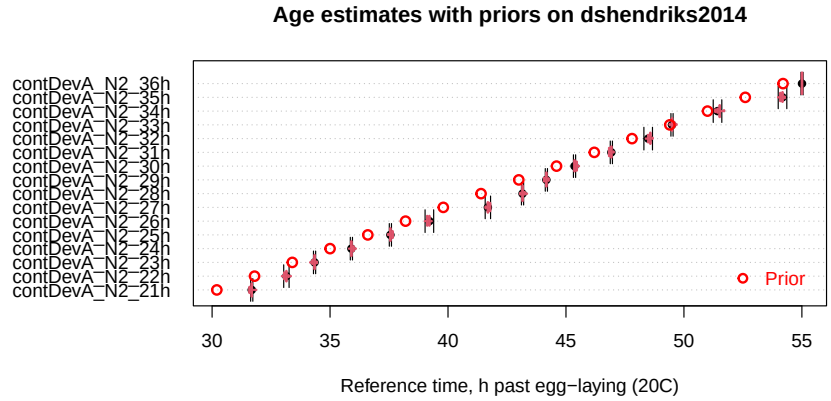
Once priors (means of the gaussian) are determined, the standard deviation is specified with the `prior.params` argument. This parameter will also indirectly change the weight of the prior over the correlation score for peak selection.

On the `dshendriks2014` example, we can use adjusted known chronological ages for 20°C.

```
# rough approximation based on our previous lm
priors <- dshendriks2014$p$age * 1.6 - 5
```

```
ae_dshendriks2014_prior <- ae(samp = dshendriks2014$g,
                              r_larv,
                              prior = priors,
                              prior.params = 10)
#>                  nb.genes
#> refdata             18718
#> samp                19595
#> intersect.genes     18718
#> Bootstrap set size is 6239
#> Performing age estimation...
#> Bootstrapping...
#>  Building gene subsets...
#>  Computing correlations...
#>  Performing age estimation...
#> Computing summary statistics...
```

```
plot(ae_dshendriks2014_prior,
     main="Age estimates with priors on dshendriks2014",
     show.prior = T, col.p = 'red', l.pos = 'bottomright')
```




```
# check prior estimates are equal to previous ones
all(ae_dshendriks2014_prior$age.estimates[,1] ==
      ae_dshendriks2014$age.estimates[,1])
#> [1] TRUE
```

As expected, here the estimates are identical to those without priors.

6 Code to generate data objects

Required libraries 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 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)))
}
```

Generating `dsaeschimann2017`.

```
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)
```

Generating `dshendriks2014`.

```
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)
```

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.

Anavy, Leon, Michal Levin, Sally Khair, Nagayasu Nakanishi, Selene L Fernandez-Valverde, Bernard M Degnan, and Itai Yanai. 2014. "BLIND Ordering of Large-Scale Transcriptomic Developmental Timecourses." *Development* 141 (5): 1161–66.

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

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.

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] limma_3.50.3       wormRef_0.5.0      RAPToR_1.2.0       ggpubr_0.6.0
#> [5] ggplot2_3.4.0.9000 BiocStyle_2.22.0
#>
#> loaded via a namespace (and not attached):
#>  [1] Rcpp_1.0.10         lattice_0.20-45     ica_1.0-3
#>  [4] tidyr_1.3.0         digest_0.6.31       utf8_1.2.3
#>  [7] R6_2.5.1            backports_1.4.1     evaluate_0.20
#> [10] highr_0.10          pillar_1.8.1        Rdpack_2.4
#> [13] rlang_1.0.6         rstudioapi_0.14     data.table_1.14.6
#> [16] car_3.1-1           jquerylib_0.1.4     magick_2.7.3
#> [19] Matrix_1.5-3        rmarkdown_2.20.1    labeling_0.4.2
#> [22] splines_4.1.2       stringr_1.5.0       munsell_0.5.0
#> [25] broom_1.0.3         compiler_4.1.2      xfun_0.37
#> [28] pkgconfig_2.0.3     mgcv_1.8-41         htmltools_0.5.4
#> [31] tidyselect_1.2.0    tibble_3.1.8        bookdown_0.32
#> [34] codetools_0.2-18    fansi_1.0.4         dplyr_1.1.0
#> [37] withr_2.5.0         rbibutils_2.2.13    grid_4.1.2
#> [40] nlme_3.1-161        jsonlite_1.8.4      gtable_0.3.1
#> [43] lifecycle_1.0.3     magrittr_2.0.3      scales_1.2.1
#> [46] cli_3.6.0           stringi_1.7.12      cachem_1.0.6
#> [49] carData_3.0-5       farver_2.1.1        ggsignif_0.6.4
#> [52] pryr_0.1.6          bslib_0.4.2         generics_0.1.3
#> [55] vctrs_0.5.2         tools_4.1.2         glue_1.6.2
#> [58] beeswarm_0.4.0      purrr_1.0.1         abind_1.4-5
#> [61] parallel_4.1.2      fastmap_1.1.0       yaml_2.3.7
#> [64] colorspace_2.1-0    BiocManager_1.30.19 rstatix_0.7.2
#> [67] knitr_1.42          sass_0.4.5
```