Bioconductor's snm package

Brig Mecham and John D. Storey
email: brig.mecham@sagebase.org and jstorey@princeton.edu

September 19, 2011

Contents

1 Overview

This document provides a tutorial for using the snm package. The package consists of a two main functions, snm, for normalizing data based on a set of study-specific variables, and sva, for modeling latent variables. Many questions about these functions will hopefully be answered by the documentation or references. As with any R package, detailed information on functions, their arguments and values, can be obtained in the help files. To view the help file for the function snm within R, type ?snm. If you identify bugs related to basic usage please contact the authors directly. Otherwise, any questions or problems regarding snm will most efficiently be addressed on

the Bioconductor mailing list, `http://stat.ethz.ch/mailman/listinfo/bioconductor`.

The remainder of this document is organized as follows. Section 2 provides a brief introduction to the concepts and ideas that underlie this software package. In section 3 we provide three examples intended to demonstrate how to use the `snm` package to normalize microarray data. Section 5.4 contains a description of how to diagnose any problems with the normalized data returned from an `snm` function call using a simple diagnostic plot. In section 3.1 we cover some of the `snm` options that can be set for more control. Finally, in section 6 we provide details on the simulated data sets explored in section three.

2 Introduction

The `snm` package contains functions for implementing supervised normalization of microrarray data. Within any given study, we identify and model the sources of systematic variation with study-specific variables, which are driven by the study design. These study-specific variables fall into one of two categories: biological variables or adjustment variables. We define biological variables to be those whose relationships with nucleic acid variation are the target of the analysis. The other variables, which arise from complexities of the biological or experimental paradigm from which the data are generated, are what we call adjustment variables. SNM is designed to model and remove the effects of adjustment variables on the data without influencing the biological variables of interest. This algorithm was described in detail in [2].

The complete data from a microarray experiment consists of three terms: the observed probe intensities, biological variables, and adjustment variables. The intensities are usually presented as an $m \times n$ matrix, $\mathbf{Y}$, where m and n describe the number of probes and arrays in the entire study, respectively. Define y_{ij} as the observed intensity for probe $i = 1, ..., m$ on array $j = 1, .., n$, and $\mathbf{y}_i = (y_{i1}, y_{i2}, ..., y_{in})$ as the observed intensities for probe i across the n arrays. The kth biological variable for array j, x_{kj}, describes factors of interest such as disease status, experimental treatment, or time point. All d covariates for an individual sample j are denoted by the vector $\mathbf{x}_j$, and we group all n such vectors into a $d \times n$ matrix $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)$.

Similarly, the ℓth adjustment variable for array j, $z_{\ell j}$, parameterizes variables to be fit in a probe-specific manner. Let r_c denote the number of probe-specific adjustment variables. When the baseline value of y_{ij} is not of interest, this probe-specific intercept term is included among the adjustment variables. Define the vector $\mathbf{z}_j$ and matrix $\mathbf{Z}$ in a fashion analogous to the one described for $\mathbf{x}_j$ and $\mathbf{X}$, respectfully. Finally, we assume that there are r_f intensity dependent effects, which we denote by f_{tj} for effect $t = 1, \ldots, r_f$ and array j.

Note that we construct $\mathbf{X}$ and $\mathbf{Z}$ such that $\mathbf{b}_i = \mathbf{0}$ represents the case where the biological variables of interest show no association with probe i. (This is straightforward to construct even for time course studies [3].) Concrete examples of $\mathbf{X}$ and $\mathbf{Z}$ can be found in the examples below.

We model y_{ij} as linear combinations of $\mathbf{x}_j$, $\mathbf{z}_j$, and intensity-dependent effects. The model for each probe intensity measurement is written as:

$$y_{ij} = \sum_{k=1}^{d} b_{ik} x_{kj} + \sum_{\ell=1}^{r_c} a_{i\ell} z_{\ell j} + \sum_{t=1}^{r_f} f_{tj}\left(m_{ij}\right) + e_{ij}, \tag{1}$$

where $m_{ij} = \sum_{k=1}^{d} b_{ik}x_{kj} + \sum_{\ell=1}^{r_c} a_{i\ell}z_{\ell j}$. The coefficients b_{ik} and $a_{i\ell}$ describe the influence of the k-th biological and ℓ-th adjustment variable on probe i's intensity. We assume that the t-th intensity dependent function f_{tj} is a random smooth function such that $\mathrm{E}[f_{tj}(m)|m] = 0$ for all m. These are parameterized as Normal distributed coefficients applied to a B-spline basis.

We can write model (1) for probe i data across all n arrays, $\mathbf{y}_i$, as:

$$\mathbf{y}_i = \mathbf{b}_i\mathbf{X} + \mathbf{a}_i\mathbf{Z} + \sum_{t=1}^{r_f} \boldsymbol{f}_t(\mathbf{b}_i\mathbf{X} + \mathbf{a}_i\mathbf{Z}) + \mathbf{e}_i, \tag{2}$$

where $\mathbf{b}_i$ and $\mathbf{a}_i$ are $1 \times d$ and $1 \times r_c$ vectors of the b_{ik} and $a_{i\ell}$ terms in (1), and $\boldsymbol{f}_t(\mathbf{b}_i\mathbf{X} + \mathbf{a}_i\mathbf{Z}) = (f_{t1}(m_{i1}), \ldots, f_{tn}(m_{in}))$. The model for the entire data set $\mathbf{Y}$ can be written as:

$$\mathbf{Y} = \mathbf{B}\mathbf{X} + \mathbf{A}\mathbf{Z} + \sum_{t=1}^{r_f} \boldsymbol{f}_t(\mathbf{B}\mathbf{X} + \mathbf{A}\mathbf{Z}) + \mathbf{E}. \tag{3}$$

where $\mathbf{B}$ and $\mathbf{A}$ are $m \times d$ and $m \times r_c$ matrices of coefficients, the i-th row corresponding to $\mathbf{b}_i$ and $\mathbf{a}_i$, respectfully. Also $\boldsymbol{f}_t(\mathbf{B}\mathbf{X}+\mathbf{A}\mathbf{Z})$ is an $m \times n$ matrix with the i-th row equal to $\boldsymbol{f}_t(\mathbf{b}_i\mathbf{X}+\mathbf{a}_i\mathbf{Z})$.

The algorithm to fit this model is described and evaluated in reference [2].

3 Basic Usage

In this section we provide a simple introduction to the `snm` function. To do so we make use of simulated examples designed to mimic commonly occuring normalization problems. First, we normalize data from a single channel experiment (e.g., Affymetrix data). Next, we normalize data from a two-color microarray experiment (e.g., Agilent or custom built cDNA microarrays) for both dye swap and reference designs. Finally, we explore normalizing data where the intensity-dependent effects have already been removed. Users interested in removing adjustment variables from data preprocessed with RMA should consult this example.

For these examples we use the default settings for the `snm` function. Doing so produces a model fit diagnostic plot with four panels that is updated at each iteration. See section 5.4 for a description of this diagnostic plot and section 3.1 for details on the different options the `snm` function can accept.

Before using the `snm` functions the library must be loaded:

```
> library(snm)
```

3.1 Input

The following objects are the fundamental inputs to the `snm` function:

raw.dat: A probes (rows) by arrays (columns) matrix of unnormalized, probe level data. Required.

bio.var: A model matrix of the biological variable of interest. Required.

adj.var: A model matrix of the adjustment variables.

int.var: A data frame of the intensity-dependent adjustment variables, one effect per column.

The number of rows of `bio.var`, `adj.var`, and `int.var` should equal the total number of arrays, i.e., the number of columns of `raw.dat` If `bio.var` contains an intercept, it is removed. If `adj.var=NULL`, then an intercept is added automatically. It is possible not include `int.var` as input, in which case it is treated as `int.var=NULL` and no intensity dependent effects are fit (see examples below).

The following options are also available:

weights: In certain instances the analyst might be aware of a set of probes that should be ignored when estimating intensity-dependent effects. For example, probes influenced by latent structure, surface artifacts, or some other source of variation that is not included in the model. The `weights` option is designed for this situation. By simply setting the weight for these probes to 0 the analyst can force the algorithm to ignore these values when estimating intensity-dependent effects.

spline.dim: The degrees of freedom of the spline utilized for intensity-dependent effects.

nbins: The number of bins formed when fitting intensity-dependent effects. This is essentially for computational speed.

num.iter: The number of iterations to perform. This is utilized rather than a numerical convergence criterion.

rm.adj: This option is used to control whether or not the probe-specific adjustment variables should be removed from the normalized data returned by the `snm` function call. If the analyst will perform subsequent formal statistical inference (e.g., hypotheses testing), then this option should be set to FALSE. If the analyst will perform clustering or network analysis, then this option should be set to TRUE.

verbose: If set to TRUE, then the iteration number is printed to the R console, and the above plots are displayed for each iteration.

diagnose: If set to TRUE, then diagnostic plots are displayed for each iteration.

3.2 Output

The output of the `snm` function is a list composed of the following elements:

norm.dat: The matrix of normalized data. The default setting is `rm.adj=FALSE`, which means that only the intensity-dependent effects have been subtracted from the data. If the user wants the adjustment variable effects removed as well, then set `rm.adj=TRUE` when calling the `snm` function.

pvalues: A vector of p-values testing the association of the biological variables with each probe. These p-values are obtained from an ANOVA comparing models where the full model contains both the probe-specific biological and adjustment variables versus a reduced model that just contains the probe-specific adjustment variables. The data used for this comparison has the intensity-dependent variables removed. These returned p-values are calculated after the final iteration of the algorithm.

pi0: The estimated proportion of true null probes π_0, also calculated after the final iteration of the algorithm.

iter.pi0s: A vector of length equal to `num.iter` containing the estimated π_0 values at each iteration of the `snm` algorithm. These values should converge and any non-convergence suggests a problem with the data, the assumed model, or both.

nulls: A vector indexing the probes utilized in estimating the intensity-dependent effects on the final iteration.

M: A matrix containing the estimated probe intensities for each array utilized in estimating the intensity-dependent effects on the final iteration. For memory parsimony, only a subset of values spanning the range is returned, currently `nbins` $\times 100$ values.

array.fx: A matrix of the final estimated intensity-dependent array effects. For memory parsimony, only a subset of values spanning the range is returned, currently `nbins` $\times 100$ values.

bio.var: The processed version of the same input variable.

adj.var: The processed version of the same input variable.

int.var: The processed version of the same input variable.

df0: Degrees of freedom of the adjustment variables.

df1: Degrees of freedom of the full model matrix, which includes the biological variables and the adjustment variables.

raw.dat: The input data.

rm.var: Same as the input (useful for later analyses).

call: The `snm` function call.

3.3 Single Channel Normalization

The first simulation is designed to mimic the most commonly assumed models for single channel microarray data such as that provided by Affymetrix. For this scenario we make use of simulated data that is influenced by a single dichotomous biological variable, two probe-specific adjustment variables (one a dichotomous variable designed to simulate a batch effect, and the other a continuous variable designed to simulate an age effect), and intensity-dependent array effects. See section 6.1 for a detailed description of this simulation.

The following commands simulate data for this example and displays the structure of the data object.

```
> singleChannel <- sim.singleChannel(12345)
> str(singleChannel)

List of 5
 $ raw.data  : num [1:25000, 1:50] 4.92 1.909 0.794 7.954 0.684 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:50] "1" "2" "3" "4" ...
 $ bio.var   : num [1:50, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
```

```
  .. ..$ : chr [1:50] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "(Intercept)" "groupsB"
  ..- attr(*, "assign")= int [1:2] 0 1
  ..- attr(*, "contrasts")=List of 1
  .. ..$ groups: chr "contr.treatment"
 $ adj.var   : num [1:50, 1:6] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:50] "1" "2" "3" "4" ...
  .. ..$ : chr [1:6] "(Intercept)" "batchesB" "batchesC" "batchesD" ...
  ..- attr(*, "assign")= int [1:6] 0 1 1 1 1 2
  ..- attr(*, "contrasts")=List of 1
  .. ..$ batches: chr "contr.treatment"
 $ int.var   :'data.frame':        50 obs. of  1 variable:
  ..$ array: Factor w/ 50 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ true.nulls: int [1:17500] 1 2 3 5 6 8 9 11 12 15 ...
```

You can see that the singleChannel object is a list with 5 values. Here is a brief description of each value:

raw.data: A 25,000 by 50 matrix of simulated data

bio.var: A 50 by 2 model matrix of the biological variable of interest.

adj.var: A 50 by 6 model matrix of the adjustment variables

int.var: A 50 by 1 data frame of the intensity-dependent adjustment variables

true.nulls: a vector of indices corresponding to the rows in raw.data of the probes unaffected by the biological variable of interest

We can normalize this data using the following command:

```
> snm.obj1 <- snm(singleChannel$raw.data, singleChannel$bio.var,
+     singleChannel$adj.var, singleChannel$int.var)

 Iteration:  1
 Iteration:  2
 Iteration:  3
 Iteration:  4
 Iteration:  5
 Iteration:  6
 Iteration:  7
 Iteration:  8
 Iteration:  9
 Iteration:  10
```

These simulated data allow us to check the validity of the normalized data. Specifically, we can check the validity of the normalized data by testing whether or not the estimated p-values from the genes unaffected by the biological variables of interest are Uniform(0,1). The following command tests this explicitly:

```
> ks.test(snm.obj1$pval[singleChannel$true.nulls], "punif")
```

```
        One-sample Kolmogorov-Smirnov test

data:  snm.obj1$pval[singleChannel$true.nulls]
D = 0.0064, p-value = 0.4689
alternative hypothesis: two-sided
```

This result suggests the null p-values are Uniform(0,1) distributed, thus satisfying the first criteria of a normalized data set as described in [2]..

3.4 Double Channel Normalization

The second simulation is designed to mimic the assumed model for many double channel microarray data such as that provided by Agilent or a spotted cDNA microarray. For this scenario we make use of simulated data that is influenced by a single dichotomous biological variable, a single continuous probe-specific adjustment variable designed to simulate an age effect, and intensity-dependent array and dye effects. For more details see section 6.2. The following command simulates the data, then fits the model using the default `snm` function settings. See the Single Channel Normalization example above for a more detailed description of this process.

```
doubleChannel <- sim.doubleChannel(12346)
```

The `int.var` variable contains both the intensity-dependent array effects and the intensity-dependent dye effects:

```
print(doubleChannel$int.var)
```

We normalize the data with the following and check the null p-values:

```
snm.obj2 <- snm(doubleChannel$raw.data,
                            doubleChannel$bio.var,
                            doubleChannel$adj.var,
                            doubleChannel$int.var)

ks.test(snm.obj2$pval[doubleChannel$true.nulls],"punif")
```

3.5 Reference Design

The third simulation is designed to mimic studies that utilize a reference design. For this scenario we simulated data that uses a uniform reference to compare expression differences between two conditions. We do not include any probe-specific adjustment variables, but do include intensity-dependent array and dye effects. For more details see section 6.3. The following command simulates the data, then fits the model using the default `snm` function settings. See the Single Channel Normalization example above for a more detailed description of this process.

```
refChannel <- sim.refDesign(12347)
snm.obj3 <- snm(refChannel$raw.data, refChannel$bio.var,
                            refChannel$adj.var, refChannel$int.var)
ks.test(snm.obj3$pval[refChannel$true.nulls],"punif")
```

The above model was fit on the cy3 and cy5 data. Here we show how to fit the model after first subtracting the reference channel from the experimental channel.

```
refChannel$raw.data = refChannel$raw.data[,1:20]-refChannel$raw.data[,21:40]

#adjust the covariates
refChannel$bio.var = refChannel$bio.var[1:20,-3]
refChannel$int.var = data.frame(refChannel$int.var[1:20,1])

snm.obj4 <- snm(refChannel$raw.data,refChannel$bio.var,
                            refChannel$adj.var, refChannel$int.var)
ks.test(snm.obj4$pval[refChannel$true.nulls],"punif")
```

3.6 Preprocessed Normalization

We strongly recommend applying `snm` to the raw data whenever possible. However, sometimes the only available data has already been preprocessed by another method. For example, data from two-color experiments where the only available data are normalized ratios, or data from single channel experiments that has already been processed with RMA. One can still utilize `snm` to account for known sources of variation through the study-specific model. The fourth simulation is designed to mimic the approach in this scenario. The following command simulates the data, then fits the model using the default `snm` function settings. See the Single Channel Normalization example above for a more detailed description of this process.

```
preProcessed <- sim.preProcessed(12347)

snm.obj5 <- snm(preProcessed$raw.data,
                            preProcessed$bio.var,
                            preProcessed$adj.var, rm.adj=TRUE)

ks.test(snm.obj5$pval[preProcessed$true.nulls],"punif")
```

4 Diagnosing the `snm` output

The `snm` methodology requires that the analyst has specified a well behaved study-specific model. When this is not the case the normalized data set can lead to biased inference as shown in [2]. To help identify when the assumed model is incorrect we provide a plot that displays the progression of the model fit over the prescribed iterations, which is activated with the `snm` argument `diagnose=TRUE`. An example plot from an iteration of the single channel normalization from section 3.3 is presented in figure 2.
The `snm` diagnostic plot contains four panels. Here is a brief description of each panel:

(A) Probes Per Bin: To estimate intensity-dependent effects we employ a binning strategy where null probes are grouped based on their estimated RNA concentration. This figure provides counts of the number of probes in each bin. These counts are important for identifying when certains ranges of intensities are poorly represented given the assumed model. The optimal shape of this function is defined by the distribution of average probe intensities. Any large variation suggests there are too few probes unaffected by the biological variable of interest.

(B) Latent Structure: The percent variance explained by each principal component from a principal components analysis of the estimated residual matrix. These values are sorted in descending order and if the first value is noticeably larger than the remaining then the assumed

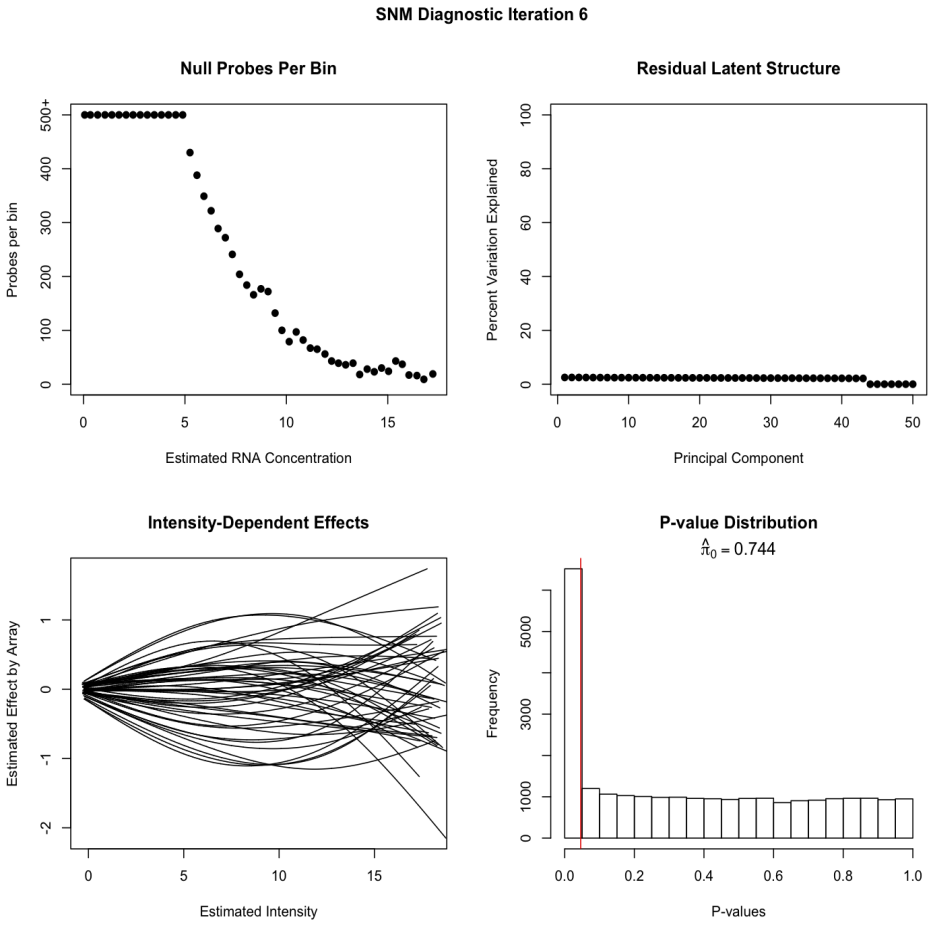

Figure 1: Example Iteration Level Diagnostic Plot (`diagnostic=TRUE`).

model is missing relevant study-specific variables. In this case we recommend the analyst utilize the SVA approach [1].

(C) Estimated Array Effects: The cumulative estimated intensity-dependent effect for each column of the raw data matrix.

(D) P-value Histogram: The histogram of p-values obtained from an ANOVA comparing models where the full model contains both the probe-specific biological and adjustment variables versus a reduced model that just contains the probe-specific adjustment variables. The data used for this comparison has the intensity-dependent variables removed. The subset of probes used to estimate the intensity-dependent effects are those to the right of the vertical red line.

A similar plot of the final model fit can be displayed with the following command:

```
plot(snm.obj1)
```

The intensity-dependent model estimates can be colored according to the study-specific model. Here are two examples:

```
plot(snm.obj1, col.by=snm.obj1$bio.var) #color by biological group
plot(snm.obj1, col.by=snm.obj1$adj.var[,-6]) #color by batch
```

The model fits can be extracted from the `snm` output from the `fitted` function. The model fit under the adjustment variables only ($\sim -1+$ `adj.var`) is returned as the `fit0` matrix in the returned list. Likewise, the model fit under the adjustment variables + biological variables ($\sim -1+$ `adj.var + bio.var`) is returned as the `fit1` matrix in the returned list. This can be explicitly checked:

```
check.fit = fitted(snm.obj1)
rbind(check.fit$fit0[1,1:8], lm(snm.obj1$norm.dat[1,] ~ -1 + snm.obj1$adj.var)$fit[1:8])
rbind(check.fit$fit1[1,1:8], lm(snm.obj1$norm.dat[1,] ~ -1 + snm.obj1$adj.var
                                                + snm.obj1$bio.var)$fit[1:8])
```

The following example shows how these fits can be used to further investigate residual structure.

```
snm.obj <- snm(singleChannel$raw.data,
                           singleChannel$bio.var,
                           singleChannel$adj.var[,-6],
                           singleChannel$int.var, num.iter=4)
snm.fit = fitted(snm.obj)
res1 = snm.obj$norm.dat - snm.fit$fit1
snm.svd = fast.svd(res1)
cor(snm.svd$v[,1], singleChannel$adj.var[,6])
plot(snm.svd$v[,1], singleChannel$adj.var[,6])
```

5 Surrogate Variable Analysis

We include a function called `sva` to model latent variables. This function is distinct from that provided by Jeff Leeks `sva` package in two ways. First, we provide the user with the ability to define the rank of the dependence kernel. Second, we estimate the probabilities using the joint distribution of P values. We also include a useful diagnostic plot that can help identify issues with the estimated dependence kernel.

5.1 Input

The following objects are the fundamental inputs to the `sva` function.

dat: Either an m genes by n arrays matrix of expression data or an object of class edge obtained from a previous sva function call.

bio.var: A model matrix (see 'model.matrix') or data frame with n rows of the biological variables. If NULL, then all probes are treated as "null" in the algorithm.

adj.var: A model matrix (see 'model.matrix') or data frame with n rows of the probe-specific adjustment variables. If NULL, a model with an intercept term is used.

n.sv: Rank of dependence kernel. If equal to NULL (default) this value is estimated from the data.

num.iter: The number of iterations of the algorithm to perform.

diagnose: A flag telling the software whether or not to produce diagnostic output in the form of consecutive plots. TRUE produces the plot.

verbose: A flag telling the software whether or not to display a report after each iteration. TRUE produces the output.

5.2 Output

An object of `class` edge (see the `edge` package). These are lists with the following elements:

dat: m $\times$ n data matrix

n.arrays: number of arrays or columns of dat

n.probes: number of probes or rows of dat

bio.var: A model matrix (see 'model.matrix') or data frame with n rows of the biological variables. If NULL, then all probes are treated as "null" in the algorithm.

adj.var: A model matrix (see 'model.matrix') or data frame with n rows of the probe-specific adjustment variables. If NULL, a model with an intercept term is used.

df.full: The degrees of freedom consumed by the full model

df.null: The degrees of freedom consumed by the reduced model

num.iter: Number of iterations used to build the estimated dependence kernel

svd: A list of length *iteration* with index i containing the $n \times n.sv$ matrix of estimated surrogate variables at iteration i.

iteration: Number of iterations

diagnose: Logical determining whether or not diagnostics were plotted

singular.values: An $n \times iteration$ matrix of singular values. Column i contains the singular values at interation i.

n.sv: Dimension of dependence kernel.

pval: An m-length vector of P values describing the influence of the biological variables from the model that includes the latent variables at the final iteration.

5.3 Preprocessed Normalization

Here we make use of the preprocessed data simulation to demonstrate how to use the sva function. First, we simulate data, then use sva to remove the influence of the adjustment variables on the data. We can accomplish this using the following commands:

```
> seed <- 1234
> sim.d1 <- sim.preProcessed(seed=seed,0.5,0.3,0.1)
> # Update and fit model
> sva.obj <- sva(sim.d1$raw.data, sim.d1$bio.var, NULL, n.sv=5,num.iter=5,diagnose=TRUE)
Estimated rank of dependence kernel:  5
 Iteration:  4
> ps <- snm:::f.pvalue(sim.d1$raw.dat,
model.matrix(~-1+sim.d1$bio.var+sva.obj$svd[[5]]$v),
model.matrix(~sva.obj$svd[[5]]$v))
> ks.test(ps[sim.d1$true.nulls],"punif")$p
[1] 0.800437
```

A unique feature of our software package is that we allow the object returned from sva to serve as the input for a subsequent sva call. This is especially useful when the data set is very large. The following command updates the object:

```
> # Update model and fit again
> sva.obj2 <- sva(sva.obj,num.iter=5)
 Iteration:  9
> ps <- snm:::f.pvalue(sim.d1$raw.dat,
model.matrix(~-1+sim.d1$bio.var+sva.obj2$svd[[10]]$v),
model.matrix(~sva.obj2$svd[[10]]$v))
> ks.test(ps[sim.d1$true.nulls],"punif")$p
[1] 0.8311041
```

Finally, the sva function can control for latent structure in the presence of adjustment variables. This can be accomplished through the following command:

```
> # Now include one of the adjustment variables and fit
> sva.obj <- sva(sim.d1$raw.data, sim.d1$bio.var, NULL, n.sv=5,num.iter=5,diagnose=TRUE)
Estimated rank of dependence kernel:  5
 Iteration:  4
> ps <- snm:::f.pvalue(sim.d1$raw.dat,
model.matrix(~-1+sim.d1$bio.var+sim.d1$adj.var[,6] + sva.obj$svd[[5]]$v),
model.matrix(~sim.d1$adj.var[,6] + sva.obj$svd[[5]]$v))
> ks.test(ps[sim.d1$true.nulls],"punif")$p
[1] 0.3584705
```

5.4 Diagnosing the sva output

The sva methodology requires that a portion of the probes whose data span the rowspace spanned by the latent strucure are independent of the observed variables. When this is not the case the estimated surrogate variables are biased. To help identify this bias we provide a plot that displays the progression of the algorithm over the prescribed iterations, which is activated with the sva

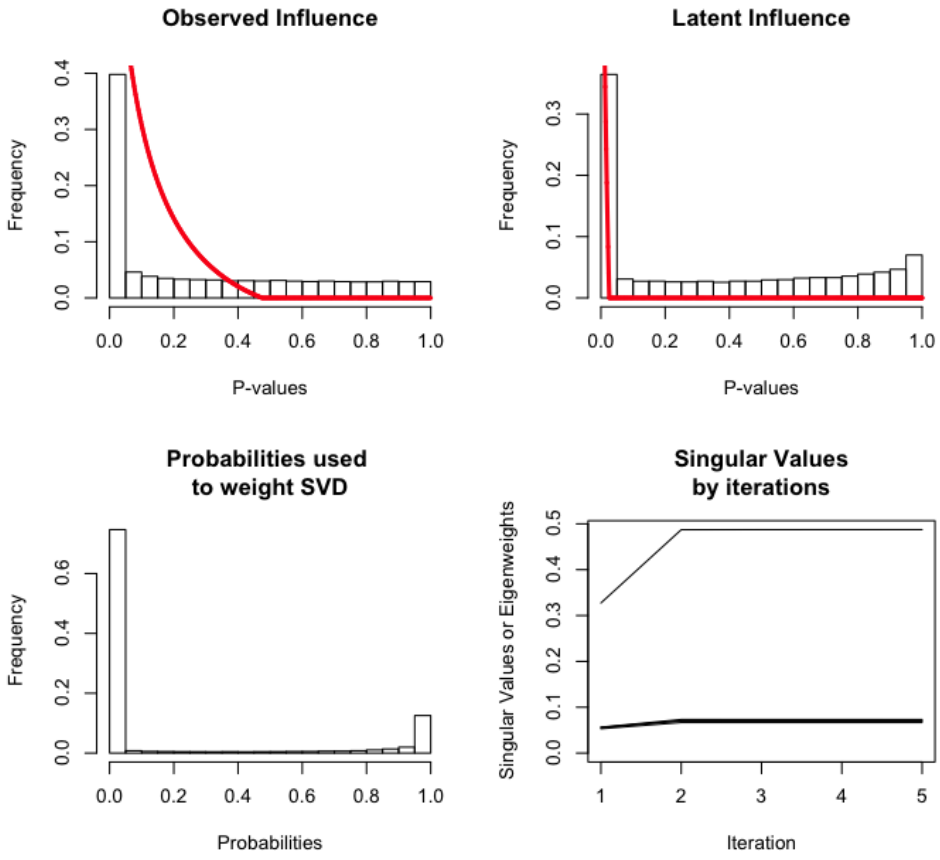

Figure 2: Example Iteration Level Diagnostic Plot (`diagnostic=TRUE`).

argument `diagnose=TRUE`. An example plot from an iteration of the preprocessed dat sva from section 5.3 is presented in figure **??**.
The `snm` diagnostic plot contains four panels. Here is a brief description of each panel:

(A) Observed Influence: Histogram of P values describing the influence of the observed variables on the data. The red line are the corresponding probabilities we estimate.

(B) Latent Influence: Histogram of P values describing the influence of the latent variables on the data. The red line are the corresponding probabilities we estimate.

(C) Probabilities used to weight SVD: A histogram of the probabilities used to weight the SVD.

(D) Singular Values by Iterations: Each line displays the change in percent variable explained by one of the $n.sv$ singular values. This panel can be used to determine whether or not the estimate of the dependence kernel has converged.

6 Simulation Details

6.1 Single Channel Data

Data were simulated for a total of 25,000 probes and 50 arrays. The biological variable of interest is a dichotomous variable defining two groups with 25 arrays sampled from each group. The 50 arrays are split into 5 batches of 10, with each batch containing 5 samples from each biological group. The height variable for each sample is sampled from a Normal(1,0.1). The baseline probe intensities were sampled from a $\chi(1,2)$ distribution. Any baseline intensities greater than 15 were set to a random value from the interval [15,16] in order to remove any artifacts that arise from any extremely large values that can arise from sampling so many values. The random variation terms were sampled from a Normal(0,0.25) and the array functions were defined by randomly sampling coefficients for a two-dimensional β-spline basis functions from a Normal(0,1).

Randomly selected subsets of 30%, 10%, and 20% of the probes were defined as influenced by the biological groups, batch, and height variables, respectfully. The magnitude of the biological effects were sampled from a Normal(1,0.3) distribution, the probe-specific batch effects from a Normal(0,0.3) and the probe-specific height effects from a Normal(1,0.1). An instance of this simulated data is available as the singleChannel data object.

6.2 Double Channel Data

Data were simulated for a total of 25,000 probes and 20 arrays. The design of this study uses a standard dye-swap methodology where each array contains one sample from each of two biological groups. The height variable for each sample is sampled from a Normal(1,0.1). The baseline probe intensities were sampled from a $\chi(1,2)$ distribution. Any baseline intensities greater than 15 were set to a random value from the interval [15,16] in order to remove any artifacts that arise from any extremely large values that can arise from sampling so many values. The random variation terms were sampled from a Normal(0,0.25) and the array and dye functions were defined by randomly sampling coefficients for a two-dimensional β-spline basis functions from a Normal(0,1).

Randomly selected subsets of 30% and 20% of the probes were defined as influenced by the biological groups and height variable, respectfully. The magnitude of the biological effects were sampled from a Normal(1,0.3) distribution, and the probe-specific height effects from a Normal(1,0.1). An instance of this simulated data is available as the doubleChannel data object.

6.3 Reference Design

Simulated data set influenced by a probe-specific biological variable, and intensity-dependent array and dye effects. Data were simulated assuming a uniform reference design for a total of 25,000 probes and 20 arrays, each consisting of two channels. The reference channel consists of a single reference RNA population. The experimental channel measures a dichotomous biological variable specifying two groups (Group 1 and Group 2), with 10 samples per group. The baseline probe intensities were sampled from a chi(1,2) distribution. Any baseline intensities greater than 15 were set to a random value from the interval [15,16]. The random variation terms were sampled from a Normal(0,0.25) and the array and dye functions were defined by randomly sampling coefficients for a two-dimensional B-spline basis functions from a Normal(0,1).
A randomly selected subset of 30% of the probes was defined as influenced by the biological group variable. The magnitude of the biological effects were sampled from a Normal(1,0.3) distribution. An instance of this simulated data can be generated using the code in the examples section below.

6.4 Preprocessed Data

The third simulation is designed to mimic the assumed model for many microarray data sets that come preprocessed in a way that the intensity-dependent effects have already been removed. This commonly occurs for old data sets where the raw data has been deleted or for new studies where only processed data has been released into the public domain. For this scenario we make use of the simulated data from simulation 1, however we do not include the intensity-dependent array effects. An instance of this simulated data is available as the noIntDepFX data object.

References

[1] J. T. Leek and J.D. Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLos Genetics*, 3:e161, 2007.

[2] Mecham B.H Nelson P.S. and Storey J.D. Supervised normalization of microarrays. *Bioinformatics*, 19:387–407, 2010.

[3] J. D. Storey, W. Xiao, J. T. Leek, R. G. Tompkins, and R. W. Davis. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences*, 102:12837–12842, 2005.