

Performance assessment of *vsn* with simulated data

Wolfgang Huber

March 19, 2007

Contents

1	Overview	1
2	Number of features n	1
3	Number of samples d	2
4	Number of strata	2
5	Differentially expressed genes	3
6	Missing values	4
7	Incremental normalization	6
8	Session Info	6

1 Overview

The purpose of this vignette is to assess that the software in *vsn* does what it is intended do, and in particular, to assess the performance of the parameter estimation on simulated data where the true parameters are known.

There are two functions `sagmbSimulateData` and `sagmbAssess` that can be used to generate simulated data and assess the difference between the 'true' and 'estimated' data calibration and transformation by *vsn*. This vignette demonstrates some examples. Please refer to reference [1] for more detail on the simulation model, the assessment strategy and a comprehensive suite of assessments with respect to the number of features n , the number of arrays d , the fraction of differentially expressed genes de , and the fraction of up-regulated genes up .

2 Number of features n

Fig. 1 shows the estimation error for the transformation (i.e. the root mean squared difference between true and estimated transformed data) as a function of the number of features n . If *vsn* works correctly, the estimation error should decrease roughly as $n^{-1/2}$.

```
> n = 1000 * 2^seq(-2, 5)
> makeFig("fign1", 1, 1, {
```

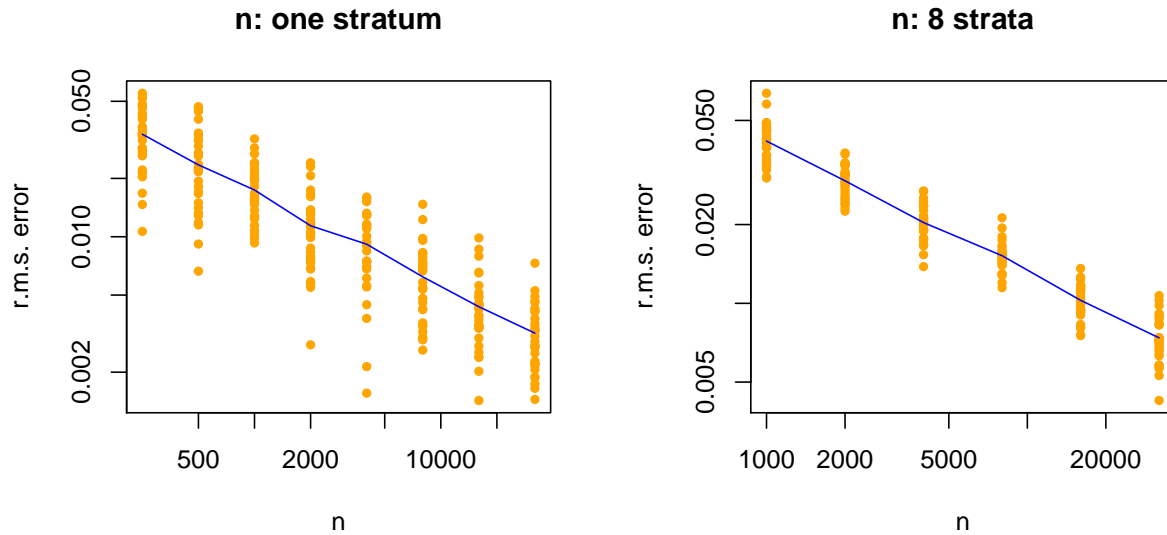


Figure 1: Estimation error as a function of the number of features n . If `vsn` works correctly, the estimation error should decrease roughly as $n^{-1/2}$.

```
+   res = sim(n = n)
+   myPlot(n, res, main = "n: one stratum")
+ })

> n = 1000 * 2^seq(0, 5)
> makeFig("fign2", 1, 1, {
+   res = sim(n = n, nrstrata = 8)
+   myPlot(n, res, main = "n: 8 strata")
+ })
```

3 Number of samples d

Fig. 2a shows the estimation error as a function of the number of samples d . This curve is essentially flat. This is because the number of parameters that need to be estimated is proportional to d , so the "number of data points per parameter" is constant in this plot (in contrast to Fig. 1).

```
> makeFig("figd", 1, 1, {
+   d = 2^seq(1, 5)
+   res = sim(d = d)
+   myPlot(d, res, main = "a) d")
+ })
```

4 Number of strata

In Fig. 2b, we see the estimation error as a function of the number of strata. It should increase, since for each stratum, we need to estimate separate parameters, and if the overall number of features

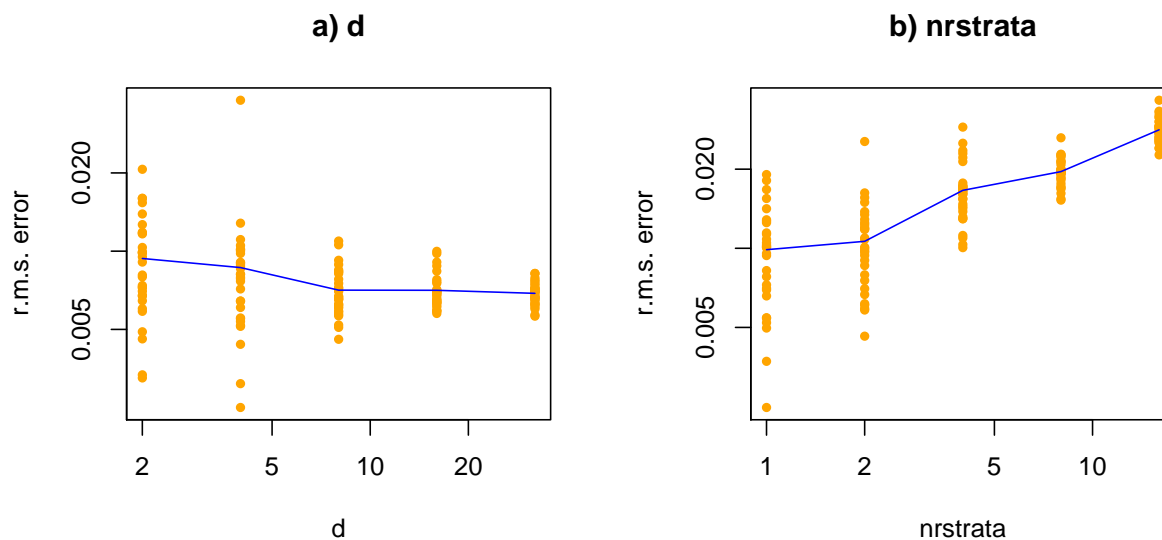


Figure 2: Estimation error as a function of (a) the number of samples and (b) the number of strata. See Sections 3 and 4.

does not change, more strata means less data per parameter.

```
> makeFig("fignrstrata", 1, 1, {
+   nrstrata = 2^seq(0, 4)
+   res = sim(nrstrata = nrstrata)
+   myPlot(nrstrata, res, main = "b) nrstrata")
+ })
```

5 Differentially expressed genes

In the following code, `de` is the fraction of differentially expressed genes. We run the simulation both with default settings `lts.quantile=0.9` and the more robust `lts.quantile=0.5`. The reason why `lts.quantile=0.5` is not the default is that the estimator with `lts.quantile=0.9` is more efficient (more precise with less data) *if* the fraction of differentially expressed genes is not that large. See Figure 3.

```
> makeFig("figdiff", 2, 1, {
+   de = (0:6)/10
+   res1 = sim(de = de, nrstrata = 2)
+   res2 = sim(de = de, nrstrata = 2, lts.quantile = 0.5)
+   myPlot2(de, list("de, lts.quantile=0.9" = res1, "de, lts.quantile=0.5" = res2))
+ })
```

In the next code chunk, `up` is the fraction of up-regulated genes among the differentially expressed genes. The best results are obtained for `up` \approx 0.5, while the estimation error becomes larger the more unbalanced the situation becomes.

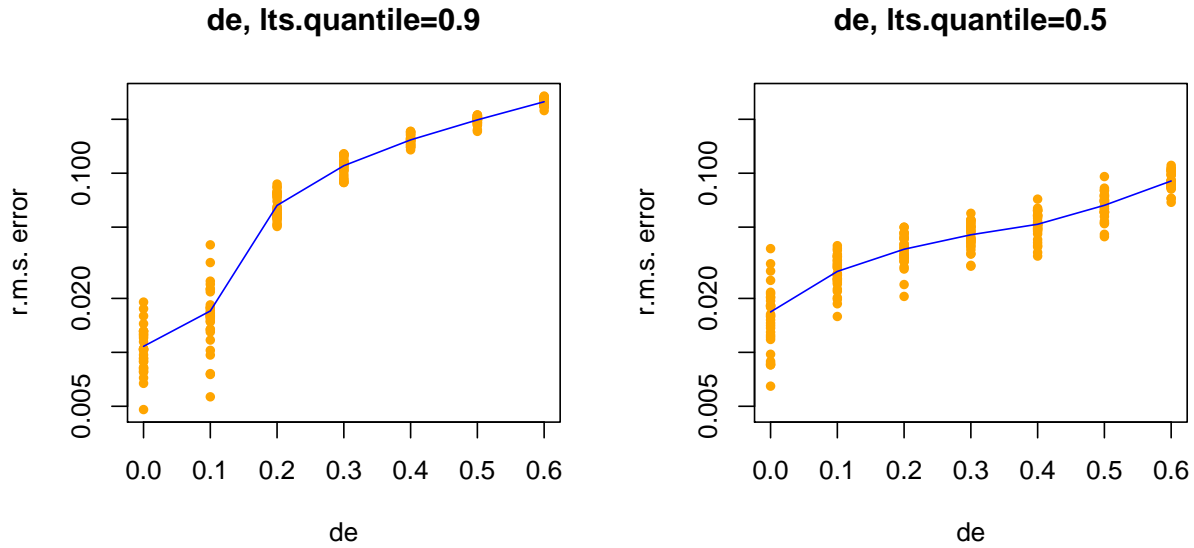


Figure 3: Estimation error as a function of the number of differentially expressed genes, for two different settings of `lts.quantile`. Note how a) is better for small values of `de` (0 and 0.1), but becomes worse for larger values of `de`. See Section 5.

```
> makeFig("figup", 2, 1, {
+   up = (0:8)/8
+   res1 = sim(up = up, nrstrata = 2, de = 0.2)
+   res2 = sim(up = up, nrstrata = 2, de = 0.2, lts.quantile = 0.5)
+   myPlot2(up, list("a) up, lts.quantile=0.9" = res1, "b) up, lts.quantile=0.5" = res2))
+ })
```

6 Missing values

In this Section, we check the impact of missing values on the performance of the estimator. `miss` is the fraction of missing values in the overall

```
> makeFig("figmiss", 2, 1, {
+   miss1 = seq(0, 0.5, length = 6)
+   res1 = sim(miss = miss1, d = 8)
+   miss2 = seq(0, 0.1, length = 6)
+   res2 = sim(miss = miss2, d = 2)
+   myPlot2(list(miss1, miss2), list("fraction NA (d=8)" = res1,
+     "b) fraction NA (d=2)" = res2))
+ })
```

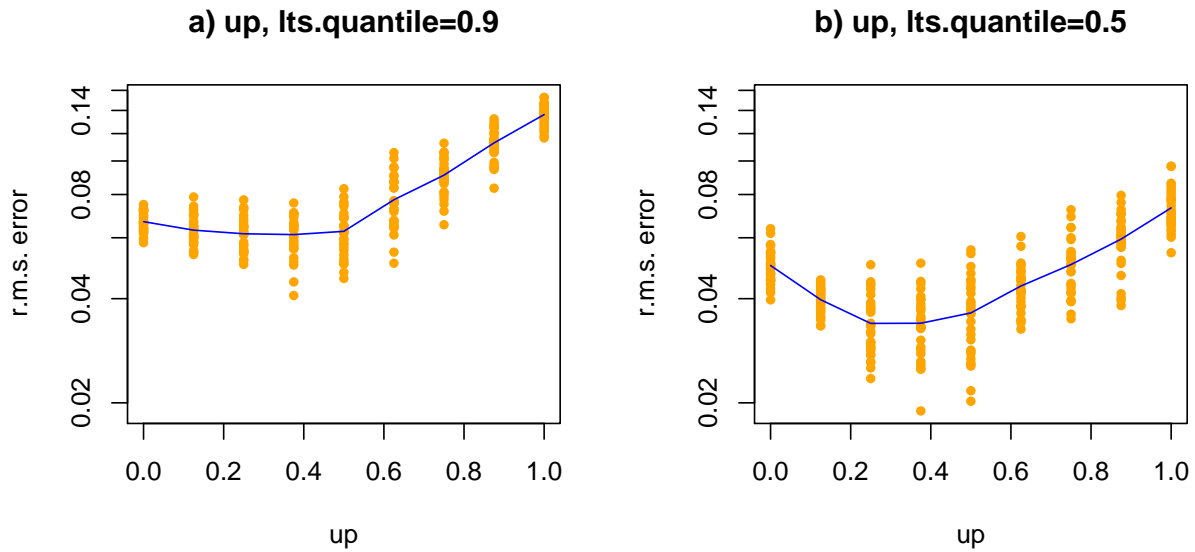


Figure 4: Estimation error as a function of the fraction of up-regulated genes, for two different settings of `lts.quantile`; see Section 5.

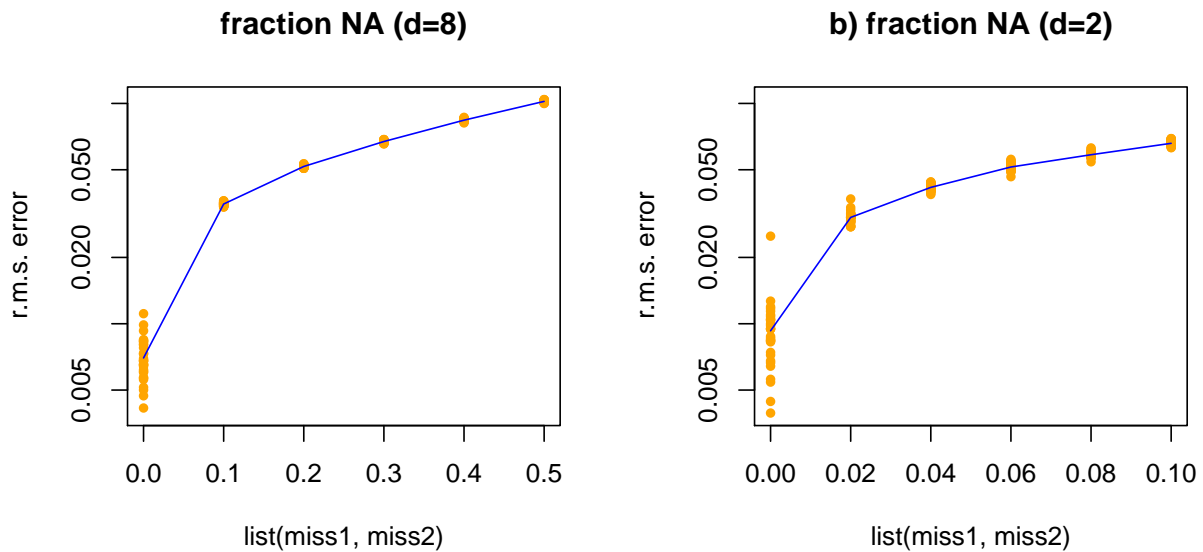


Figure 5: Estimation error as a function of the fraction of missing data points, see Section 6.

7 Incremental normalization

First, let's simulate a dataset with 10000 features, 12 arrays, and no differentially expressed genes (in order to be able to look at the ML estimates rather than their robustified modifications).

```
> dat = sagmbSimulateData(n = 10000, d = 12, de = 0, nrstrata = 1,
+   miss = 0, log2scale = TRUE)
> v = new("vsn", refh = dat$mu, refsigma = dat$sigma, n = length(dat$mu))
> fit = vsn2(dat$y, lts.quantile = 1, verbose = FALSE)
```

`fit` contains the maximum profile likelihood estimate of the *vsn* model. Then we use the *incremental mode* of *vsn* to estimate, in turn for each array individually, the parameters. The results are shown in Figure 6.

```
> parRef = array(as.numeric(NA), dim = dim(fit@par))
> for (j in 1:ncol(dat$y)) {
+   vj = vsn2(dat$y[, j], reference = v, lts.quantile = 1, verbose = FALSE)
+   parRef[, j, ] = vj@par
+ }

> makeFig("figincr", 2, 2, {
+   par(mfcol = c(2, 2))
+   for (k in 1:2) {
+     plot(dat$par[1, , k], parRef[1, , k], pch = 16, xlab = "True",
+         ylab = "Reference fit", main = c("offset", "factor")[k])
+     abline(a = 0, b = 1, col = "orange")
+     plot(fit@par[1, , k], parRef[1, , k], pch = 16, xlab = "Profile Likelihood fit",
+         ylab = "Reference fit", main = c("offset", "factor")[k])
+     abline(a = 0, b = 1, col = "orange")
+   }
+ })
```

8 Session Info

```
> sessionInfo()
```

```
R version 2.5.0 Under development (unstable) (2007-03-18 r40854)
x86_64-unknown-linux-gnu
```

```
locale:
```

```
LC_CTYPE=en_GB.UTF-8;LC_NUMERIC=C;LC_TIME=en_GB.UTF-8;LC_COLLATE=en_GB.UTF-8;LC_MONETARY=en_GB.UTF-8;LC_
```

```
attached base packages:
```

```
[1] "tools"      "stats"      "graphics"   "grDevices"  "utils"      "datasets"
[7] "methods"   "base"
```

```
other attached packages:
```

```
vsna      limma      affy      affyio     Biobase    fortunes
"2.0.29"  "2.9.13"  "1.13.16" "1.3.3"   "1.13.41"  "1.3-2"
```

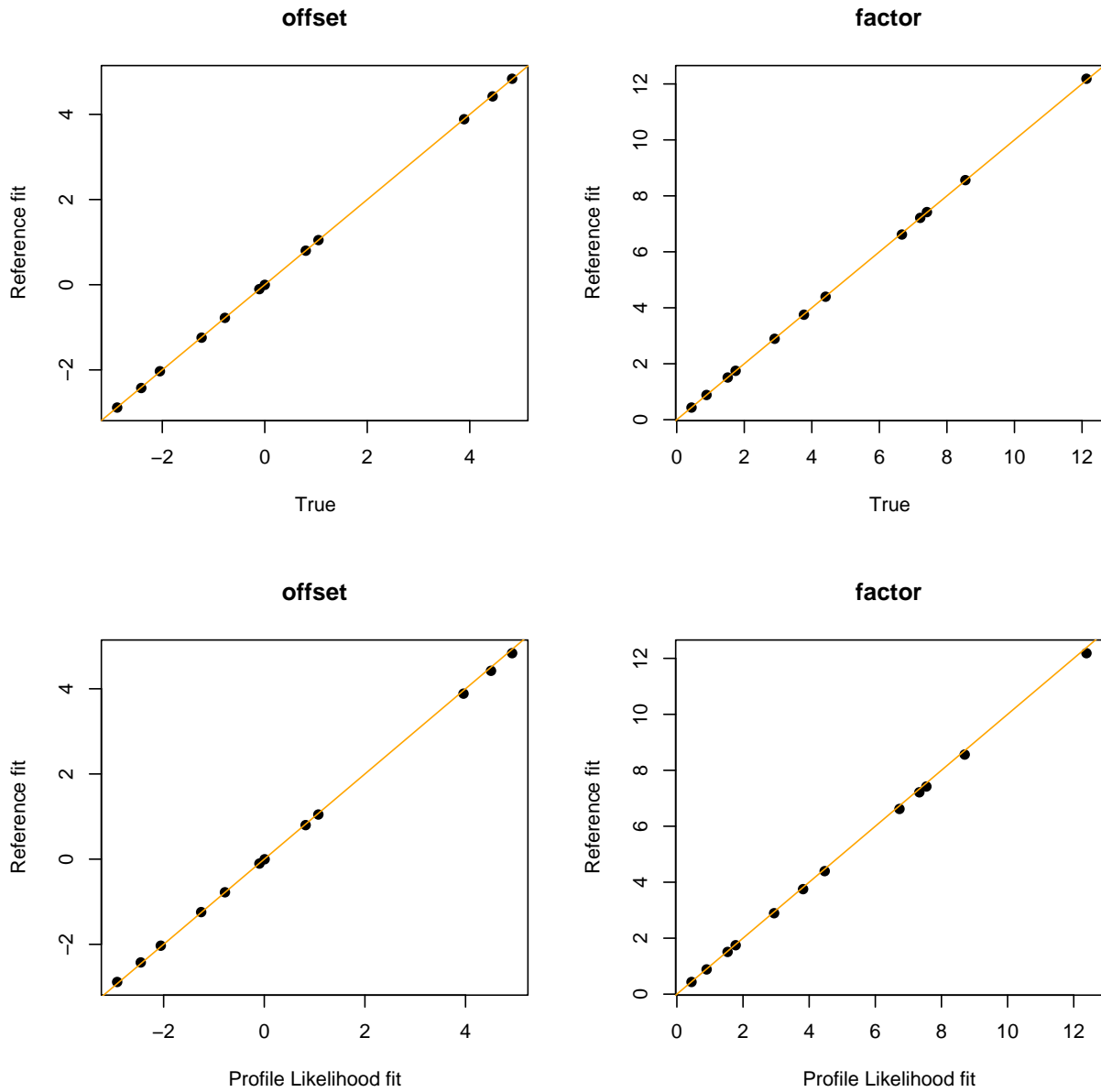


Figure 6: Comparison of parameters fitted from incremental normalisation (y -axis) with true parameters (x -axis, upper row) and with parameters fitted from joint profile-likelihood normalisation (x -axis, lower row); see Section 7.

References

- [1] W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Parameter estimation for the calibration and variance stabilization of microarray data. *Statistical Applications in Genetics and Molecular Biology*, Vol. 2: No. 1, Article 3, 2003. <http://www.bepress.com/sagmb/vol2/iss1/art3>