# Package 'sva'

November 12, 2026

**Title** Surrogate Variable Analysis

**Version** 3.60.0

**Author** Jeffrey T. Leek <jtleek@gmail.com>, W. Evan Johnson <wej@bu.edu>, Hilary S. Parker <hiparker@jhsph.edu>, Elana J. Fertig <ejfertig@jhmi.edu>, Andrew E. Jaffe <ajaffe@jhsph.edu>, Yuqing Zhang <zhangyuqing.pkusms@gmail.com>, John D. Storey <jstorey@princeton.edu>, Leonardo Collado Torres <lcolladotor@gmail.com>

**Description** The sva package contains functions for removing batch effects and other unwanted variation in high-throughput experiment. Specifically, the sva package contains functions for the identifying and building surrogate variables for high-dimensional data sets. Surrogate variables are covariates constructed directly from high-dimensional data (like gene expression/RNA sequencing/methylation/brain imaging data) that can be used in subsequent analyses to adjust for unknown, unmodeled, or latent sources of noise. The sva package can be used to remove artifacts in three ways: (1) identifying and estimating surrogate variables for unknown sources of variation in high-throughput experiments (Leek and Storey 2007 PLoS Genetics,2008 PNAS), (2) directly removing known batch effects using ComBat (Johnson et al. 2007 Biostatistics) and (3) removing batch effects with known control probes (Leek 2014 biorXiv). Removing batch effects and using surrogate variables in differential expression analysis have been shown to reduce dependence, stabilize error rate estimates, and improve reproducibility, see (Leek and Storey 2007 PLoS Genetics, 2008 PNAS or Leek et al. 2011 Nat. Reviews Genetics).

**Maintainer** Jeffrey T. Leek <jtleek@gmail.com>, John D. Storey <jstorey@princeton.edu>, W. Evan Johnson <wej@bu.edu>

**Depends** R (>= 3.2), mgcv, genefilter, BiocParallel

**Imports** matrixStats, stats, graphics, utils, limma, edgeR

**Suggests** pamr, bladderbatch, BiocStyle, zebrafishRNASeq, testthat

**License** Artistic-2.0

**biocViews** ImmunoOncology, Microarray, StatisticalMethod, Preprocessing, MultipleComparison, Sequencing, RNASeq, BatchEffect, Normalization

**RoxygenNote** 7.0.2

**git_url** https://git.bioconductor.org/packages/sva

**git_branch** RELEASE_3_23

**git_last_commit** 87a4798

**git_last_commit_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-11

# Contents

ComBat . . . 2
ComBat_seq . . . 4
empirical.controls . . . 5
f.pvalue . . . 6
fstats . . . 7
fsva . . . 7
irwsva.build . . . 9
num.sv . . . 10
psva . . . 11
qsva . . . 12
read.degradation.matrix . . . 13
ssva . . . 14
sva . . . 15
sva.check . . . 17
svaseq . . . 18
sva_network . . . 20
twostepsva.build . . . 20

**Index** **22**

---

`ComBat` *Adjust for batch effects using an empirical Bayes framework*

---

## Description

ComBat allows users to adjust for batch effects in datasets where the batch covariate is known, using methodology described in Johnson et al. 2007. It uses either parametric or non-parametric empirical Bayes frameworks for adjusting data for batch effects. Users are returned an expression matrix that has been corrected for batch effects. The input data are assumed to be cleaned and normalized before batch effect removal.

## Usage

```
ComBat(
  dat,
  batch,
  mod = NULL,
  par.prior = TRUE,
  prior.plots = FALSE,
  mean.only = FALSE,
  ref.batch = NULL,
  BPPARAM = bpparam("SerialParam")
)
```

## Arguments

| | |
|---|---|
| dat | Genomic measure matrix (dimensions probe x sample) - for example, expression matrix |
| batch | Batch covariate (only one batch allowed) |
| mod | Model matrix for outcome of interest and other covariates besides batch |
| par.prior | (Optional) TRUE indicates parametric adjustments will be used, FALSE indicates non-parametric adjustments will be used |
| prior.plots | (Optional) TRUE give prior plots with black as a kernel estimate of the empirical batch effect density and red as the parametric |
| mean.only | (Optional) FALSE If TRUE ComBat only corrects the mean of the batch effect (no scale adjustment) |
| ref.batch | (Optional) NULL If given, will use the selected batch as a reference for batch adjustment. |
| BPPARAM | (Optional) BiocParallelParam for parallel operation |

## Value

data A probe x sample genomic measure matrix, adjusted for batch effects.

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:50,]

pheno = pData(dat)
edata = exprs(dat)
batch = pheno$batch
mod = model.matrix(~as.factor(cancer), data=pheno)

# parametric adjustment
combat_edata1 = ComBat(dat=edata, batch=batch, mod=NULL, par.prior=TRUE, prior.plots=FALSE)

# non-parametric adjustment, mean-only version
combat_edata2 = ComBat(dat=edata, batch=batch, mod=NULL, par.prior=FALSE, mean.only=TRUE)
```

```
# reference-batch version, with covariates
combat_edata3 = ComBat(dat=edata, batch=batch, mod=mod, par.prior=TRUE, ref.batch=3)
```

---

`ComBat_seq` *Adjust for batch effects using an empirical Bayes framework in RNA-seq raw counts*

---

## Description

ComBat_seq is an improved model from ComBat using negative binomial regression, which specifically targets RNA-Seq count data.

## Usage

```
ComBat_seq(
  counts,
  batch,
  group = NULL,
  covar_mod = NULL,
  full_mod = TRUE,
  shrink = FALSE,
  shrink.disp = FALSE,
  gene.subset.n = NULL
)
```

## Arguments

| | |
|---|---|
| `counts` | Raw count matrix from genomic studies (dimensions gene x sample) |
| `batch` | Vector / factor for batch |
| `group` | Vector / factor for biological condition of interest |
| `covar_mod` | Model matrix for multiple covariates to include in linear model (signals from these variables are kept in data after adjustment) |
| `full_mod` | Boolean, if TRUE include condition of interest in model |
| `shrink` | Boolean, whether to apply shrinkage on parameter estimation |
| `shrink.disp` | Boolean, whether to apply shrinkage on dispersion |
| `gene.subset.n` | Number of genes to use in empirical Bayes estimation, only useful when shrink = TRUE |

## Value

data A gene x sample count matrix, adjusted for batch effects.

## Examples

```
count_matrix <- matrix(rnbinom(400, size=10, prob=0.1), nrow=50, ncol=8)
batch <- c(rep(1, 4), rep(2, 4))
group <- rep(c(0,1), 4)

# include condition (group variable)
adjusted_counts <- ComBat_seq(count_matrix, batch=batch, group=group, full_mod=TRUE)

# do not include condition
adjusted_counts <- ComBat_seq(count_matrix, batch=batch, group=NULL, full_mod=FALSE)
```

---

`empirical.controls` *A function for estimating the probability that each gene is an empirical control*

---

## Description

This function uses the iteratively reweighted surrogate variable analysis approach to estimate the probability that each gene is an empirical control.

## Usage

```
empirical.controls(
  dat,
  mod,
  mod0 = NULL,
  n.sv,
  B = 5,
  type = c("norm", "counts")
)
```

## Arguments

| | |
|---|---|
| `dat` | The transformed data matrix with the variables in rows and samples in columns |
| `mod` | The model matrix being used to fit the data |
| `mod0` | The null model being compared when fitting the data |
| `n.sv` | The number of surogate variables to estimate |
| `B` | The number of iterations of the irwsva algorithm to perform |
| `type` | If type is norm then standard irwsva is applied, if type is counts, then the moderated log transform is applied first |

## Value

pcontrol A vector of probabilites that each gene is a control.

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)

n.sv = num.sv(edata,mod,method="leek")
pcontrol <- empirical.controls(edata,mod,mod0=NULL,n.sv=n.sv,type="norm")
```

---

`f.pvalue` *A function for quickly calculating f statistic p-values for use in sva*

---

## Description

This function does simple linear algebra to calculate f-statistics for each row of a data matrix comparing the nested models defined by the design matrices for the alternative (mod) and and null (mod0) cases. The columns of mod0 must be a subset of the columns of mod.

## Usage

```
f.pvalue(dat, mod, mod0)
```

## Arguments

| | |
|---|---|
| `dat` | The transformed data matrix with the variables in rows and samples in columns |
| `mod` | The model matrix being used to fit the data |
| `mod0` | The null model being compared when fitting the data |

## Value

p A vector of F-statistic p-values one for each row of dat.

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:50,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)
mod0 = model.matrix(~1,data=pheno)

pValues = f.pvalue(edata,mod,mod0)
qValues = p.adjust(pValues,method="BH")
```

## fstats — *A function for quickly calculating f statistics for use in sva*

### Description

This function does simple linear algebra to calculate f-statistics for each row of a data matrix comparing the nested models defined by the design matrices for the alternative (mod) and and null (mod0) cases. The columns of mod0 must be a subset of the columns of mod.

### Usage

```
fstats(dat, mod, mod0)
```

### Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| mod | The model matrix being used to fit the data |
| mod0 | The null model being compared when fitting the data |

### Value

fstats A vector of F-statistics one for each row of dat.

### Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:50,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)
mod0 = model.matrix(~1,data=pheno)

fs <- fstats(edata, mod, mod0)
```

## fsva — *A function for performing frozen surrogate variable analysis as proposed in Parker, Corrada Bravo and Leek 2013*

### Description

This function performs frozen surrogate variable analysis as described in Parker, Corrada Bravo and Leek 2013. The approach uses a training database to create surrogate variables which are then used to remove batch effects both from the training database and a new data set for prediction purposes. For inferential analysis see sva, svaseq, with low level functionality available through irwsva.build and ssva.

## Usage

```
fsva(dbdat, mod, sv, newdat = NULL, method = c("fast", "exact"))
```

## Arguments

| | |
|---|---|
| dbdat | A m genes by n arrays matrix of expression data from the database/training data |
| mod | The model matrix for the terms included in the analysis for the training data |
| sv | The surrogate variable object created by running sva on dbdat using mod. |
| newdat | (optional) A set of test samples to be adjusted using the training database |
| method | If method ="fast" then the SVD is calculated using an online approach, this may introduce slight bias. If method="exact" the exact SVD is calculated, but will be slower |

## Value

db An adjusted version of the training database where the effect of batch/expression heterogeneity has been removed

new An adjusted version of the new samples, adjusted one at a time using the fsva methodology.

newsv Surrogate variables for the new samples

## Examples

```
library(bladderbatch)
library(pamr)
data(bladderdata)
dat <- bladderEset[1:50,]

pheno = pData(dat)
edata = exprs(dat)

set.seed(1234)
trainIndicator = sample(1:57,size=30,replace=FALSE)
testIndicator = (1:57)[-trainIndicator]
trainData = edata[,trainIndicator]
testData = edata[,testIndicator]
trainPheno = pheno[trainIndicator,]
testPheno = pheno[testIndicator,]

mydata = list(x=trainData,y=trainPheno$cancer)
mytrain = pamr.train(mydata)
table(pamr.predict(mytrain,testData,threshold=2),testPheno$cancer)

trainMod = model.matrix(~cancer,data=trainPheno)
trainMod0 = model.matrix(~1,data=trainPheno)
trainSv = sva(trainData,trainMod,trainMod0)

fsvaobj = fsva(trainData,trainMod,trainSv,testData)
mydataSv = list(x=fsvaobj$db,y=trainPheno$cancer)
mytrainSv = pamr.train(mydataSv)
```

```
table(pamr.predict(mytrainSv,fsvaobj$new,threshold=1),testPheno$cancer)
```

---

`irwsva.build` *A function for estimating surrogate variables by estimating empirical control probes*

---

## Description

This function is the implementation of the iteratively re-weighted least squares approach for estimating surrogate variables. As a buy product, this function produces estimates of the probability of being an empirical control. See the function `empirical.controls` for a direct estimate of the empirical controls.

## Usage

```
irwsva.build(dat, mod, mod0 = NULL, n.sv, B = 5)
```

## Arguments

| | |
|---|---|
| `dat` | The transformed data matrix with the variables in rows and samples in columns |
| `mod` | The model matrix being used to fit the data |
| `mod0` | The null model being compared when fitting the data |
| `n.sv` | The number of surogate variables to estimate |
| `B` | The number of iterations of the irwsva algorithm to perform |

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity

pprob.b A vector of the posterior probabilities each gene is affected by mod

n.sv The number of significant surrogate variables

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)

n.sv = num.sv(edata,mod,method="leek")
res <- irwsva.build(edata, mod, mod0 = NULL,n.sv,B=5)
```

num.sv *A function for calculating the number of surrogate variables to estimate in a model*

## Description

This function estimates the number of surrogate variables that should be included in a differential expression model. The default approach is based on a permutation procedure originally prooposed by Buja and Eyuboglu 1992. The function also provides an interface to the asymptotic approach proposed by Leek 2011 Biometrics.

## Usage

```
num.sv(dat, mod, method = c("be", "leek"), vfilter = NULL, B = 20, seed = NULL)
```

## Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| mod | The model matrix being used to fit the data |
| method | One of "be" or "leek" as described in the details section |
| vfilter | You may choose to filter to the vfilter most variable rows before performing the analysis |
| B | The number of permutaitons to use if method = "be" |
| seed | Set a seed when using the permutation approach |

## Value

n.sv The number of surrogate variables to use in the sva software

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)

n.sv = num.sv(edata,mod,method="leek")
```

psva *A function for estimating surrogate variables with the two step approach of Leek and Storey 2007*

## Description

This function is the implementation of the two step approach for estimating surrogate variables proposed by Leek and Storey 2007 PLoS Genetics. This function is primarily included for backwards compatibility. Newer versions of the sva algorithm are available through sva, svaseq, with low level functionality available through irwsva.build and ssva.

## Usage

```
psva(dat, batch, ...)
```

## Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| batch | A factor variable giving the known batch levels |
| ... | Other arguments to the sva function. |

## Value

psva.D Data with batch effect removed but biological heterogeneity preserved

## Author(s)

Elana J. Fertig

## Examples

```
library(bladderbatch)
library(limma)
data(bladderdata)
dat <- bladderEset[1:50,]

pheno = pData(dat)
edata = exprs(dat)
batch = pheno$batch
batch.fac = as.factor(batch)

psva_data <- psva(edata,batch.fac)
```

qsva *A function for computing quality surrogate variables (qSVs)*

## Description

This function computes quality surrogate variables (qSVs) from the library-size- and read-length-normalized degradation matrix for subsequent RNA quality correction

## Usage

```
qsva(
  degradationMatrix,
  mod = matrix(1, ncol = 1, nrow = ncol(degradationMatrix))
)
```

## Arguments

| | |
|---|---|
| degradationMatrix | the normalized degradation matrix, region by sample |
| mod | (Optional) statistical model used in DE analysis |

## Value

the qSV adjustment variables

## Examples

```
## Find files
bwPath <- system.file('extdata', 'bwtool', package = 'sva')

## Read the data
degCovAdj = read.degradation.matrix(
 covFiles = list.files(bwPath,full.names=TRUE),
 sampleNames = list.files(bwPath), readLength = 76,
 totalMapped = rep(100e6,5),type="bwtool")

## Input data
 head(degCovAdj)

## Results
qsva(degCovAdj)
```

`read.degradation.matrix` *A function for reading in coverage data from degradation-susceptible regions*

## Description

This function reads in degradation regions to form a library-size- and read-length-normalized degradation matrix for subsequent RNA quality correction

## Usage

```
read.degradation.matrix(
  covFiles,
  sampleNames,
  totalMapped,
  readLength = 100,
  normFactor = 8e+07,
  type = c("bwtool", "region_matrix_single", "region_matrix_all"),
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| `covFiles` | coverage file(s) for degradation regions |
| `sampleNames` | sample names; creates column names of degradation matrix |
| `totalMapped` | how many reads per sample (library size normalization) |
| `readLength` | read length in base pairs (read length normalization) |
| `normFactor` | common library size to normalize to; 80M reads as default |
| `type` | whether input are individual 'bwtool' output, 'region_matrix' run on individual samples, or 'region_matrix' run on all samples together |
| `BPPARAM` | (Optional) BiocParallelParam for parallel operation |

## Value

the normalized degradation matrix, region by sample

## Examples

```
# bwtool
bwPath = system.file('extdata', 'bwtool', package = 'sva')
degCovAdj = read.degradation.matrix(
 covFiles = list.files(bwPath,full.names=TRUE),
 sampleNames = list.files(bwPath), readLength = 76,
 totalMapped = rep(100e6,5),type="bwtool")
 head(degCovAdj)
```

```
# region_matrix: each sample
r1Path = system.file('extdata', 'region_matrix_one', package = 'sva')
degCovAdj1 = read.degradation.matrix(
 covFiles = list.files(r1Path,full.names=TRUE),
 sampleNames = list.files(r1Path), readLength = 76,
 totalMapped = rep(100e6,5),type="region_matrix_single")
 head(degCovAdj1)

r2Path = system.file('extdata', 'region_matrix_all', package = 'sva')
degCovAdj2 = read.degradation.matrix(
 covFiles = list.files(r2Path,full.names=TRUE),
 sampleNames = list.files(r1Path), readLength = 76,
 totalMapped = rep(100e6,5),type="region_matrix_all")
head(degCovAdj2)
```

---

`ssva` *A function for estimating surrogate variables using a supervised approach*

---

## Description

This function implements a supervised surrogate variable analysis approach where genes/probes known to be affected by artifacts but not by the biological variables of interest are assumed to be known in advance. This supervised sva approach can be called through the `sva` and `svaseq` functions by specifying controls.

## Usage

```
ssva(dat, controls, n.sv)
```

## Arguments

| | |
|---|---|
| `dat` | The transformed data matrix with the variables in rows and samples in columns |
| `controls` | A vector of probabilities (between 0 and 1, inclusive) that each gene is a control. A value of 1 means the gene is certainly a control and a value of 0 means the gene is certainly not a control. |
| `n.sv` | The number of surogate variables to estimate |

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity (exactly equal to controls for ssva)

pprob.b A vector of the posterior probabilities each gene is affected by mod (always null for ssva)

n.sv The number of significant surrogate variables

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)

n.sv = num.sv(edata,mod,method="leek")
set.seed(1234)
controls <- runif(nrow(edata))
ssva_res <- ssva(edata,controls,n.sv)
```

---

sva *sva: a package for removing artifacts from microarray and sequencing data*

---

## Description

sva has functionality to estimate and remove artifacts from high dimensional data the sva function can be used to estimate artifacts from microarray data the svaseq function can be used to estimate artifacts from count-based RNA-sequencing (and other sequencing) data. The ComBat function can be used to remove known batch effecs from microarray data. The fsva function can be used to remove batch effects for prediction problems.

This function is the implementation of the iteratively re-weighted least squares approach for estimating surrogate variables. As a by product, this function produces estimates of the probability of being an empirical control. See the function empirical.controls for a direct estimate of the empirical controls.

## Usage

```
sva(
  dat,
  mod,
  mod0 = NULL,
  n.sv = NULL,
  controls = NULL,
  method = c("irw", "two-step", "supervised"),
  vfilter = NULL,
  B = 5,
  numSVmethod = "be"
)
```

## Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| mod | The model matrix being used to fit the data |
| mod0 | The null model being compared when fitting the data |
| n.sv | The number of surogate variables to estimate |
| controls | A vector of probabilities (between 0 and 1, inclusive) that each gene is a control. A value of 1 means the gene is certainly a control and a value of 0 means the gene is certainly not a control. |
| method | For empirical estimation of control probes use "irw". If control probes are known use "supervised" |
| vfilter | You may choose to filter to the vfilter most variable rows before performing the analysis. vfilter must be NULL if method is "supervised" |
| B | The number of iterations of the irwsva algorithm to perform |
| numSVmethod | If n.sv is NULL, sva will attempt to estimate the number of needed surrogate variables. This should not be adapted by the user unless they are an expert. |

## Details

A vignette is available by typing `browseVignettes("sva")` in the R prompt.

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity

pprob.b A vector of the posterior probabilities each gene is affected by mod

n.sv The number of significant surrogate variables

## Author(s)

Jeffrey T. Leek, W. Evan Johnson, Hilary S. Parker, Andrew E. Jaffe, John D. Storey, Yuqing Zhang

## References

For the package: Leek JT, Johnson WE, Parker HS, Jaffe AE, and Storey JD. (2012) The sva package for removing batch effects and other unwanted variation in high-throughput experiments. Bioinformatics DOI:10.1093/bioinformatics/bts034

For sva: Leek JT and Storey JD. (2008) A general framework for multiple testing dependence. Proceedings of the National Academy of Sciences , 105: 18718-18723.

For sva: Leek JT and Storey JD. (2007) Capturing heterogeneity in gene expression studies by 'Surrogate Variable Analysis'. PLoS Genetics, 3: e161.

For Combat: Johnson WE, Li C, Rabinovic A (2007) Adjusting batch effects in microarray expression data using empirical Bayes methods. Biostatistics, 8 (1), 118-127

For svaseq: Leek JT (2014) svaseq: removing batch and other artifacts from count-based sequencing data. bioRxiv doi: TBD

For fsva: Parker HS, Bravo HC, Leek JT (2013) Removing batch effects for prediction problems with frozen surrogate variable analysis arXiv:1301.3947

For psva: Parker HS, Leek JT, Favorov AV, Considine M, Xia X, Chavan S, Chung CH, Fertig EJ (2014) Preserving biological heterogeneity with a permuted surrogate variable analysis for genomics batch correction Bioinformatics doi: 10.1093/bioinformatics/btu375

## Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)
mod0 = model.matrix(~1,data=pheno)

n.sv = num.sv(edata,mod,method="leek")
svobj = sva(edata,mod,mod0,n.sv=n.sv)
```

---

`sva.check` *A function for post-hoc checking of an sva object to check for degenerate cases.*

---

## Description

This function is designed to check for degenerate cases in the sva fit and fix the sva object where possible.

## Usage

```
sva.check(svaobj, dat, mod, mod0)
```

## Arguments

| | |
|---|---|
| `svaobj` | The transformed data matrix with the variables in rows and samples in columns |
| `dat` | The data set that was used to build the surrogate variables |
| `mod` | The model matrix being used to fit the data |
| `mod0` | The null model matrix being used to fit the data |

## Details

`empirical.controls` for a direct estimate of the empirical controls.

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity

pprob.b A vector of the posterior probabilities each gene is affected by mod

n.sv The number of significant surrogate variables

## Examples

```
library(bladderbatch)
data(bladderdata)
#dat <- bladderEset
dat <- bladderEset[1:5000,]

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)
mod0 = model.matrix(~1,data=pheno)

n.sv = num.sv(edata,mod,method="leek")
svobj = sva(edata,mod,mod0,n.sv=n.sv)
svacheckobj = sva.check(svobj,edata,mod,mod0)
```

---

`svaseq` *A function for estimating surrogate variables for count based RNA-seq data.*

---

## Description

This function is the implementation of the iteratively re-weighted least squares approach for estimating surrogate variables. As a by product, this function produces estimates of the probability of being an empirical control. This function first applies a moderated log transform as described in Leek 2014 before calculating the surrogate variables. See the function `empirical.controls` for a direct estimate of the empirical controls.

## Usage

```
svaseq(
  dat,
  mod,
  mod0 = NULL,
  n.sv = NULL,
  controls = NULL,
  method = c("irw", "two-step", "supervised"),
  vfilter = NULL,
  B = 5,
  numSVmethod = "be",
```

```
  constant = 1
)
```

## Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| mod | The model matrix being used to fit the data |
| mod0 | The null model being compared when fitting the data |
| n.sv | The number of surogate variables to estimate |
| controls | A vector of probabilities (between 0 and 1, inclusive) that each gene is a control. A value of 1 means the gene is certainly a control and a value of 0 means the gene is certainly not a control. |
| method | For empirical estimation of control probes use "irw". If control probes are known use "supervised" |
| vfilter | You may choose to filter to the vfilter most variable rows before performing the analysis. vfilter must be NULL if method is "supervised" |
| B | The number of iterations of the irwsva algorithm to perform |
| numSVmethod | If n.sv is NULL, sva will attempt to estimate the number of needed surrogate variables. This should not be adapted by the user unless they are an expert. |
| constant | The function takes log(dat + constant) before performing sva. By default constant = 1, all values of dat + constant should be positive. |

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity

pprob.b A vector of the posterior probabilities each gene is affected by mod

n.sv The number of significant surrogate variables

## Examples

```
library(zebrafishRNASeq)
data(zfGenes)
filter = apply(zfGenes, 1, function(x) length(x[x>5])>=2)
filtered = zfGenes[filter,]
genes = rownames(filtered)[grep("^ENS", rownames(filtered))]
controls = grepl("^ERCC", rownames(filtered))
group = as.factor(rep(c("Ctl", "Trt"), each=3))
dat0 = as.matrix(filtered)

mod1 = model.matrix(~group)
mod0 = cbind(mod1[,1])
svseq = svaseq(dat0,mod1,mod0,n.sv=1)$sv
plot(svseq,pch=19,col="blue")
```

## sva_network *A function to adjust gene expression data before network inference*

### Description

This function corrects a gene expression matrix prior to network inference by returning the residuals after regressing out the top principal components. The number of principal components to remove can be determined using a permutation-based approach using the "num.sv" function with method = "be"

### Usage

```
sva_network(dat, n.pc)
```

### Arguments

| | |
|---|---|
| dat | The uncorrected normalized gene expression data matrix with samples in rows and genes in columns |
| n.pc | The number of principal components to remove |

### Value

dat.adjusted Cleaned gene expression data matrix with the top prinicpal components removed

### Examples

```
library(bladderbatch)
data(bladderdata)
dat <- bladderEset[1:5000,]

edata = exprs(dat)
mod = matrix(1, nrow = dim(dat)[2], ncol = 1)

n.pc = num.sv(edata, mod, method="be")
dat.adjusted = sva_network(t(edata), n.pc)
```

## twostepsva.build *A function for estimating surrogate variables with the two step approach of Leek and Storey 2007*

### Description

This function is the implementation of the two step approach for estimating surrogate variables proposed by Leek and Storey 2007 PLoS Genetics. This function is primarily included for backwards compatibility. Newer versions of the sva algorithm are available through sva, svaseq, with low level functionality available through irwsva.build and ssva.

## Usage

```
twostepsva.build(dat, mod, n.sv)
```

## Arguments

| | |
|---|---|
| dat | The transformed data matrix with the variables in rows and samples in columns |
| mod | The model matrix being used to fit the data |
| n.sv | The number of surogate variables to estimate |

## Value

sv The estimated surrogate variables, one in each column

pprob.gam: A vector of the posterior probabilities each gene is affected by heterogeneity

pprob.b A vector of the posterior probabilities each gene is affected by mod (this is always null for the two-step approach)

n.sv The number of significant surrogate variables

## Examples

```
library(bladderbatch)
library(limma)
data(bladderdata)
dat <- bladderEset

pheno = pData(dat)
edata = exprs(dat)
mod = model.matrix(~as.factor(cancer), data=pheno)

n.sv = num.sv(edata,mod,method="leek")
svatwostep <- twostepsva.build(edata,mod,n.sv)
```

# Index

ComBat, 2, *15*
ComBat_seq, 4

empirical.controls, 5, *9*, *15*, *17, 18*

f.pvalue, 6
fstats, 7
fsva, 7, *15*

irwsva.build, *7*, 9, *11*, *20*

num.sv, 10

psva, 11

qsva, 12

read.degradation.matrix, 13

ssva, *7*, *11*, 14, *20*
sva, *7*, *11*, *14, 15*, 15, *20*
sva.check, 17
sva_network, 20
svaseq, *7*, *11*, *14, 15*, 18, *20*

twostepsva.build, 20