

# RNA-Seq data normalization using artificial spike-ins

Carina Demel and Julien Gagneur

June 6, 2016

## Abstract

A common problem of RNA-Seq experiments is the normalization of different samples, especially when they were also treated differently. A special case is the 4sU-labeling followed by deep-sequencing, where two libraries are generated from each RNA fraction: First, in one part of the fraction, the 4sU-labeled RNAs are pulled down and gives the labeled (“L”, called 4sU-Seq or TT-Seq) fraction of the experiment for library preparation. Second, the total RNA (labeled and unlabeled, “T”) is used for the preparation of another RNA-Seq library. As the RNA amount for library preparation is usually stringent, the sequencing results do not reflect reality. Here, we have to account for the sequencing depth of different samples and especially adjust the ratio of Labeled to Total RNA-Seq libraries. Additionally, the labeled RNA extraction is not perfect and also a little bit of unlabeled RNA may contaminate the labeled fraction. As the gene-expression also always varies a little bit for biological samples, read counts from expressed genes might lead to confusing estimations. Therefore, we apply our normalization approach only to artificial spike-ins from the External RNA Control Consortium (ERCC), for which the initial amount of each spike-in is known and the same across all samples. Some of the spike-ins were 4sU-labeled by in-vitro transcription, so the cross-contamination of unlabeled spike-ins in labeled samples can be monitored. The goal of this package is to reliably estimate sequencing depths and cross-contamination rates per sample, given 4sU-/TT-Seq and total RNA-Seq data.

# Contents

<b>1</b>	<b>Background</b>	<b>2</b>
<b>2</b>	<b>Getting started</b>	<b>3</b>
2.1	Input data . . . . .	3
<b>3</b>	<b>Normalizing RNA-Seq samples using spike-in counts</b>	<b>4</b>
<b>4</b>	<b>Estimating gene-specific synthesis and degradation rates</b>	<b>6</b>
4.1	Providing gene-wise dispersion estimates . . . . .	6
4.2	Gene-wise synthesis and degradation rate estimates . . . . .	7
<b>5</b>	<b>Session Information</b>	<b>8</b>
<b>6</b>	<b>References</b>	<b>9</b>

## 1 Background

As described in [?] 4sU-seq allows to monitor changes in the RNA metabolism. If cells are exposed to 4sU, they rapidly take up this Uridine analog and incorporate it into newly-synthesized RNAs. This way, newly-synthesized RNAs are labeled and can be extracted from the total RNA in the sample. The longer the labeling time, e.i. the time from 4sU addition to harvesting the cells, the bigger is the proportion of labeled RNAs among all RNAs. Figure 1 shows how the amounts of total, labeled and unlabeled RNA change over time in a typical 4sU experiment. The synthesis rate  $\mu$  and degradation rate  $\lambda$  are assumed to be constant over time.

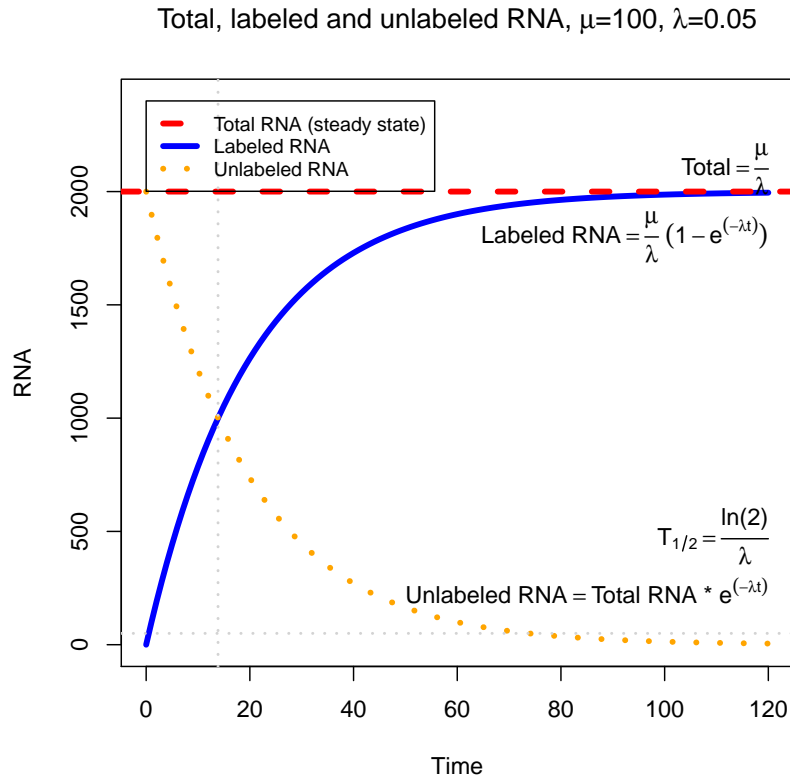


Figure 1: Time course of RNA amounts in a 4sU-experiment

## 2 Getting started

In this vignette, you will learn how to estimate sample specific sequencing depths and cross-contamination rates from spike-in counts. These values can be used to normalize gene expression values obtained by RNA-Seq and thus estimate gene-specific synthesis and degradation rates. Before starting, the package must be loaded by:

```
library("RNAlife")
```

### 2.1 Input data

For the estimation of sample and gene-specific parameters, two count tables are required, namely a count table with counts for the artificial spike-ins and a count table with counts for the genes (or features like exons, introns etc). Those count tables should have the same number of columns, which represent the different samples. The "RNAlife" package provides some example data.

```
data(spikein.counts)
data(gene.counts)
spikein.counts
```

	L	A	Rep1	L	A	Rep2	L	B	Rep1	L	B	Rep2	T	A	Rep1	T	A	Rep2	T	B	Rep1	T	B	Rep2
Spike12		18060			10089			945			13894		208559			11544			16075			158490		
Spike2	1321853			1086426			86181			1323880			265385			16156			19920			190858		
Spike4	1115970			1073272			96470			1282230			210573			12792			18299			176584		
Spike5	13190			3969			801			3862			216795			10127			19261			191365		
Spike8	1771066			1648524			200196			2075083			200196			12508			19825			191829		
Spike9	7865			1462			382			1106			116795			5220			10704			106093		

Before we start with the normalization, we also need to know the length of the spike-ins and which of the artificial spike-ins were labeled by in-vitro transcription. This information should be provided by single vectors. For the given example count table, the package also provides example vectors:

```
data(spikein.labeling)
data(spikein.lengths)
spikeins = rownames(spikein.counts)
```

Each of the experimental samples should also be explained by a vector giving the experimental condition (e.g. treatment, time point), the labeling condition ("L" or "T"), and the replicate number. In case no replicates are available just use the same integer for all samples. For the given example, the samples data.frame contains those three vectors:

```
data(samples)
samples
```

	conditions	conditions.labeling	replicates
1	A	L	1
2	A	L	2
3	B	L	1
4	B	L	2
5	A	T	1
6	A	T	2
7	B	T	1
8	B	T	2

### 3 Normalizing RNA-Seq samples using spike-in counts

The sample specific parameters like sequencing depth and cross-contamination rate are estimated from spike-in counts only. Therefore, we fit a generalized linear model (GLM) of the Negative Binomial family with a log link function. The response of the GLM are the observed spike-in counts, and the terms that specify the linear predictor of the response are comprised of:

- a sample specific factor (that reflects the sample specific sequencing depth),
- a labeled sample specific factor (that reflects the control for cross contamination (only estimated for unlabeled spike-ins in labeled samples)), and
- a spike-in specific factor to allow for some spike-in specific variation e.g. due to sequence biases.

Additionally, the length of each spike-in is used as an offset, i.e. a known slope for the covariate.

The GLM works on a dataframe, consisting of  $n \times m$  rows, where  $n$  is the number of spike-ins and  $m$  is the number of samples. For every observed count of every spike-in in every sample, the `data.frame` holds the information about the spike-in, the labeling status of the spike-in, the (log) length of the spike-in, the sample name, and the labeling status of the sample. This `data.frame` can be generated by the following function for visualization, but when the normalization procedure is called (below), the `data.frame` is created inside the function.

```
sample.names = apply(samples,1,paste,collapse=" ")
mat = spikein.dataframe(counts=spikein.counts, spikeins=spikeins,
  spikein.lengths=spikein.lengths,
  spikein.labeling=spikein.labeling,
  samples=sample.names,
  conditions.labeling=samples$conditions.labeling)
summary(mat)
```

spike	length	spike.labeled	sample	sample.labeling
Spike12:8	Min. :1023	L:24	A L 1 : 6	L:24
Spike2 :8	1st Qu.:1023	U:24	A L 2 : 6	T:24
Spike4 :8	Median :1038		A T 1 : 6	
Spike5 :8	Mean :1051		A T 2 : 6	
Spike8 :8	3rd Qu.:1061		B L 1 : 6	
Spike9 :8	Max. :1124		B L 2 : 6	
			(Other):12	

counts	control.for.cross.contamination	ccc	log.length
Min. : 382	F:36	F :36	Min. :6.93
1st Qu.: 11334	T:12	L 1 : 3	1st Qu.:6.93
Median : 53050		L 2 : 3	Median :6.94
Mean : 324224		L 3 : 3	Mean :6.96
3rd Qu.: 209062		L 4 : 3	3rd Qu.:6.97
Max. :2075083			Max. :7.02

```
head(mat, n=10)
```

	spike	length	spike.labeled	sample	sample.labeling	counts
1	Spike12	1023	U	A L 1	L	18060
2	Spike2	1023	L	A L 1	L	1321853
3	Spike4	1033	L	A L 1	L	1115970
4	Spike5	1042	U	A L 1	L	13190
5	Spike8	1124	L	A L 1	L	1771066
6	Spike9	1061	U	A L 1	L	7865
7	Spike12	1023	U	A L 2	L	10089
8	Spike2	1023	L	A L 2	L	1086426
9	Spike4	1033	L	A L 2	L	1073272
10	Spike5	1042	U	A L 2	L	3969

	control.for.cross.contamination	ccc	log.length
1	T L 1		6.93

2	F	F	6.93
3	F	F	6.94
4	T L 1		6.95
5	F	F	7.02
6	T L 1		6.97
7	T L 2		6.93
8	F	F	6.93
9	F	F	6.94
10	T L 2		6.95

For the Negative Binomial GLM we use the implementation of the MASS package:

```
glm.nb(counts ~ offset(log.length)+sample+ccc+spikein, data=mat, link="log")
```

The following function combines the generation of the data.frame, the fit of the glm and the formatting of the results and returns all the fitted factors.

```
norm.res = spikein.normalization(spikein.counts, spikeins,
                                spikein.lengths, spikein.labeling,
                                samples=colnames(spikein.counts),
                                samples$conditions.labeling)
summary(norm.res)
```

	Length	Class	Mode
results	3	data.frame	list
intercept	1	-none-	numeric
sequencing.depth	8	-none-	numeric
cross.contamination	8	-none-	numeric
spike.specific.bias	6	-none-	numeric
fitted.counts	48	-none-	numeric

```
norm.res$sequencing.depth
```

L A Rep1	L A Rep2	L B Rep1	L B Rep2	T A Rep1	T A Rep2	T B Rep1	T B Rep2
1.00000	0.90453	0.08914	1.10879	0.16871	0.00919	0.01453	0.14209

```
norm.res$cross.contamination
```

L A Rep1	L A Rep2	L B Rep1	L B Rep2	T A Rep1	T A Rep2	T B Rep1	T B Rep2
0.01200	0.00445	0.00721	0.00411	1.00000	1.00000	1.00000	1.00000

Figure 2 shows that the measured counts can be explained by the fitted variables.

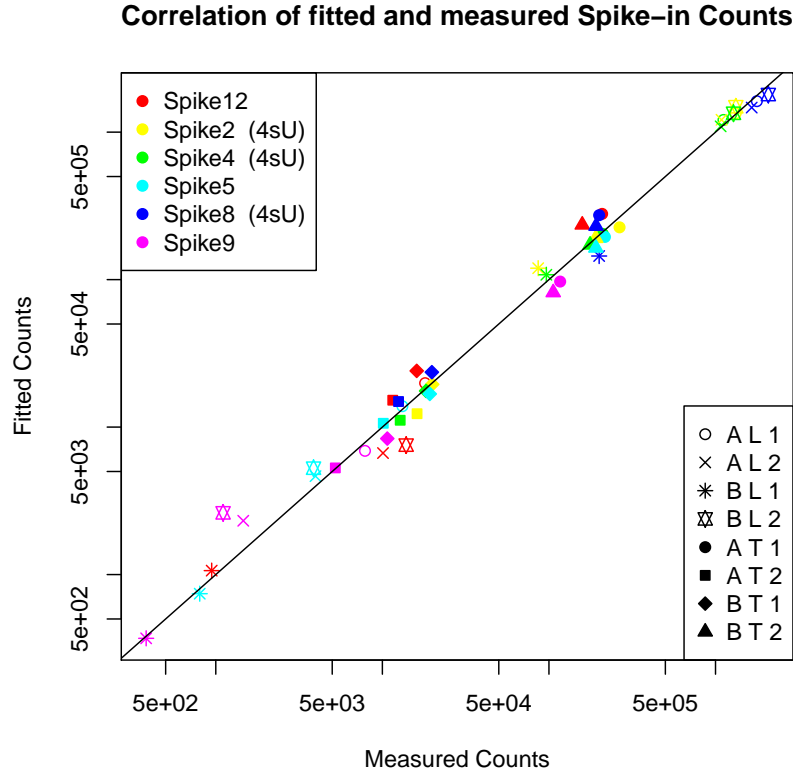


Figure 2: Correlation of fitted and measured spike-in counts, different colors indicate different spike-ins, different symbols indicate different samples

## 4 Estimating gene-specific synthesis and degradation rates

### 4.1 Providing gene-wise dispersion estimates

For each gene, a single dispersion estimate for all 4sU-Seq samples and for all Total RNA-Seq samples is needed. Here, we can use the method provided in the DESeq2 package [?]. The wrapper function `estimateGeneDispersion` applies the DESeq2 algorithm to all genes, while separating the count table according to the RNA-Seq protocol (labeled or total RNA). It is possible to choose between all provided DESeq dispersion estimates, namely the genewise maximum likelihood dispersion estimate (“`dispGeneEst`”), the smooth curve fitted through the gene-wise dispersion estimates (“`dispFit`”) and the genewise dispersion estimates shrunk towards the fitted curve (“`dispMAP`”, default). The input for the `estimateGeneDispersion` function is therefore only a  $n \times m$  matrix, providing count data for  $n$  genes under  $m$  conditions, and a vector of length  $m$ , indicating for each condition if it was a labeled 4sU-Seq sample (“L”) or a total RNA-Seq sample (“T”). The function returns a matrix consisting of  $n$  rows and 2 columns, the Labeled (“L”) and Total (“T”) dispersion estimates for all  $n$  genes.

The gene counts in the following example are randomly generated read counts for 12 genes, under the same experimental settings as used before.

```
data(gene.counts)
dispersion = estimateGeneDispersion(gene.counts, samples$conditions.labeling,
                                   samples$conditions)
head(dispersion)
```

	L	T
gene_1	0.268	0.00289
gene_2	0.210	0.00249
gene_3	0.217	0.00212
gene_4	0.249	0.00210

```
gene_5 0.203 0.00218
gene_6 0.209 0.00220
```

Note that, if a gene's dispersion cannot be estimated (due to only 0 counts in all relevant samples), it is set to the median dispersion values across all other genes. If you don't have replicates or there are not multiple conditions to test, please estimate dispersions across labeled and total samples together, by setting all `conditions.labeling` to the same value (e.g. "T").

## 4.2 Gene-wise synthesis and degradation rate estimates

To estimate gene-specific synthesis and degradation rates, we use the same assumption as for the spike-in read counts. The number of counts for each gene  $i$  in a sample  $j$  is dependent on the gene's length  $L_i$ , the labeled and unlabeled amounts of RNA for gene  $i$ , the sequencing depth of the sample  $\sigma_j$ , the cross-contamination rate in the sample  $\epsilon_j$  and the number of cells  $N$ . The labeled RNA amount  $\alpha_i$  and the unlabeled RNA amount  $\beta_i$  for each gene  $i$  is then optimized such that the difference between the expected read counts and the observed read counts for gene  $i$  is minimized.

```
data(gene.lengths)
seq.depths = norm.res$sequencing.depth
cross.cont = norm.res$cross.contamination
fittingres = estimate.rates(gene.counts, dispersion, gene.lengths, samples$conditions,
                           samples$conditions.labeling, samples$replicates, cross.cont, seq.depths, N=1,
                           consider.replicates=TRUE, lab.time=c(5,5), alpha.initial=1, beta.initial=1,
                           gene.indices=1:12)
names(fittingres)

[1] "alpha"          "beta"           "loss"           "exp.counts.L"
[5] "exp.counts.T"   "mu"             "lambda"         "half.life"
[9] "alpha.initial"  "beta.initial"

# Synthesis rates
head(fittingres$mu)

      A      B
gene_1 1444 3938
gene_2 1164 1109
gene_3  240  301
gene_4  220  356
gene_5  371  301
gene_6  553  473

# Degradation rates
head(fittingres$lambda)

      A      B
gene_1 1.35 1.52
gene_2 1.34 1.33
gene_3 1.04 1.09
gene_4 1.04 1.13
gene_5 1.14 1.11
gene_6 1.21 1.18
```

Note: The estimation can be performed for single genes or a subset of genes when setting the `gene.indices` to the corresponding row number in the count matrix `gene.counts`. Estimates can also be obtained for only one of the replicates by setting `consider.replicates=FALSE` and indicating with `rep=1` the replicate to be considered (as named in the experimental matrix `samples`).

The correctness of the fit can be checked by comparing the estimated read counts with the observed read counts.

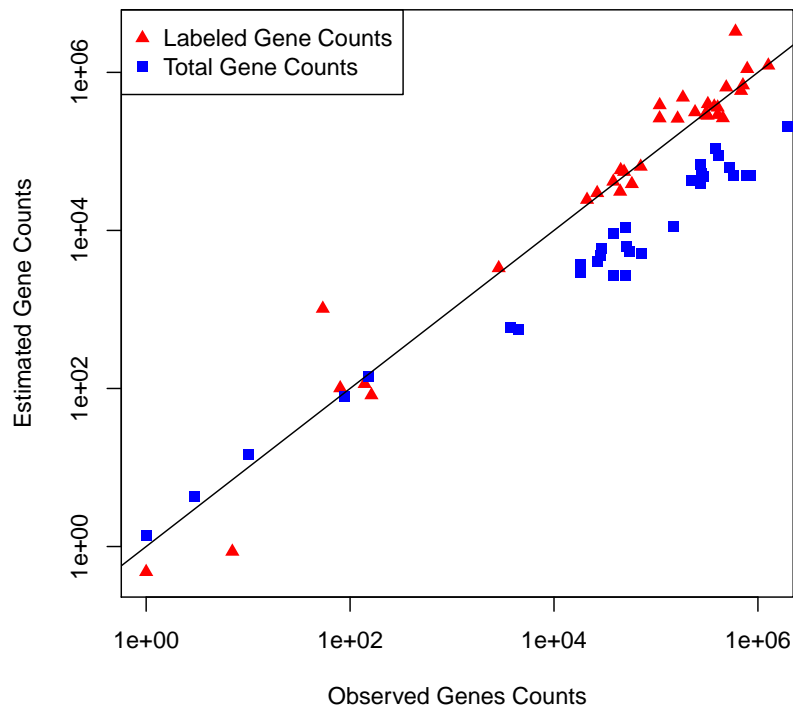


Figure 3: Correlation of fitted and measured gene counts in Labeled (red) and Total (blue) RNA-seq samples.

## 5 Session Information

The session information records the versions of all the packages used in the generation of the present document.

```
sessionInfo()
```

```
R version 3.1.0 (2014-04-10)
```

```
Platform: x86_64-apple-darwin13.1.0 (64-bit)
```

```
locale:
```

```
[1] C/de_DE.UTF-8/de_DE.UTF-8/C/de_DE.UTF-8/de_DE.UTF-8
```

```
attached base packages:
```

```
[1] parallel stats4 stats graphics grDevices utils datasets
```

```
[8] methods base
```

```
other attached packages:
```

```
[1] RNAlife_0.5 DESeq2_1.6.3
[3] RcppArmadillo_0.6.500.4.0 Rcpp_0.12.3
[5] GenomicRanges_1.18.4 GenomeInfoDb_1.2.5
[7] IRanges_2.0.1 S4Vectors_0.4.0
[9] BiocGenerics_0.12.1 MASS_7.3-45
```

```
loaded via a namespace (and not attached):
```

```
[1] AnnotationDbi_1.28.2 BBmisc_1.9 BatchJobs_1.6
[4] Biobase_2.26.0 BiocParallel_1.0.3 DBI_0.3.1
[7] Formula_1.2-1 Hmisc_3.17-1 RColorBrewer_1.1-2
```



[10] RSQLite_1.0.0	XML_3.98-1.3	XVector_0.6.0
[13] acepack_1.3-3.3	annotate_1.44.0	base64enc_0.1-3
[16] brew_1.0-6	checkmate_1.7.0	cluster_2.0.3
[19] codetools_0.2-14	colorspace_1.2-6	digest_0.6.9
[22] fail_1.3	foreach_1.4.3	foreign_0.8-66
[25] genefilter_1.48.1	geneplotter_1.44.0	ggplot2_2.1.0
[28] grid_3.1.0	gridExtra_2.0.0	gtable_0.1.2
[31] iterators_1.0.8	lattice_0.20-33	latticeExtra_0.6-26
[34] locfit_1.5-9.1	magrittr_1.5	munsell_0.4.2
[37] nnet_7.3-11	plyr_1.8.3	rpart_4.1-10
[40] scales_0.3.0	sendmailR_1.2-1	splines_3.1.0
[43] stringi_1.0-1	stringr_1.0.0	survival_2.38-3
[46] tools_3.1.0	xtable_1.8-0	

## 6 References