

Introduction to robust calibration and variance stabilisation with VSN

Wolfgang Huber

May 12, 2008

Contents

1	Getting started	1
2	Running VSN on data from a single two-colour array	2
3	Running VSN on data from multiple arrays (“single colour normalisation”)	3
4	Running VSN on Affymetrix genechip data	4
5	Running VSN on RGList objects	5
5.1	Background subtraction	7
5.2	Print-tip groups	8
6	Missing values	9
7	Normalisation against an existing reference dataset	9
8	The calibration parameters	10
8.1	More on calibration	11
9	Assessing the performance of VSN	11
10	VSN, shrinkage and background correction	13
11	Quality assessment	13
11.1	Stratifying factors such as print-tip, PCR plate, reporter-sequence	13
11.2	Most genes unchanged assumption .	15
12	Acknowledgements	15

1 Getting started

VSN is a method to preprocess microarray intensity data. This can be as simple as

```
> require("vsn")
> data("kidney")
> xnorm = justvsn(kidney)
```

where `kidney` is an *ExpressionSet* object with unnormalised data and `xnorm` the resulting *ExpressionSet* with calibrated and glog_2 -transformed data.

```
> M = exprs(xnorm)[,1] - exprs(xnorm)[,2]
```

produces the vector of generalised log-ratios between the data in the first and second column.

VSN is a model-based method, and the more explicit way of doing the above is

```
> fit = vsn2(kidney)
> ynorm = predict(fit, kidney)
```

where `fit` is an object of class *vsn* that contains the fitted calibration and transformation parameters, and the method `predict` applies the fit to the data. The two-step protocol is useful when you want to fit the parameters on a subset of the data, e. g. a set of control or spike-in features, and then apply the model to the complete set of data. Furthermore, it allows further inspection of the `fit` object, e. g. for the purpose of quality assessment.

Besides *ExpressionSets*, there are also `justvsn` methods for *AffyBatch* objects from the *affy* package and *RGList* objects from the *limma* package. They are described in this vignette.

The so-called glog_2 (short for *generalised logarithm*) is a function that is like the logarithm (base 2) for large values (large compared to the amplitude of the background noise), but is less steep for smaller values. Differences between the transformed values are the *generalised log-ratios*. These are shrinkage estimators of the logarithm of the fold change. The usual *log-ratio* is another example for an estimator¹ of log fold change. There is also

¹In statistics, the term *estimator* is used to denote an algorithm that calculates a value from a set of measured data.

a close relationship between background correction of the intensities and the variance properties of the different estimators. Please see Section 10 for more explanation of these issues.

How does VSN work? In short, each column is calibrated by an affine transformation², then the whole data are transformed by a glog_2 variance-stabilising transformation. After this, systematic array- or dye-biases should be removed, and the variance should be approximately independent of the mean intensity. Many common hypothesis testing, clustering or classification schemes are based on an assumption of common variance. If this assumption does not hold, such methods can produce misleading results; a transformation which makes this assumption roughly true makes the use of such methods more appropriate.

Note, however, that VSN only addresses the dependence of the variance on the mean intensity. There may be other factors influencing the variance, such as gene-inherent properties or changes of the tightness of transcriptional control in different conditions. These need to be addressed by other methods.

2 Running VSN on data from a single two-colour array

The dataset `kidney` contains example data from a spotted cDNA two-colour microarray on which cDNA from two adjacent tissue samples of the same kidney were hybridised, one labeled in green (Cy3), one in red (Cy5). The two columns of the matrix `exprs(kidney)` contain the green and red intensities, respectively. A local background estimate³ was calculated by the image analysis software and subtracted, hence some of the intensities in `kidney` are close to zero or negative. In Figure 1 you can see the scatterplot of the calibrated

This value is intended to correspond to a parameter of the underlying distribution that generated the data. Depending on the amount of the available data and the quality of the estimator, the correspondence between the estimated value and the true value may be more or less faithful. For example, the arithmetic mean and the median are two different estimators of the center of a symmetric distribution.

²It is possible to stratify the transformations within columns, see the `strata` parameter of the function `vsn2`.

³See Section 10 for more on the relationship between background correction and variance stabilising transformations.

and transformed data. For comparison, the scatterplot of the log-transformed raw intensities is also shown.

```
> par(mfrow=c(1,2))
> select = (0==rowSums(exprs(kidney)<=0))
> plot(log2(exprs(kidney)[select, ]),
+   main = "a) raw", pch = ".", asp=1)
> plot(exprs(xnorm), main = "b) vsn",
+   pch = ".", asp=1,
+   col=ifelse(select, "black", "orange"))
```

To verify the variance stabilisation, there is the function `meanSdPlot`. For each feature $k = 1, \dots, n$ it shows the empirical standard deviation $\hat{\sigma}_k$ on the y -axis versus the rank of the average $\hat{\mu}_k$ on the x -axis.

$$\hat{\mu}_k = \frac{1}{d} \sum_{i=1}^d h_{ki} \quad \hat{\sigma}_k^2 = \frac{1}{d-1} \sum_{i=1}^d (h_{ki} - \hat{\mu}_k)^2 \quad (1)$$

```
> par(mfrow=c(1,2))
> meanSdPlot(xnorm, ranks=TRUE)
> meanSdPlot(xnorm, ranks=FALSE)
```

The two plots are shown in Figure 2. The red dots, connected by lines, show the running median of the standard deviation⁴. The aim of these plots is to see whether there is a systematic trend in the standard deviation of the data as a function of overall expression. The assumption that underlies the usefulness of these plots is that most genes are not differentially expressed, so that the running median is a reasonable estimator of the standard deviation of feature level data conditional on the mean. After variance stabilisation, this should be approximately a horizontal line. It may have some random fluctuations, but should not show an overall trend. If this is not the case, that usually indicates a data quality problem, or is a consequence of inadequate prior data preprocessing. The rank ordering distributes the data evenly along the x -axis. A plot in which the x -axis shows the average intensities themselves is obtained by calling the `plot` command with the argument `ranks=FALSE`; but this is less effective in assessing variance and hence is not the default.

⁴The parameters used were: window width 10%, window midpoints 5%, 10%, 15%, It should be said that the proper way to do is with quantile regression such as provided by the `quantreg` package - what is done here for these plots is simple, cheap and should usually be good enough due to the abundance of data.

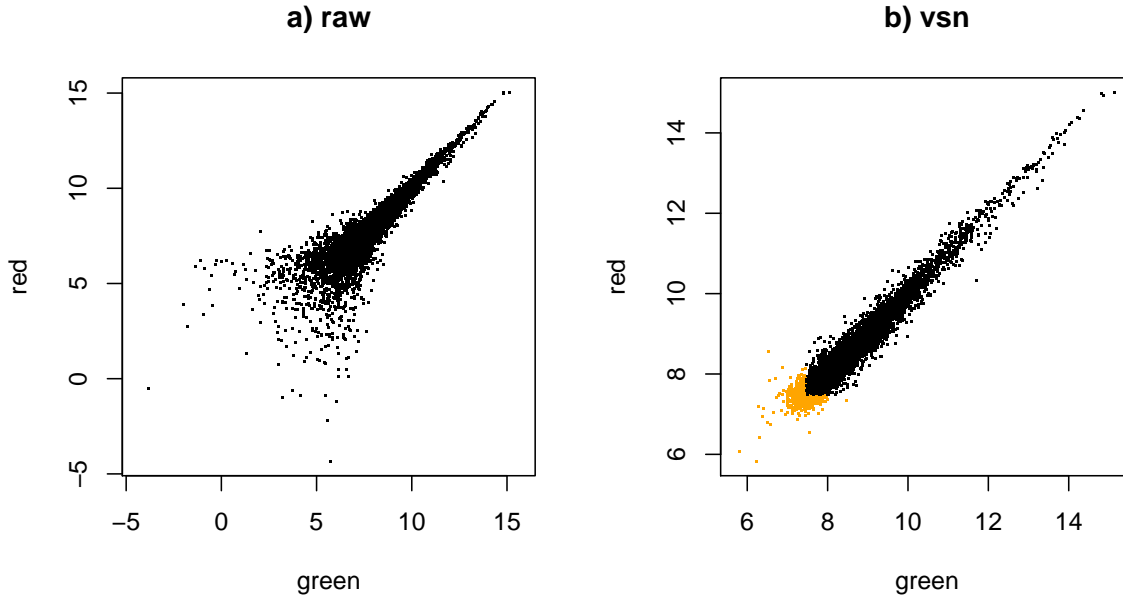


Figure 1: Scatterplots of the kidney example data, which were obtained from a two-colour cDNA array by quantitating spots and subtracting a local background estimate. a) unnormalised and \log_2 -transformed. b) normalised and transformed with VSN, Panel b shows the data from the complete set of 8704 spots on the array, panel a only the 7806 spots for which both red and green net intensities were greater than 0. Those spots which are missing in panel a are coloured in orange in panel b.

The histogramme of the generalized log-ratios is shown in Figure 3.

3 Running VSN on data from multiple arrays (“single colour normalisation”)

The package includes example data from a series of 8 spotted cDNA arrays on which cDNA samples from different lymphoma were hybridised together with a reference cDNA [7].

```
> data("lymphoma")
> dim(lymphoma)
```

```
Features  Samples
    9216      16
```

The 16 columns of the `lymphoma` object contain the red and green intensities, respectively, from the 8 slides, as shown in Table 1. Thus, the CH1 intensities are in columns 1, 3, ..., 15, the CH2 intensities in columns 2, 4, ..., 16. We can call `justvsn` on all of them at once:

```
> lym = justvsn(lymphoma)
> meanSdPlot(lym)
```

Again, Figure 4 helps to visually verify that the variance stabilisation worked. As above, we can obtain the generalised log-ratios for each slide by subtracting the common reference intensities from those for the 8 samples:

```
> iref = seq(1, 15, by=2)
> ismp = seq(2, 16, by=2)
> M= exprs(lym)[,ismp]-exprs(lym)[,iref]
> A=(exprs(lym)[,ismp]+exprs(lym)[,iref])/2
> colnames(M) = lymphoma$sample[ismp]
> colnames(A) = colnames(M)
> require("geneplotter")
> j = "DLCL-0032"
> smoothScatter(A[,j], M[,j], main=j,
+               xlab="A", ylab="M", pch=".")
> abline(h=0, col="red")
```

Figure 5 shows the resulting *M-A*-plot [6] for one of the arrays.

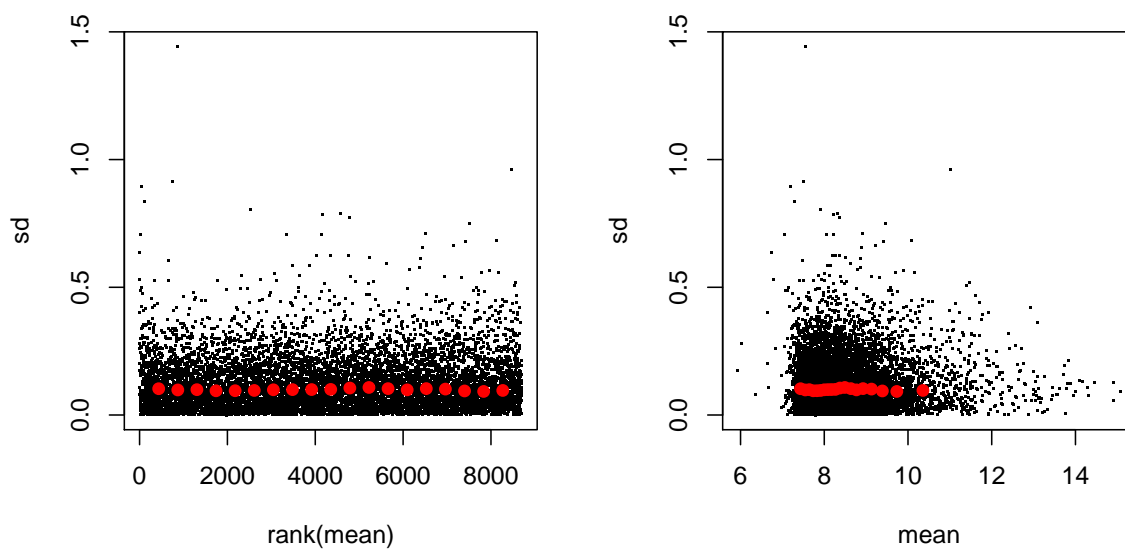


Figure 2: Standard deviation versus rank of the mean, and the mean, respectively.

4 Running VSN on Affymetrix genechip data

The package *affy* provides excellent functionality for reading and processing Affymetrix genechip data, and you are encouraged to refer to the documentation of the package *affy* for more information about data structures and methodology.

The preprocessing of Affymetrix genechip data involves the following steps: (i) background correction, (ii) between-array normalization, (iii) transformation and (iv) summarisation. The VSN method addresses steps (i)–(iii). For the summarisation, I recommend to use the RMA method [10], and a simple wrapper that provides all of these is provided through the method `vsnrma`.

```
> require("affydata")
> data("Dilution")
> d.vsn = vsnrma(Dilution)
```

For comparison, we also run `rma`.

```
> d.rma = rma(Dilution)
```

The resulting scatterplots are shown and compared in Figure 6.

```
> par(mfrow=c(2,2), pch=".")
> w = c(1,3)
```

```
> titvsn = "vsn: chip 1 vs 3"
> titrma = "rma: chip 1 vs 3"
> ax = c(2, 16)
> plot(exprs(d.vsn)[,w], main=titvsn,
+      asp=1, xlim=ax, ylim=ax)
> plot(exprs(d.rma)[,w], main=titrma,
+      asp=1, xlim=ax, ylim=ax)
> plot(exprs(d.rma)[,1], exprs(d.vsn)[,1],
+      xlab="rma", ylab="vsn",
+      asp=1, xlim=ax, ylim=ax,
+      main="chip 1: expression values")
> abline(a=0, b=1, col="red")
> plot(diff(t(exprs(d.rma)[,w])),
+      diff(t(exprs(d.vsn)[,w])),
+      xlab="rma", ylab="vsn", asp=1,
+      main="M values chip 1 vs 3",
+      xlim=c(-1,1), ylim=c(-1,1))
> abline(a=0, b=1, col="red")
```

Both methods control the variance at low intensities, but we see that VSN does so more strongly. The shrinkage (see also Section 10) is stronger with VSN (cf. the bottom right plot), which also leads to a lower dynamic range, cf. the bottom left plot.

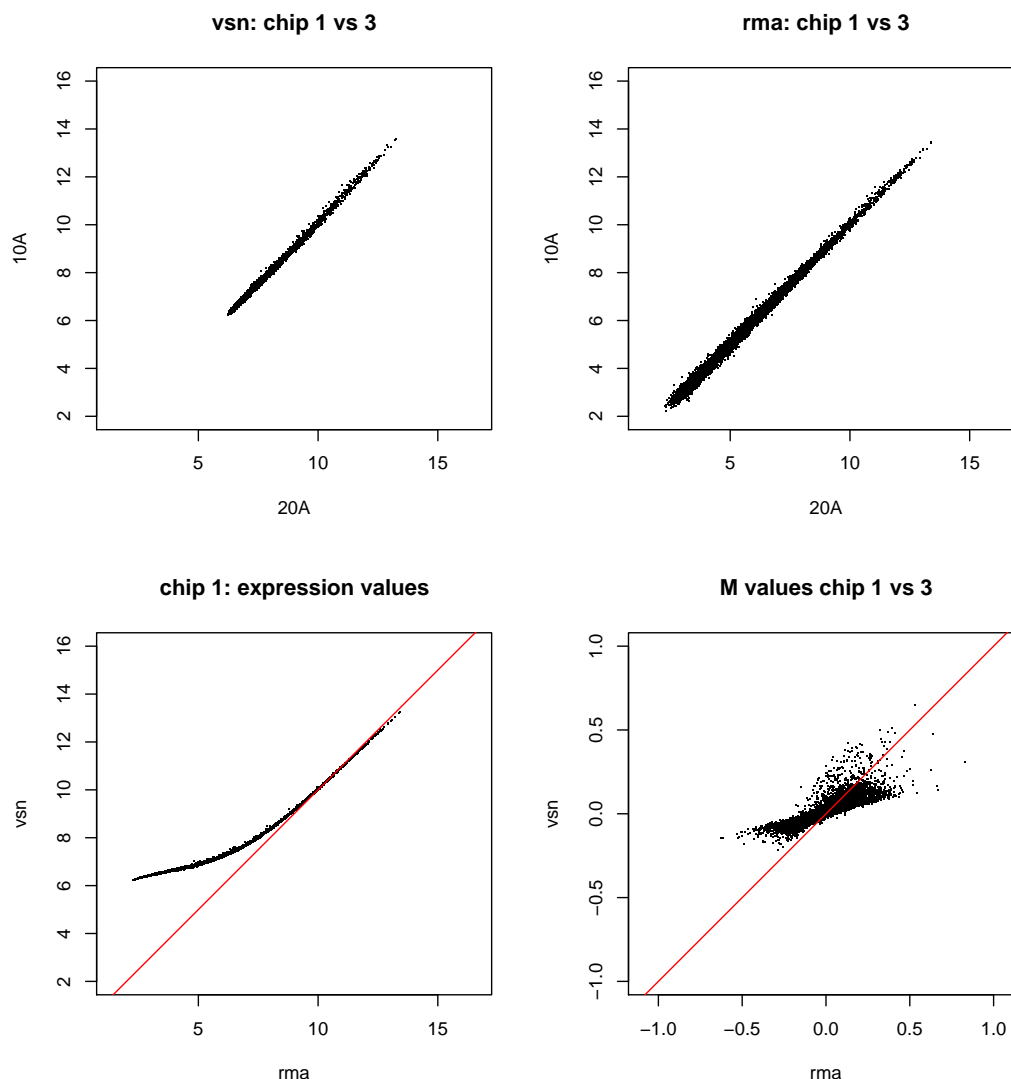


Figure 6: Results of `vsnrma` and `rma` on the Dilution example data. Chip 1 was hybridised with 20 μg RNA from liver, Chip 3 with 10 μg of the same RNA.

5 Running VSN on RGList objects

There is a `justvsn` method for *RGList* objects. Usually, you will produce an *RGList* from your own data using the `read.maimages` from the *limma* package. Here, for the sake of demonstration, we construct an *RGList* from `lymphoma`.

```
> require("limma")
> wg = which(lymphoma$dye=="Cy3")
> wr = which(lymphoma$dye=="Cy5")
> lymRG = new("RGList", list(
+   R=exprs(lymphoma)[, wr],
```

```
+   G=exprs(lymphoma)[, wg]))
> lymNCS = justvsn(lymRG)
```

The `justvsn` method for *RGList* converts its argument into an *NChannelSet*, using a copy of the coercion method from Martin Morgan in the package *convert*. It then passes this on to the `justvsn` method for *NChannelSet*. The return value is an *NChannelSet*, shown in Table 2. Note that, due to the flexibility in the amount and quality of metadata that is in an *RGList*, and due to differences in the implementation of these classes, the transfer of the metadata into the *NChannelSet* may not always produce the expected results, and that some

```
> lymNCS
```

```
NChannelSet (storageMode: lockedEnvironment)
assayData: 9216 features, 8 samples
  element names: G, R
phenoData
  sampleNames: lc7b047.reference, lc7b048.reference, ..., lc7b058.reference (8 total)
  varLabels and varMetadata description: none
featureData
  featureNames: 1, 2, ..., 9216 (9216 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation:
```

Table 2: The *NChannelSet* object lymNCS.

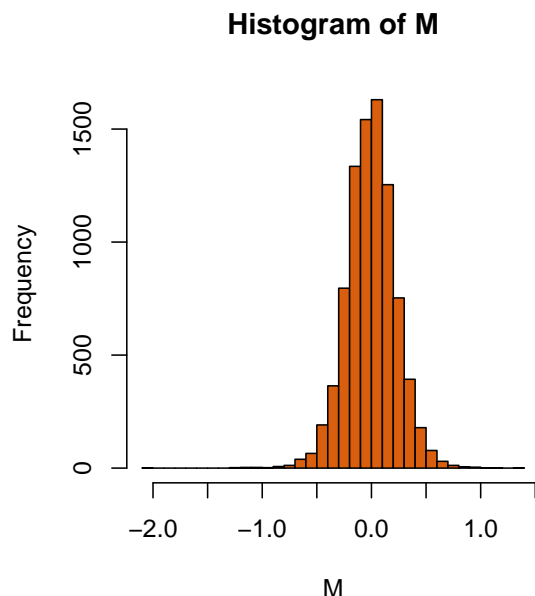


Figure 3: Histogramme of generalised log-ratios *M* for the kidney example data.

checking and often further dataset-specific postprocessing of the sample metadata and the array feature annotation is needed. For the current example, we construct the *AnnotatedDataFrame* object *adf* and assign it into the *phenoData* slot of *lymNCS*.

```
> vmd = data.frame(
+   labelDescription=I(c("array ID",
+     "sample in G", "sample in R")),
+   channel=c("_ALL", "G", "R"),
```

```
> pData(lymphoma)
```

	name	sample	dye
lc7b047.reference	lc7b047	reference	Cy3
lc7b047.CLL-13	lc7b047	CLL-13	Cy5
lc7b048.reference	lc7b048	reference	Cy3
lc7b048.CLL-13	lc7b048	CLL-13	Cy5
lc7b069.reference	lc7b069	reference	Cy3
lc7b069.CLL-52	lc7b069	CLL-52	Cy5
lc7b070.reference	lc7b070	reference	Cy3
lc7b070.CLL-39	lc7b070	CLL-39	Cy5
lc7b019.reference	lc7b019	reference	Cy3
lc7b019.DLCL-0032	lc7b019	DLCL-0032	Cy5
lc7b056.reference	lc7b056	reference	Cy3
lc7b056.DLCL-0024	lc7b056	DLCL-0024	Cy5
lc7b057.reference	lc7b057	reference	Cy3
lc7b057.DLCL-0029	lc7b057	DLCL-0029	Cy5
lc7b058.reference	lc7b058	reference	Cy3
lc7b058.DLCL-0023	lc7b058	DLCL-0023	Cy5

Table 1: The *phenoData* of the lymphoma dataset.

```
+   row.names=c("arrayID", "sampG", "sampR"))
> arrayID = lymphoma$name[wr]
> stopifnot(identical(arrayID, lymphoma$name[wg]))
> ## remove sample number suffix
> sampleType = factor(sub("-.*", "",
+   lymphoma$sample))
> v = data.frame(
+   arrayID = arrayID,
+   sampG = sampleType[wg],
+   sampR = sampleType[wr])
> v
```

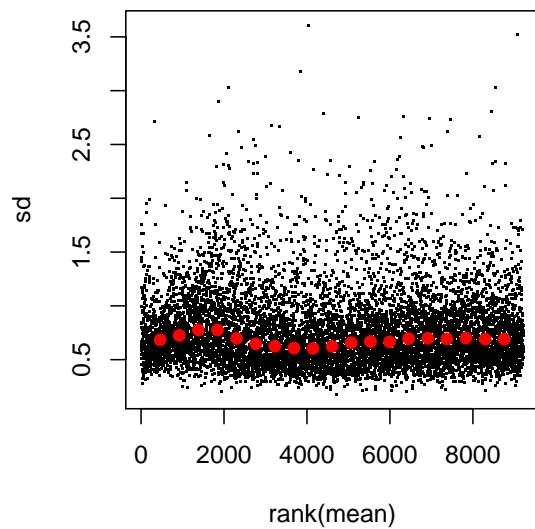


Figure 4: Standard deviation versus rank of the mean for the lymphoma example data

```

arrayID      sampG sampR
1 lc7b047 reference  CLL
2 lc7b048 reference  CLL
3 lc7b069 reference  CLL
4 lc7b070 reference  CLL
5 lc7b019 reference  DLCL
6 lc7b056 reference  DLCL
7 lc7b057 reference  DLCL
8 lc7b058 reference  DLCL

```

```

> adf = new("AnnotatedDataFrame",
+   data=v,
+   varMetadata=vmd)
> phenoData(lymNCS) = adf

```

Now let's combine the red and green values from each array into the log-ratio M and use the linear modeling tools from *limma* to find differentially expressed genes (note that it is often suboptimal to only consider M , and that taking into account absolute intensities as well can improve analyses).

```

> lymM = (assayData(lymNCS)$R -
+   assayData(lymNCS)$G)

> design = model.matrix(~ lymNCS$sampR)
> lf = lmFit(lymM, design[, 2, drop=FALSE])
> lf = eBayes(lf)

```

DLCL-0032

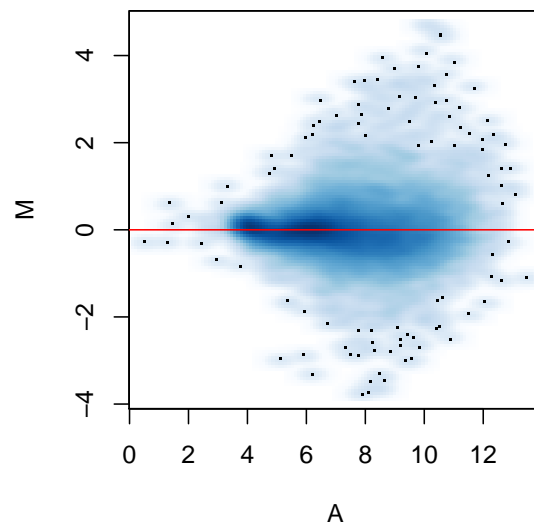


Figure 5: M - A -plot for slide DLCL-0032 from the lymphoma example data. A false-colour representation of the data point density is used, in addition the 100 data points in the least dense regions are plotted as dots.

Figure 7 shows the resulting p -values and the expression profiles of the genes corresponding to the top 5 features.

```

> par(mfrow=c(1,2))
> hist(lf$p.value, 100, col="orange")
> pdat=t(lymM[order(lf$p.value)[1:5],])
> matplot(pdat,
+   lty=1, type="b", lwd=2,
+   col=HSV(seq(0,1,length=5), 0.7, 0.8),
+   ylab="M", xlab="arrays")

```

5.1 Background subtraction

Many image analysis programmes for microarrays provide local background estimates, which are typically calculated from the fluorescence signal outside, but next to the features. These are not always useful. Just as with any measurement, these local background estimates are also subject to random measurement error, and subtracting them from the foreground intensities will lead to increased random noise in the signal. On the other hand side, doing so may remove systematic artifactual drifts

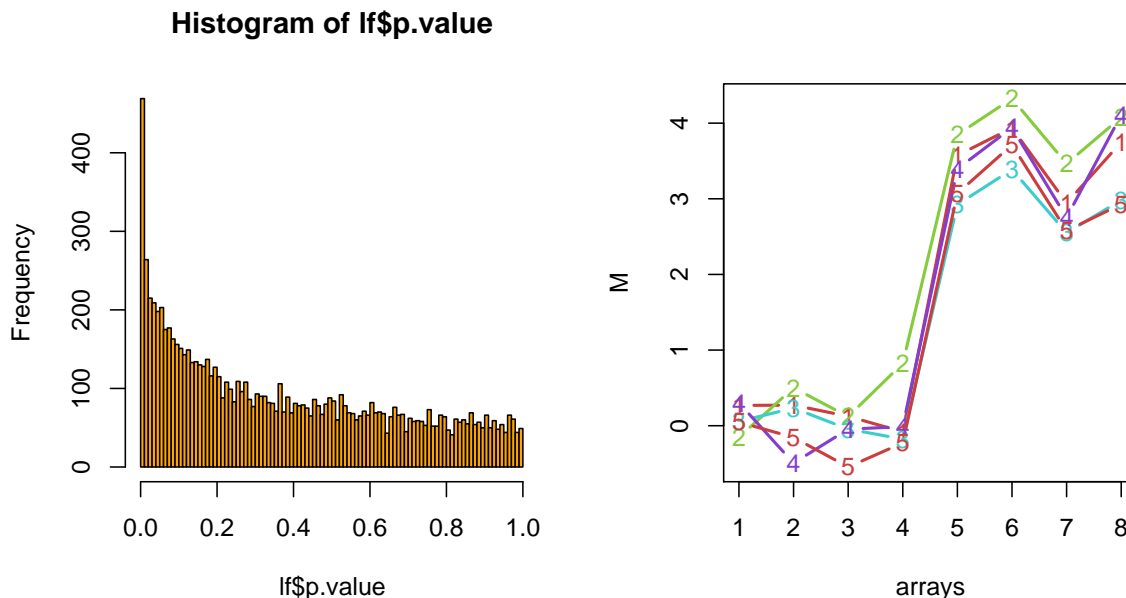


Figure 7: Left: histogramme of p -values from the moderated t -test between the CLL and DLCL groups on the `lymM` values. Right: M -values for the 5 genes with the smallest p -values.

in the data, for example, a spatial gradient.

So what is the optimal analysis strategy, should you subtract local background estimates or not? The answer depends on the properties of your particular data. VSN itself estimates and subtracts an over-all background estimate (per array and colour, see Section 8), so an additional local background correction is only useful if there actually is local variability across an array, for example, a spatial gradient.

Supposing that you have decided to subtract the local background estimates, how is it done? When called with the argument `backgroundsubtract=TRUE`⁵, the `justvsns` method will subtract local background estimates in the `Rb` and `Gb` slots of the incoming *RGList*. To demonstrate this, we construct an *RGList* object `lymRGwbg`.

```
> rndbg=function(x, off, fac)
+   array(off+fac*runif(prod(dim(x))),
+         dim=dim(x))
> lymRGwbg = lymRG
> lymRGwbg$Rb = rndbg(lymRG, 100, 30)
> lymRGwbg$Gb = rndbg(lymRG, 50, 20)
```

In practice, of course, these values will be read from

⁵Note that the default value for this parameter is `FALSE`.

the image quantitation file with a function such as `read.maimages` that produces the *RGList* object. We can call `justvsns`

```
> lymESwbg = justvsns(lymRGwbg[, 1:3],
+   backgroundsubtract=TRUE)
```

Here we only do this for the first 3 arrays to save compute time.

5.2 Print-tip groups

By default, VSN computes one normalisation transformation with a common set of parameters for all features of an array (separately for each colour if it is a multi-colour microarray), see Section 8. Sometimes, there is a need for stratification by further variables of the array manufacturing process, for example, print-tip groups (sectors) or microtitre plates. This can be done with the `strata` parameter of `vsns2`.

The example data that comes with the package does not directly provide the information which print-tip each feature was spotted with, but we can easily reconstruct it:

```
> ngr = ngc = 4L
> nsr = nsc = 24L
```

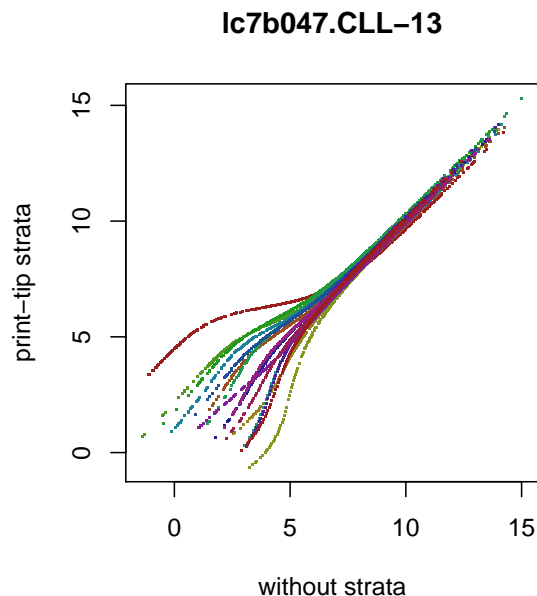



Figure 8: Scatterplot of normalised and transformed intensities for the red channel of array 1. Values on the x -axis correspond to normalisation without strata (`EsenzaStr`), values on the y -axis to normalisation with strata (`EconStr`). The different colours correspond to the 16 different strata.

```
> arrayGeometry = data.frame(
+   spotcol = rep(1:nsc,
+     times = nsr*ngr*ngc),
+   spotrow = rep(1:nsr,
+     each = nsc, times=ngr*ngc),
+   pin = rep(1:(ngr*ngc),
+     each = nsr*nsc))
```

and call

```
> EconStr = justvsnsn(lymRG[,1],
+   strata=arrayGeometry$pin)
```

To save CPU time, we only call this on the first array. We compare the result to calling `justvsnsn` without `strata`,

```
> EsenzaStr = justvsnsn(lymRG[,1])
```

A scatterplot comparing the transformed red intensities, using the two models, is shown in Figure 8.

```
> j = 1L
> plot(assayData(EsenzaStr)$R[,j],
```

```
+   assayData(EconStr)$R[,j],
+   pch = ".", asp = 1,
+   col = hsv(seq(0, 1, length=ngr*ngc),
+     0.8, 0.6)[arrayGeometry$pin],
+   xlab = "without strata",
+   ylab = "print-tip strata",
+   main = sampleNames(lymNCS)$R[j])
```

6 Missing values

The parameter estimation algorithm of VSN is able to deal with missing values in the input data. To demonstrate this, we generate an *ExpressionSet* `lym2` in which about 10% of all intensities are missing,

```
> lym2 = lymphoma
> wh = sample(prod(dim(lym2)), 16000)
> exprs(lym2)[wh] = NA
> table(is.na(exprs(lym2)))
```

```
FALSE  TRUE
131456 16000
```

and call `vsns2` on it.

```
> fit1 = vsns2(lymphoma)
> fit2 = vsns2(lym2)
```

The resulting fitted parameters are not identical, but very similar, see Figure 9.

```
> par(mfrow=c(1,2))
> for(j in 1:2){
+   p1 = coef(fit1)[,,j]
+   p2 = coef(fit2)[,,j]
+   d = max(abs(p1-p2))
+   stopifnot(d<(2/j))
+   plot(p1, p2, pch = 16, asp = 1,
+     main = paste(letters[j],
+       ": max.diff =", signif(d,2), sep = ""),
+     xlab = "no missing data",
+     ylab = "10% of data missing")
+   abline(a = 0, b = 1, col = "blue")
+ }
```

7 Normalisation against an existing reference dataset

So far, we have considered the joint normalisation of a set of arrays to each other. What happens if,

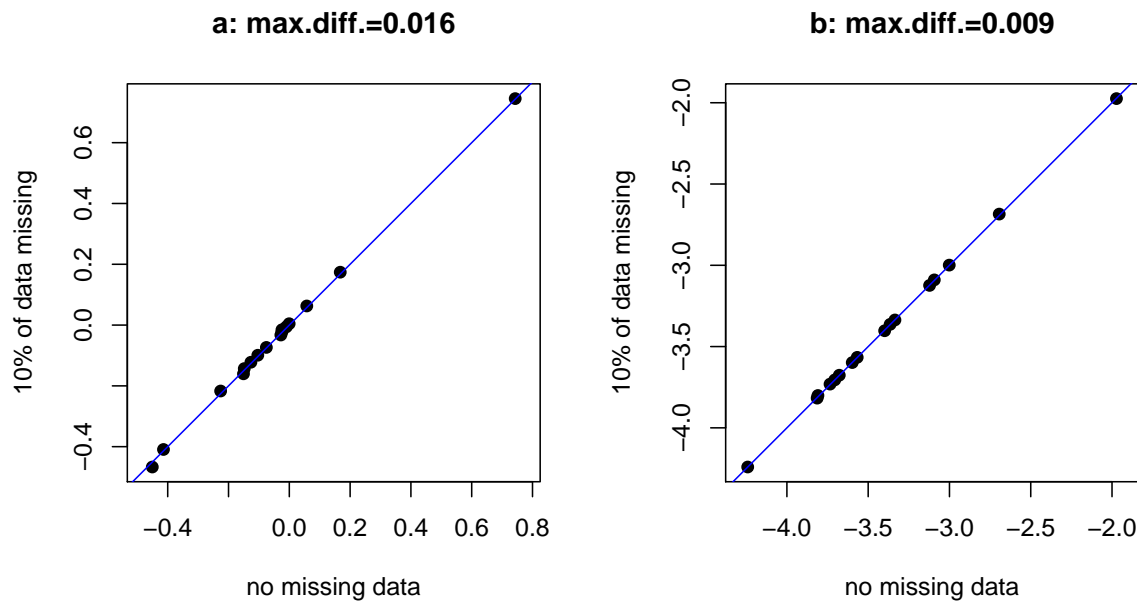


Figure 9: Scatterplots of fitted parameters, values on the x -axis correspond to normalisation without missing data (`fit1`), values on the y -axis to normalisation with $\approx 10\%$ missing data (`fit2`).

after analysing a set of arrays in this fashion, we obtain some additional arrays? Do we re-run the whole normalisation again for the complete, new and bigger set of arrays? This may sometimes be impractical.

Suppose we have used a set of training arrays for setting up a classifier that is able to discriminate different biological states of the samples based on their mRNA profile. Now we get new test arrays to which we want to apply the classifier. Clearly, we do not want to re-run the normalisation for the whole, new and bigger dataset, as this would change the training data; neither can we normalise only the test arrays among themselves, without normalising them “towards” the reference training dataset. What we need is a normalisation procedure that normalises the new test arrays “towards” the existing reference dataset without changing the latter.

To simulate this situation with the available example data, pretend that the Cy5 channels of the lymphoma dataset can be treated as 8 single-colour arrays, and fit a model to the first 7.

```
> ref = vsn2(lymphoma[, ismp[1:7]])
```

Now we call `vsn2` on the 8-th array, with the output

from the previous call as the reference.

```
> f8 = vsn2(lymphoma[, ismp[8]],
+          reference = ref)
```

We can compare this to what we get if we fit the model to all 8 arrays,

```
> fall = vsn2(lymphoma[, ismp])
```

```
> coefficients(f8)[1,]
```

```
[1] -0.134 -3.456
```

```
> coefficients(fall)[8,]
```

```
[1] -0.242 -3.428
```

and compare the resulting values in the scatterplot Figure 10: they are very similar.

```
> plot(exprs(f8), exprs(fall)[8,],
+       pch=".", asp=1)
> abline(a=0, b=1, col="red")
```

More details on this can be found in the vignettes *Verifying and assessing the performance with simulated data* and *Likelihood Calculations for vsn* that come with this package.

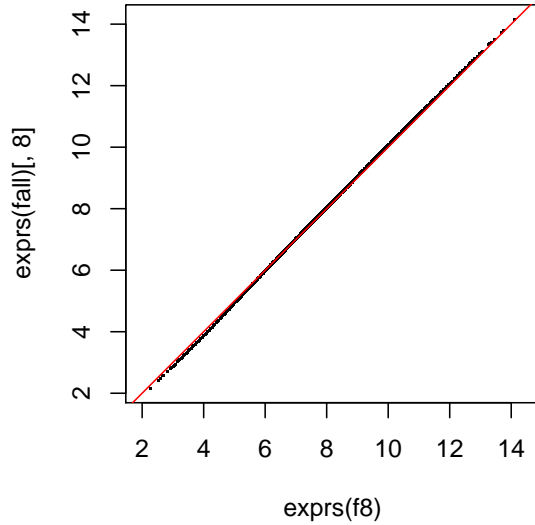


Figure 10: Scatterplot of normalised intensities after normalisation by reference (x -axis, `f8`) and joint normalisation (y -axis, `fall`). There is good agreement.

8 The calibration parameters

If y_{ki} is the matrix of uncalibrated data, with k indexing the rows and i the columns, then the calibrated data y'_{ki} is obtained through scaling by λ_{si} and shifting by α_{si} :

$$y'_{ki} = \lambda_{si} y_{ki} + \alpha_{si} \quad (2)$$

where $s \equiv s(k)$ is the so-called *stratum* for feature k . In the simplest case, there is only one stratum, i.e. the index s is always equal to 1, or may be omitted altogether. This amounts to assuming that the data of all features on an array were subject to the same systematic effects, such that an array-wide calibration is sufficient.

A model with multiple strata per array may be useful for spotted arrays. For these, stratification may be according to print-tip [6] or PCR-plate [2]. For oligonucleotide arrays, it may be useful to stratify the features by physico-chemical properties, e.g. to assume that features of different sequence composition attract systematically different levels of unspecific background signal.

The transformation to a scale where the variance

of the data is approximately independent of the mean is

$$\begin{aligned} h_{ki} &= \operatorname{arsinh}(\lambda_0 y'_{ki} + \alpha_0) \\ &= \log \left(\lambda_0 y'_{ki} + \alpha_0 + \sqrt{(\lambda_0 y'_{ki} + \alpha_0)^2 + 1} \right), \end{aligned} \quad (3)$$

with two parameters λ_0 and α_0 . Equations (2) and (3) can be combined, so that the whole transformation is given by

$$h_{ki} = \operatorname{arsinh} \left(e^{b_{si}} \cdot y_{ki} + a_{si} \right). \quad (4)$$

Here, $a_{si} = \alpha_{si} + \lambda_0 \alpha_{si}$ and $b_{si} = \log(\lambda_0 \lambda_{si})$ are the combined calibration and transformation parameters for features from stratum s and sample i . Using the parameter b_{si} as defined here rather than $e^{b_{si}}$ appears to make the numerical optimisation more reliable (less ill-conditioned).

We can access the calibration and transformation parameters through

```
> coef(fit)[1,,]

      [,1] [,2]
[1,] -0.0844 -5.93
[2,] -0.0679 -5.96
```

For a dataset with d samples and s strata, `coef(fit)` is a numeric array with dimensions $(s, d, 2)$. For the example data that was used in Section 1 to generate `fit`, $d = 2$ and $s = 1$. `coef(fit)[s, i, 1]`, the first line in the results of the above code chunk, is what was called a_{si} in Eqn. (4), and `coef(fit)[s, i, 2]`, the second line, is b_{si} .

8.1 More on calibration

Now suppose the kidney example data were not that well measured, and the red channel had a baseline that was shifted by 500 and a scale that differed by a factor of 0.25:

```
> bkid=kidney
> exprs(bkid)[,1]=0.25*(500+exprs(bkid)[,1])
```

We can again call `vsn2` on these data

```
> bfit <- vsn2(bkid)
```

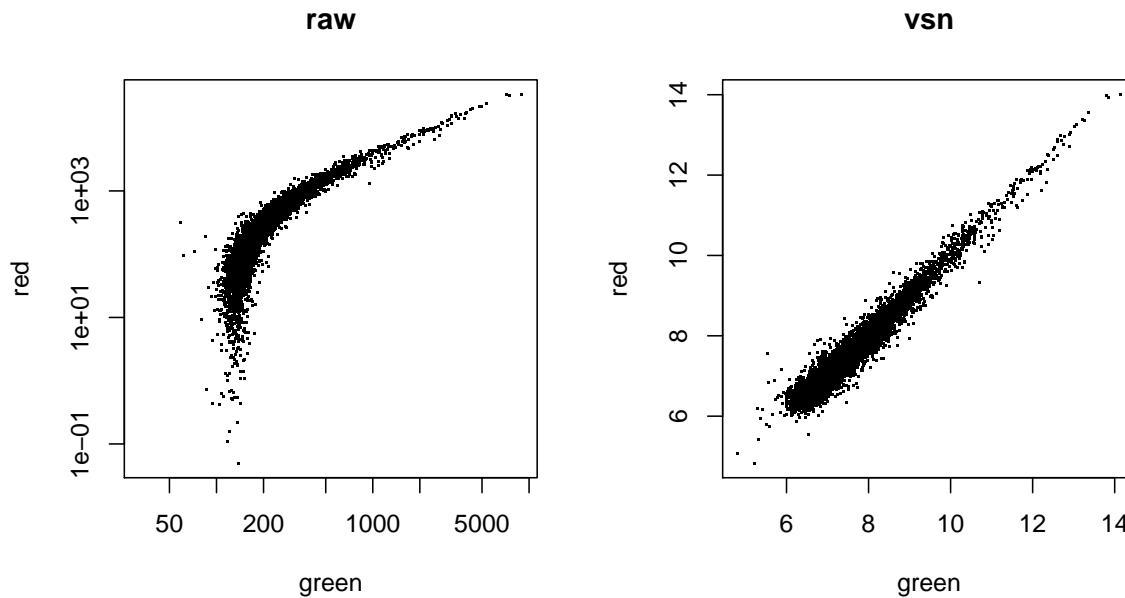


Figure 11: Scatterplots for badly biased data. Left hand side: raw data on log-log scale, right hand side: after calibration and transformation with vsn.

```
> par(mfrow=c(1,2))
> plot(exprs(bkid), main="raw",
+       pch=".", log="xy")
> plot(exprs(bfit), main="vsn",
+       pch=".")
> coef(bfit)[1,,]

      [,1] [,2]
[1,] -1.4140 -4.54
[2,] -0.0641 -5.96
```

Notice the change in the parameter b of the red channel: it is now larger by about $\log(4) \approx 1.4$, and the shift parameter a has also been adjusted. The result is shown in Figure 11.

9 Assessing the performance of VSN

VSN is a parameter estimation algorithm that fits the parameters for a certain model. In order to see how good the estimator is, we can look at bias, variance, sample size dependence, robustness against model misspecification and outliers. This is done in the vignette *Verifying and assessing the performance with simulated data* that comes with this package.

Practically, the more interesting question is how different microarray calibration and data transformation methods compare to each other. Two such comparisons were made in reference [1], one with a set of two-colour cDNA arrays, one with an Affymetrix genechip dataset. Fold-change estimates from VSN led to higher sensitivity and specificity in identifying differentially expressed genes than a number of other methods.

A much more sophisticated and wider-scoped approach was taken by the *Affycomp* benchmark study, presented at <http://affycomp.biostat.jhsph.edu>. It uses two benchmark datasets: a *Spike-In* dataset, in which a small number of cDNAs was spiked in at known concentrations and over a wide range of concentrations on top of a complex RNA background sample; and a *Dilution* dataset, in which RNA samples from heart and brain were combined in a number of dilutions and proportions. The design of the benchmark study, which has been open for anyone to submit their method, was described in reference [8]. A discussion of its results was given in reference [9]. One of the results that emerged was that VSN compares well with the background correction and quantile normalization

method of RMA; both methods place a high emphasis on *precision* of the expression estimate, at the price of a certain *bias* (see also Section 10). Another result was that reporter-sequence specific effects (e.g. the effect of GC content) play a large role in these data and that substantial improvements can be achieved when they are taken into account (something which VSN does not do).

Of course, the two datasets that were used in Affycomp were somewhat artificial: they had fewer differentially expressed genes and were probably of higher quality than in most real-life applications. And, naturally, in the meanwhile the existence of this benchmark has led to the development of new processing methods where a certain amount of overfitting may have occurred.

I would also like to note the interaction between normalization/preprocessing and data quality. For data of high quality, one can argue that any decent preprocessing method should produce more or less the same results; differences arise when the data are problematic, and when more or less successful measures may be taken by preprocessing methods to correct these problems.

10 VSN, shrinkage and background correction

Generalised log-ratios can be viewed as a *shrinkage estimator*: for low intensities either in the numerator and denominator, they are smaller in absolute value than the standard log-ratios, whereas for large intensities, they become equal. Their advantage is that they do not suffer from the variance divergence of the standard log-ratios at small intensities: they remain well-defined and have limited variance when the data come close to zero or even become negative. An illustration is shown in Figure 12. Please consult the references for more on the mathematical background [1–3].

It is possible to give a *Bayesian* interpretation: our prior assumption is the conservative one of no differential expression. Evidence from a feature with high overall intensity is taken strongly, and the posterior results in an estimate close to the empirical intensity ratio. Evidence from features with low intensity is downweighted, and the posterior is still strongly influenced by the prior.

11 Quality assessment

Quality problems can often be associated with physical parameters of the manufacturing or experimental process. Let us look a bit closer at the `lymphoma` data. Recall that `M` is the 9216 times 8 matrix of generalized log-ratios and `A` a matrix of the same size with the average \log_2 -transformed intensities. The dataframe `arrayGeometry` (from Section 5.2) contains, for each array feature, the identifier of the print-tip by which it was spotted and the row and column within the print-tip sector. Figure 13 shows the boxplots of `A` values of array `CLL-13` stratified by row.

```
> colours = hsv(seq(0,1,length=nsr),0.6,1)
> j = "CLL-13"
> boxplot(A[, j] ~ arrayGeometry$spotrow,
+         col=colours, main=j,
+         ylab="A", xlab="spotrow")
```

You may want to explore similar boxplots for other stratifying factors such as column within print-tip sector or print-tip sector and look at these plots for the other arrays as well.

In Figure 13, we see that the features in rows 22 and 23 are all very dim. If we now look at these data in the *M-A*-plot (Figure 14), we see that these features not only have low *A*-values, but fall systematically away from the $M = 0$ line.

```
> plot(A[,j], M[,j], pch=16, cex=0.3,
+      col=ifelse(arrayGeometry$spotrow%in%(22:23),
+                "orange", "black"))
> abline(h=0, col="blue")
```

Hence, in a naive analysis the data from these features would be interpreted as contributing evidence for differential expression, while they are more likely just the result of a quality problem. So what can we do? There are some options:

1. Flag the data of the affected features as unreliable and set them aside from the subsequent analysis.
2. Use a more complex, stratified normalisation method that takes into account the different row behaviours, for example, VSN with strata (see Section 5.2).
3. It has also been proposed to address this type of problem by using a non-linear regression on

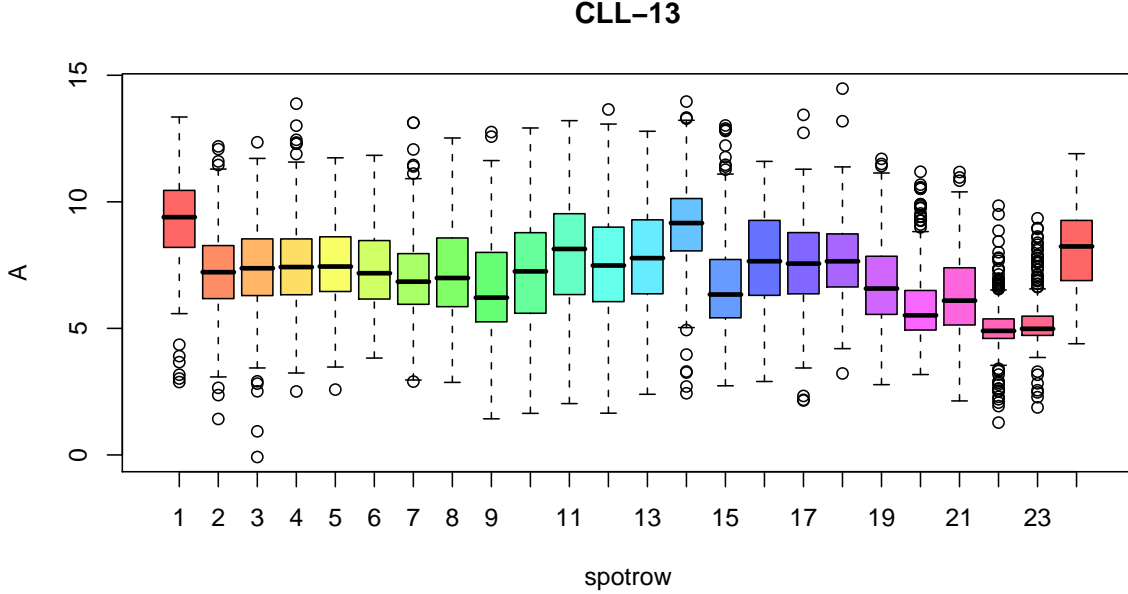


Figure 13: Boxplot of A values of array CLL-13 stratified by within-sector row. The features in rows 22 and 23 are all very dim.

the A -values, for example the *loess* normalization of reference [11] that simply squeezes the M - A -plot to force the centre of the distribution of M to lie at 0 along the whole A -range.

An advantage of option 3 is that it works without knowing the real underlying stratifying factor. However, it assumes that the stratifying factor is strongly confounded with A , and that biases that it causes can be removed through a regression on A .

In the current example, if we believe that the real underlying stratifying factor is indeed row within sector, this assumption means that (i) few of the data points from rows 22 and 23 have high A -values, and that (ii) almost all data points with very low A values are from these rows; while (i) appears tenable, (ii) is definitely not the case.

11.1 Stratifying factors such as print-tip, PCR plate, reporter-sequence

By default, the VSN method assumes that the measured signal y_{ik} increases, to sufficient approximation, proportionally to the mRNA abundance c_{ik} of gene k on the i -th array, or on the i -th colour

channel:

$$y_{ik} \approx \alpha_i + \lambda_i \lambda_k c_{ik}. \quad (5)$$

For a series of d single-colour arrays, $i = 1, \dots, d$, and the different factors λ_i reflect the different initial amounts of sample mRNA or different overall reverse transcription, hybridisation and detection efficiencies. The feature affinity λ_k contains factors that affect all measurements with feature k in the same manner, such as sequence-specific labelling efficiency. The λ_k are assumed to be the same across all arrays. There can be a non-zero overall offset α_i . For a two-colour cDNA array, $i = 1, 2$, and the λ_i take into account the different overall efficiencies of the two dyes⁶.

Equation 5 can be generalised to

$$y_{ik} \approx \alpha_{is} + \lambda_{is} \lambda_k c_{ik}. \quad (6)$$

⁶It has been reported that for some genes the dye bias is different from gene to gene, such that the proportionality factor does not simply factorise as in (5). As long as this only occurs sporadically, this will not have much effect on the estimation of the calibration and variance stabilisation parameters. Further, by using an appropriate experimental design such as colour-swap or reference design, the effects of gene-specific dye biases on subsequent analyses can be reduced.

that is, the background term α_{is} and the gain factor λ_{is} can be different for different groups s of features on an array. The VSN methods allows for this option by using the `strata` argument of the function `vs2`. We have seen an example above where this could be useful. For Affymetrix genechips, one can find systematic dependences of the affinities λ_{is} and the background terms α_{is} on the reporter sequence, however, the optimal stratification of reporters based on their sequence is an active area of research.

Nevertheless, there are situations in which either assumption (5) or (6) is violated, and these include:

Saturation. The biochemical reactions and/or the photodetection can be run in such a manner that saturation effects occur. It may be possible to rescue such data by using non-linear transformations. Alternatively, it is recommended that the experimental parameters are chosen to avoid saturation.

Batch effects. The feature affinities λ_k may differ between different manufacturing batches of arrays due, e.g., to different qualities of DNA amplification or printing. VSN cannot be used to simultaneously calibrate and transform data from different batches.

How to reliably diagnose and deal with such violations is beyond the scope of this vignette; see the references for more [2, 6].

11.2 Most genes unchanged assumption

With respect to the VSN model fitting, data from differentially transcribed genes can act as outliers (but they do not necessarily need to do so in all cases). The maximal number of outliers that do not gravely affect the model fitting is controlled by the parameter `lts.quantile`. Its default value is 0.9, which allows for 10% outliers. The value of `lts.quantile` can be reduced down to 0.5, which allows for up to 50% outliers. The maximal value is 1, which results in a least-sum-of-squares estimation that does not allow for any outliers.

So why is this parameter `lts.quantile` user-definable and why don't we just always use the most "robust" value of 0.5? The answer is that the *precision* of the estimated VSN parameters is

better the more data points go into the estimates, and this may especially be an issue for arrays with a small number of features⁷. So if you are confident that the number of outliers is not that large, using a high value of `lts.quantile` can be justified.

There has been a little bit of confusion on the role of the "most genes unchanged assumption", which presumes that only a minority of genes on the arrays is detectably differentially transcribed across the experiments. This assumption is a *sufficient* condition for there being only a small number of outliers, and these would not gravely affect the VSN model parameter estimation. However, it is not a *necessary* condition: the parameter estimates and the resulting normalised data may still be useful if the assumption does not hold, but if the effects of the data from differentially transcribed genes balance out.

12 Acknowledgements

I acknowledge helpful comments and feedback from Anja von Heydebreck, Richard Bourgon, Martin Vingron, Ulrich Mansmann and Robert Gentleman.

References

- [1] W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to quantification of differential expression. *Bioinformatics*, 18:S96–S104, 2002.
- [2] W. Huber, A. von Heydebreck, and M. Vingron. Analysis of microarray gene expression data. *Handbook of Statistical Genetics*, Eds.: D. J. Balding, M. Bishop, C. Cannings. John Wiley & Sons, Inc. (2003). (*preprint available from author*)
- [3] 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>

⁷more precisely, number of features per stratum


```
> toLatex(sessionInfo())
```

- R version 2.7.0 (2008-04-22), i386-pc-mingw32
- Locale: LC_COLLATE=English_United States.1252;LC_CTYPE=English_United States.1252;LC_MONETARY=E
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: affy 1.18.0, affydata 1.11.3, affyio 1.8.0, annotate 1.18.0, AnnotationDbi 1.2.0, Biobase 2.0.1, DBI 0.2-4, geneplotter 1.18.0, hgu95av2cdf 2.2.0, lattice 0.17-7, limma 2.14.1, preprocessCore 1.2.0, RSQLite 0.6-8, vsn 3.6.0, xtable 1.5-2
- Loaded via a namespace (and not attached): grid 2.7.0, KernSmooth 2.22-22, RColorBrewer 1.0-2

Table 3: The output of `sessionInfo` on the build system after running this vignette.

- | | |
|---|---|
| <p>[4] W. Huber, A. von Heydebreck, and M. Vingron. Error models for microarray intensities. <i>Encyclopedia of Genomics, Proteomics and Bioinformatics</i>, John Wiley & Sons, Inc. (2004). (preprint available from author)</p> <p>[5] David M. Rocke and Blythe Durbin. A model for measurement error for gene expression analysis. <i>Journal of Computational Biology</i>, 8:557–569, 2001.</p> <p>[6] S. Dudoit, Y. H. Yang, T. P. Speed, and M. J. Callow. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. <i>Statistica Sinica</i>, 12:111–139, 2002.</p> <p>[7] A. A. Alizadeh et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. <i>Nature</i>, 403:503–511, 2000.</p> <p>[8] L. M. Cope, R. A. Irizarry, H. A. Jaffee, Z. Wu, and T. P. Speed. A Benchmark for Affymetrix GeneChip Expression Measures. <i>Bioinformatics</i>, 20:323–331, 2004.</p> <p>[9] R. A. Irizarry, Z. Wu, and H. A. Jaffee. Comparison of Affymetrix GeneChip expression measures. <i>Bioinformatics</i>, 22:789–794, 2006.</p> <p>[10] R. A. Irizarry, B. Hobbs, F. Collin, Y. D. Beazer-Barclay, K. J. Antonellis, U. Scherf, and T. P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. <i>Biostatistics</i> 4:249–264, 2003.</p> <p>[11] Y. H. Yang, S. Dudoit, P. Luu, and T. P. Speed. Normalization for cDNA microarray</p> | <p>data: a robust composite method addressing single and multiple slide systematic variation. <i>Nucleic Acids Research</i> 30:e15, 2002.</p> |
|---|---|

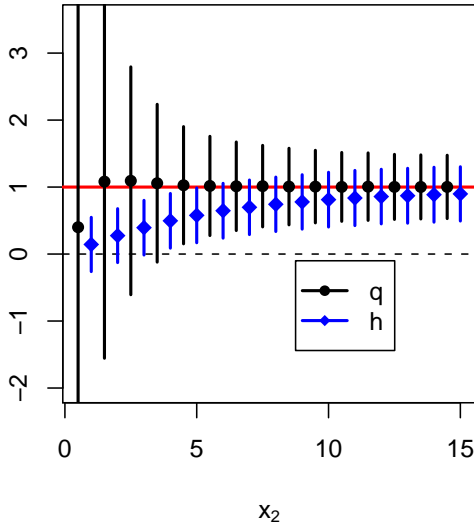


Figure 12: The shrinkage property of the generalised log-ratio. Blue diamonds and error bars correspond to mean and standard deviation of the generalised log-ratio h , as obtained from VSN, and black dots and error bars to the standard log-ratio q (both base 2). For this figure, data were generated from the additive-multiplicative error model [3–5]. The horizontal line corresponds to the true \log_2 -ratio 1 (corresponding to a factor of 2). For intensities x_2 that are larger than about ten times the additive noise level σ_a , generalised log-ratio h and standard log-ratio q coincide. For smaller intensities, we can see a *variance-bias trade-off*: q has no bias but a huge variance, thus an estimate of the fold change based on a limited set of data can be arbitrarily off. In contrast, h keeps a constant variance – at the price of systematically underestimating the true fold change.

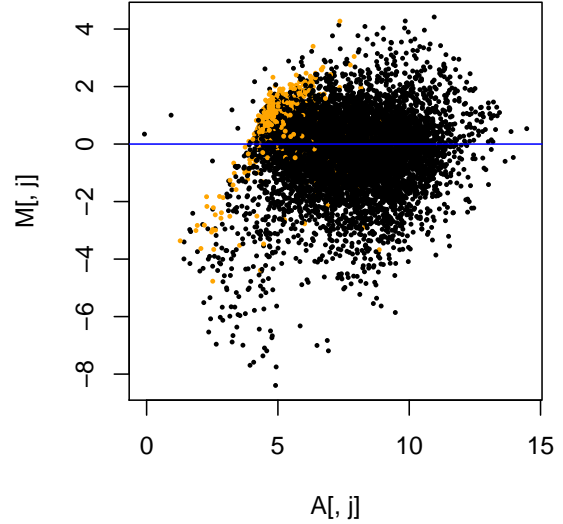


Figure 14: M - A -plot of the data from array CLL-13. Dots coloured in orange are from rows 22 and 23. A possible explanation may be as follows (although I do not know for sure that this is the right explanation): The PCR product (cDNA) that is spotted on these arrays is put on by a print head that sucks cDNA out of microtitre plates and deposits them in spots one after another, row by row. If the content of one plate is faulty, this results in a set of subsequent spots that are faulty. Because the 16 print-tip sectors are spotted in parallel, this affects all sectors in the same way.