Supplementary material

Alexey Uvarovskii, Christoph Dieterich

2017-03-16

Contents

1 Introduction **1**
1.1 Kinetic model . . . 1
1.2 Measured quantities . . . 2

2 Model definition in pulseR **3**
2.1 Formulas . . . 3
2.2 Fractions . . . 3
2.3 Conditions . . . 4
2.4 Normalisation factors . . . 4
2.5 Simulation . . . 6

3 Fitting **7**
3.1 Create the PulseData object . . . 7
3.2 Set fitting options . . . 7
3.3 Fit . . . 8

4 Results **9**
4.1 Predictions . . . 9
4.2 Normalisation factors . . . 9
4.3 Gene-specific parameters . . . 10
4.4 Conclusion . . . 13

1 Introduction

The aim of this document is to validate the pulseR package using simulated data. First, we introduce the modelled system and how we simulate the read counts. Next, we explain how one applies `pulseR` to this data. We validate the performance of the package by comparison of the parameter estimations with their true values used in simulation.

1.1 Kinetic model

We consider a pulse-experiment with no spike-ins added. In this scheme, a labelled 4sU molecules are introduced to the medium and they are being incorporated in the nascent RNA.

Using pull down procedure, it is possible to get label-enriched fraction in order to quantify the kinetics rates of for the individual genes. We consider that all three fractions (total, labelled, unlabelled) are being sequenced and read counts are available for every gene under consideration. Let us assume to have 2 replicates for 4 different time points, i.e. in total there $2 \times 4 \times 3 = 24$ samples.

```
library(pulseR)

set.seed(259)

nGenes <- 1000
nReplicates <- 2
nTime <- 4
```

In the beginning of the experiment, the amount of the labelled RNA is zero and the unlabelled RNA equals the steady state amount μ. The labelled RNA is being synthesised with the rate s, and it degrades with the rate d. We assume that unlabelled one only degrades with the same rate d.
The following system of ordinary differential equations describes this processes:

$$\begin{aligned} \frac{d[\text{labelled RNA}]}{dt} &= s - d[\text{labelled RNA}] \\ \frac{d[\text{unlabelled RNA}]}{dt} &= -d[\text{unlabelled RNA}] \\ [\text{total RNA}] &= [\text{labelled RNA}] + [\text{unlabelled RNA}] \end{aligned} \tag{1}$$

For constant rates of degradation d and synthesis s, the amounts of RNA at time t are

$$\begin{aligned} [\text{total RNA}] &= \frac{s}{d} \equiv \mu \\ [\text{labelled RNA}] &= \mu(1 - e^{-dt}) \\ [\text{unlabelled RNA}] &= \mu e^{-dt}. \end{aligned}$$

1.2 Measured quantities

In reality, there are no pure labelled or unlabelled fractions in samples. That is why it may be important to model fraction cross-contamination. Here we assume that the labelled fraction consists of certain amount of the unlabelled one, and, vice versa, the unlabelled one is contaminated with the labelled molecules. Hence, we distinguish the number of labelled reads and the number of reads in the "labelled" fraction for a given gene. We will use *fraction* to refer to the model accounting for cross-contamination:

$$\begin{aligned} [\text{total fraction}] &= \mu \\ [\text{pull down}] &= \alpha_1[\text{labelled RNA}] + \alpha_2[\text{unlabelled RNA}] \\ [\text{flow through}] &= \alpha_3[\text{unlabelled RNA}] + \alpha_4[\text{labelled RNA}] \end{aligned} \tag{2}$$

2 Model definition in pulseR

One can simulate count data using the function `generateTestDataFrom`. This function takes as an input formulas for the mean read counts, a condition matrix (e.g. condition × time), normalisation factors and the parameter values used in the formulas.

2.1 Formulas

It is possible to generate formulas using the functions `growFrom0`, `grow`, `amount` and `degrade`, which represent the simplest cases of the concentration kinetics:

```
formulas <- list(
  total      = amount("mu"),
  labelled   = growFrom0("mu", "d", "time"),
  unlabelled = degrade("mu", "d", "time")
)
```

Alternatively, one can provide the expressions for the means explicitly:

```
formulas2 <- MeanFormulas(
  total        = mu,
  labelled     = mu * (1 - exp(-d*time)),
  unlabelled   = mu * exp(-d*time)
)
identical(formulas, formulas2)
```

```
## [1] TRUE
```

2.2 Fractions

According to the introduced terms for fractions, we need to provide the types of measured values, i.e. taking into account contamination of pull down and flow through fractions:

```
# important: the labels must be the same as in formulas,
# or they must be integer indexes
formulaIndexes <- list(
  total_fraction = 'total',
  pull_down      = c('labelled', 'unlabelled'),
  flow_through   = c('unlabelled', 'labelled')
)
```

The `formulaIndexes` list defines the linear relations for fractions described by the eq.2.

2.3 Conditions

We specify the condition `data.frame`, which includes information about the type of a sample and the time point of its measurements:

```
# create nReplicates * nTime * 3 (labelled, unlabelled, total) data points
conditions <- data.frame(
  condition = rep(names(formulaIndexes), each = nTime),
  time      = rep(1:nTime, length(formulaIndexes) * nReplicates))
rownames(conditions) <- paste0("sample_", seq_along(conditions$condition))
knitr::kable(conditions)
```

	condition	time
sample_1	total_fraction	1
sample_2	total_fraction	2
sample_3	total_fraction	3
sample_4	total_fraction	4
sample_5	pull_down	1
sample_6	pull_down	2
sample_7	pull_down	3
sample_8	pull_down	4
sample_9	flow_through	1
sample_10	flow_through	2
sample_11	flow_through	3
sample_12	flow_through	4
sample_13	total_fraction	1
sample_14	total_fraction	2
sample_15	total_fraction	3
sample_16	total_fraction	4
sample_17	pull_down	1
sample_18	pull_down	2
sample_19	pull_down	3
sample_20	pull_down	4
sample_21	flow_through	1
sample_22	flow_through	2
sample_23	flow_through	3
sample_24	flow_through	4

2.4 Normalisation factors

Total amounts of labelled and unlabelled RNA are changing during the experiment. It may affect the normalisation coefficients defined in eq.2. That is why it is useful to introduce such coefficients for every time point, and, hence, we define samples groups depending on their

measurement time and fraction. Except the total fraction, other samples are considered to belong to different group if the time points are different:

```
fractions <- as.character(interaction(conditions))
# assume that the total RNA amount does not change with the time
# so all total fractions can be treated together
fractions[grep("total", fractions)] <- "total_fraction"
knitr::kable(cbind(rownames(conditions),fractions))
```

	fractions
sample_1	total_fraction
sample_2	total_fraction
sample_3	total_fraction
sample_4	total_fraction
sample_5	pull_down.1
sample_6	pull_down.2
sample_7	pull_down.3
sample_8	pull_down.4
sample_9	flow_through.1
sample_10	flow_through.2
sample_11	flow_through.3
sample_12	flow_through.4
sample_13	total_fraction
sample_14	total_fraction
sample_15	total_fraction
sample_16	total_fraction
sample_17	pull_down.1
sample_18	pull_down.2
sample_19	pull_down.3
sample_20	pull_down.4
sample_21	flow_through.1
sample_22	flow_through.2
sample_23	flow_through.3
sample_24	flow_through.4

We use the internal function `pulseR:::addKnownToFormulas` in order to get the correct form of the list with the normalisation factors.

```
known <- pulseR:::addKnownToFormulas(formulas, formulaIndexes, conditions)
normFactors <- known$formulaIndexes[unique(names(known$formulaIndexes))]
# the total fractions have the reference factor = 1
normFactors <- normFactors[-grep("total", names(normFactors))]
normFactors <- c(list(total_fraction = 1), normFactors)
# linear combinations with the coefficient 0.1 for contaminating fraction
```

```
# and the coefficient 3 for the main fraction of the sample
normFactors <- relist(c(1, rep(3, length(unlist(normFactors)) - 1)),
                      normFactors)
normFactors[-1] <- lapply(normFactors[-1], '[[<-',2,.10)
str(normFactors)
## List of 9
##  $ total_fraction: num 1
##  $ pull_down.1   : num [1:2] 3 0.1
##  $ pull_down.2   : num [1:2] 3 0.1
##  $ pull_down.3   : num [1:2] 3 0.1
##  $ pull_down.4   : num [1:2] 3 0.1
##  $ flow_through.1: num [1:2] 3 0.1
##  $ flow_through.2: num [1:2] 3 0.1
##  $ flow_through.3: num [1:2] 3 0.1
##  $ flow_through.4: num [1:2] 3 0.1
```

2.5 Simulation

First, we define the true parameter values, on the basis of which we generate the read count matrix. We sample the mean expression level and the degradation rates from finite intervals:

```
# set size factor for the rnbinom function
par <- list(size = 1e2)
# set mean read number for every gene
# by sampling from the log-uniform distribution
par$mu <- exp(runif(nGenes, 10, log(1e5)))
# set the degradation rates from log-uniform distribution
par$d <- exp(runif(nGenes,log(0.05), log(5)))
```

Now we have all the information in order to sample read counts
from the negative binomial distribution using the `generateTestDataFrom` function:

```
# the generateTestDataFrom needs normalisation factors
# provided for every individual sample (i.e. we need to utilise
# the group information ourselves)
allNormFactors <- pulseR:::multiplyList(normFactors, fractions)

counts <- generateTestDataFrom(
  formulas, formulaIndexes, allNormFactors, par, conditions)
counts[1:5,1:10]
```

```
##       [,1]  [,2]  [,3]  [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
## [1,] 36067 33146 39598 39202  74821  82398 103312 114898  63796  26901
## [2,] 84736 68838 75668 66855  36777  66120 101734 115335 203235 150688
## [3,] 66739 61764 75089 57253 205699 190955 204946 180470  21867   6022
```

```
## [4,] 25707 31834 29162 25173  17262  27879  35404  48975  72754  70421
## [5,] 46299 53235 58147 58695  20720  30305  45127  63860 122184 139027
```

```
dim(counts)
```

```
## [1] 1000   24
```

3 Fitting

We would like to investigate how well the true parameter values for the model defined in eq.1 can be recovered using the `pulseR` package.

This section describes a usual workflow for the analysis of the count data originating from the spike-ins-free experiment defined above. This is an entry point of the package application to the real data.

3.1 Create the PulseData object

The fitting procedure requires the definition of the model formulas, sample description and raw data as a single `PulseData` object. We use previously defined objects, namely, the generated read counts, the condition matrix, relations between formulas and samples. In addition, we provide the information (`fractions`) how to split the samples during the normalisation:

```
pd <- PulseData(
  counts = counts,
  conditions = conditions,
  formulas = formulas,
  formulaIndexes = formulaIndexes,
  groups = fractions
)
```

3.2 Set fitting options

Parameter values are estimated by maximum likelihood method using L-BFGS-S optimisation procedure (see help for `optim`). This function requires to define boundaries for possible parameter values and a stopping criteria. In `pulseR`, the fitting procedure uses relative change in parameter values between sequential iterations as the stopping rule ("relative tolerance"). Normalisation factors can be recovered only up to a constant multiplicative factor.
By default, the first normalisation factor is always equal 1 (in our case, for the total fraction). However, the user must provide formal boundary values for the first factor as well, although they will be ignored. This is done for the sake of consistency.

```
# lower and upper boundaries for the normalisation factors.
# The total one will be ignored, because the first coefficient
# is always the reference one with the value 1.
# The reason is that we can identify the means of read numbers only
# up to an unknown multiplier.
lbNormFactors <- list(
  total_fraction = 1,
  pull_down      = c(.1,.010),
  flow_through   = c(.1,.010))
ubNormFactors <- list(
  total_fraction = 1,
  pull_down      = c(10, 2),
  flow_through   = c(10, 2))
# set lower and upper boundaries for the rest of the parameters
opts <- setBoundaries(
  list(mu = c(1, 1e6), d = range(par$d) * c(1 / 5, 2)),
  normFactors = list(lbNormFactors, ubNormFactors))
# set the tolerance for the gene-specific parameters and
# the normalisation factors
opts <- setTolerance(params = .01, normFactors = .1, options = opts)
```

3.3 Fit

The L-BFGS-U algorithm requires an initial guess for the parameters values. The outcome of the fitting procedure may depend on the choice of the initial values. It is possible to generate random parameter values within the defined boundaries using the function `initParameters`. The function will generate only the parameter values, which were not set in the argument `par`. Since mean read number and the degradation rate are unique for every gene, we also specify it in the argument `geneParams`.

```
# initialise mu and d as gene-specific parameters, i.e. they must
# be sampled for every single gene individually
initPars <- initParameters(par = list(d=.1, mu=counts[,1]),
                           geneParams = c("mu", "d"), pd, opts)
str(initPars)
```

```
## List of 4
##  $ d          : num [1:1000] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
##  $ mu         : num [1:1000] 36067 84736 66739 25707 46299 ...
##  $ size       : num 5.39e+09
##  $ normFactors:List of 9
##   ..$ : num 1
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
```

```
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
##   ..$ : num [1:2] 1 0.01
```

Now we are ready to start the fitting:

```
res <- fitModel(pd, initPars, opts)
```

4 Results

The values of fitted parameters together with the normalisation factor estimations and `size` parameter of the negative binomial distribution (see `dnbinom` help) are in the list `res`:

```
names(res)
```

```
## [1] "d"           "mu"          "size"        "normFactors"
```

4.1 Predictions

The model predictions for the mean read numbers can be estimated using the `predictExpression` function. The plot shows very good fit of the model to the count data, fig. 1.

```
# get the expected read number for every data point in counts
pr <- predictExpression(pd, res)
par(mfrow=c(1,2))
plot(y = as.vector(pr$predictions) + .5,
     x = as.vector(pd$counts) + .5,
     pch = 16, cex = .3, log = 'xy',
     ylab = "predicted", xlab = "experiment")
plot(y =((as.vector(pr$predictions) + .5) / (as.vector(pd$counts) + .5)),
     x = as.vector(pd$counts) + .5,
     pch = 16, cex = .3, log = 'x',
     xlab = "predicted", ylab = "predicted/experiment")
```

4.2 Normalisation factors

The normalisation of the main type of RNA (labelled for the labelled fraction and unlabelled for the unlabelled) are well recovered. However, the factors for the contaminant have a poor

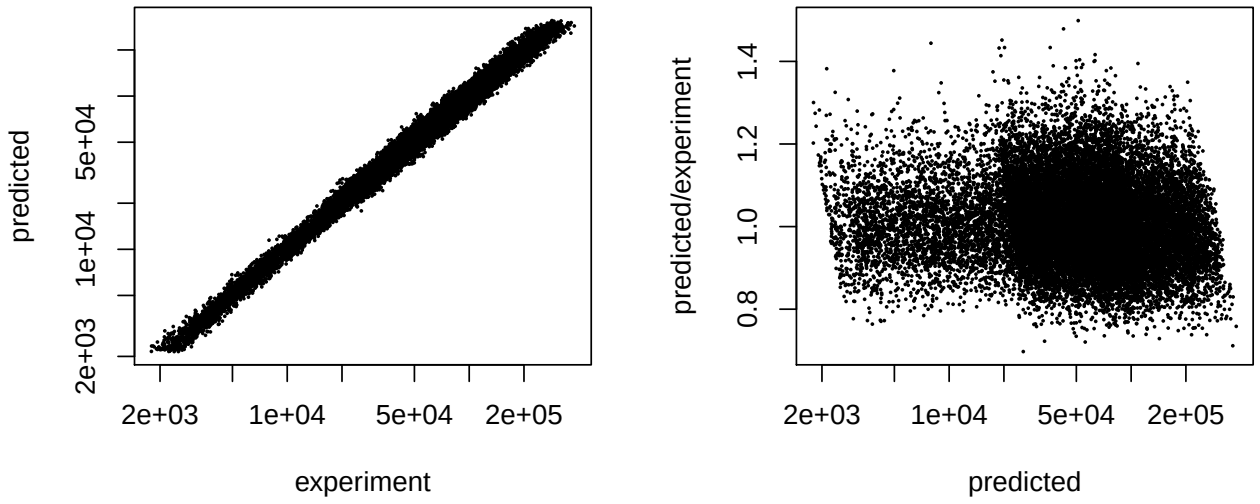

Figure 1: Estimations of mean read numbers compared with the read numbers in the simulated data in absolute values (left) and as a ratio (right).

estimation, since its amount is much lower and estimation of its contribution is much more affected by noise then, see second terms in the list of relative errors:

```
str(relist(1 - unlist(res$normFactors) / unlist(normFactors), normFactors))
```

```
## List of 9
##  $ total_fraction: num 0
##  $ pull_down.1   : num [1:2] -0.00477 0.34404
##  $ pull_down.2   : num [1:2] 0.00838 0.58401
##  $ pull_down.3   : num [1:2] 0.000403 0.9
##  $ pull_down.4   : num [1:2] 0.0131 0.9
##  $ flow_through.1: num [1:2] -0.0136 0.9
##  $ flow_through.2: num [1:2] 0.000262 0.106428
##  $ flow_through.3: num [1:2] -0.00421 0.02979
##  $ flow_through.4: num [1:2] -0.00553 0.02353
```

According to the model structure, bias in the estimation of the normalisation factors affects other model parameters, e.g. the overall degradation rate. That is why usage of spike-ins can be beneficial, since it allows to refer to absolute quantities.

4.3 Gene-specific parameters

The estimation of the expression levels are in a good accordance with the true μ parameter values, because there are several data points for the total fraction, and this fraction represents the very level of the gene expression, fig. 2.

The estimation of the degradation rate is based on the less number of samples (the total fractions provide the information only about the expression level).

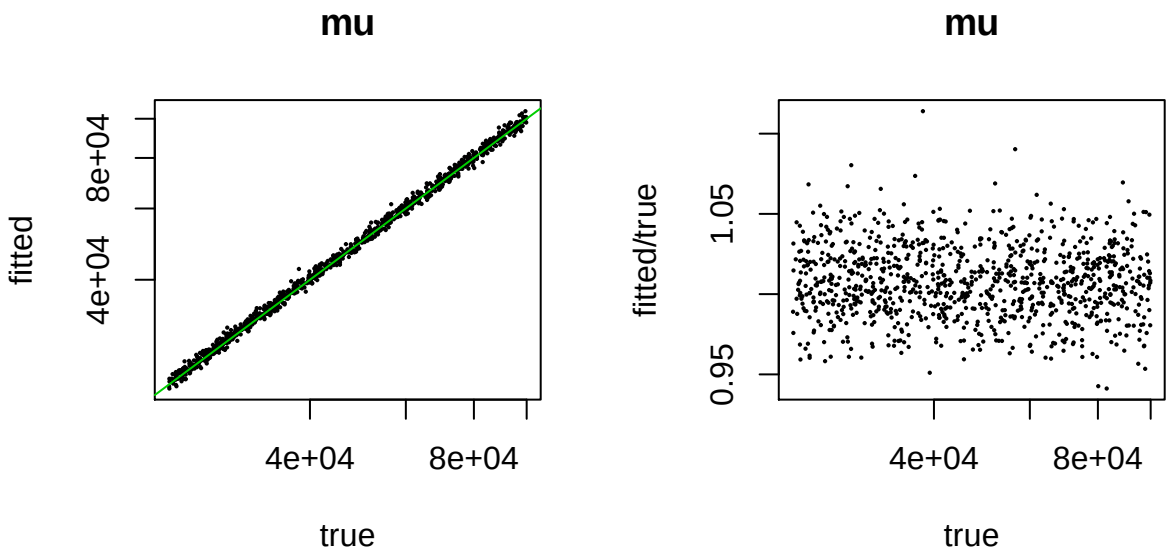

Figure 2: Left: Fitted value for the "mu" parameter versus the true parameter value. Right: Distribution of the relative error in the mean read number estimation ("mu"). The line $y = x$ is in green.

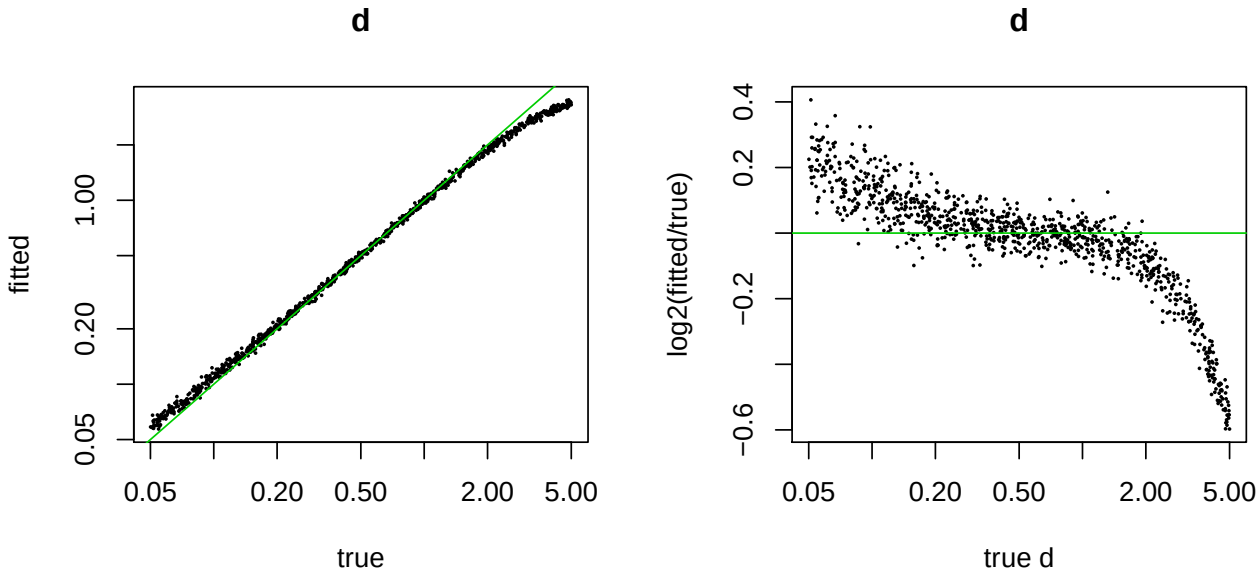

Figure 3: Fitted and true values for the degradation rate d in absolute values (left) and as a ratio (right). The relative error is higher for the genes with degradation rates close to extreme values (explained in the text). The lines $y = x$ (left) and $y = 0$ (right) are in green.

The error in estimation of d for lowly degrading genes (small d values) is higher, fig. 3 right, because misfit in d with small values has a minor effect on the mispredictions of the read number due to an exponential dependance (e.g. μe^{-dt}), fig. 4. Hence, genes with low degradation rate are not

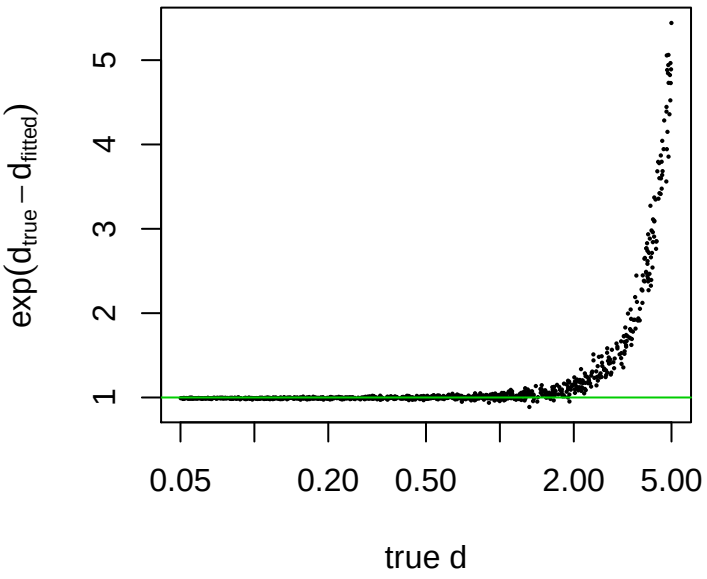

Figure 4: Contribution of the error in the degradation rate d estimation to the mean read number predition μe^{-dt} for the time point $t = 1$. Misfit in small degradation rates has a little contribution to mean read number misprediction.

From another hand, parameters for the genes with a fast kinetics are difficult to capture, if the time resolution is not fine enough. This leads to higher error in estimation of d if true d value is high, fig. 3.

Identifiability of the parameters can be affected by the model structure and experimental design. A misfit in the normalisation factors can shift estimations of the degradation rates up to a constant factor.

If d is small, the kinetics of fractions can be approximated via Taylor expansion as

$$\alpha_1 \mu e^{-dt} + \alpha_2 \mu (1 - e^{-dt}) = \alpha_1 \mu (1 - dt) + \alpha_2 \mu dt + o(d) = \alpha_1 \mu + (\alpha_2 - \alpha_1) d\mu t + o(d)$$

The degradation rate d and the difference of the normalisation factors $(\alpha_2 - \alpha_1)$ appear to be in multiplication, which results in poor identifiability of this parameters in separation from each other if a considerable number of genes with low d are present.

This results demonstrate possible issues which can take place in the experiments without added spike-ins controls. However, experimental design for the pulse-chase RNA-seq experiments goes beyound the skope if this application note.

4.4 Conclusion

Here we presented a validation of the pulseR package performance on the simulated data. We show that the values of the gene-specific parameters can be recovered by the maximum likelihood estimation even in the absence of the spike-in control. However, this approach is limited to the systems, which has structurally and practically identifiable parameters. The identifiability of the parameters depends on the number of replicates, the type of the fractions provided, the cross-contamination rates and the sequence depth (read numbers per feature). The `pulseR` package simplifies the MLE fitting procedure for pulse-chase experiments. However, it does not solve identifiability issues if the system is ill-posed and responsible and thorough experimental design is assumed.