

# Package ‘nnSVG’

May 16, 2025

**Version** 1.13.0

**Title** Scalable identification of spatially variable genes in spatially-resolved transcriptomics data

**Description** Method for scalable identification of spatially variable genes (SVGs) in spatially-resolved transcriptomics data. The method is based on nearest-neighbor Gaussian processes and uses the BRISC algorithm for model fitting and parameter estimation. Allows identification and ranking of SVGs with flexible length scales across a tissue slide or within spatial domains defined by covariates. Scales linearly with the number of spatial locations and can be applied to datasets containing thousands or more spatial locations.

**URL** <https://github.com/lmweber/nnSVG>

**BugReports** <https://github.com/lmweber/nnSVG/issues>

**License** MIT + file LICENSE

**Encoding** UTF-8

**biocViews** Spatial, SingleCell, Transcriptomics, GeneExpression, Preprocessing

**Depends** R (>= 4.2)

**Imports** SpatialExperiment, SingleCellExperiment, SummarizedExperiment, BRISC, BiocParallel, Matrix, matrixStats, stats, methods

**VignetteBuilder** knitr

**Suggests** BiocStyle, knitr, rmarkdown, STexampleData, WeberDivechaLCdata, scran, ggplot2, testthat

**RoxygenNote** 7.2.3

**git\_url** <https://git.bioconductor.org/packages/nnSVG>

**git\_branch** devel

**git\_last\_commit** 4a2d421

**git\_last\_commit\_date** 2025-04-15

**Repository** Bioconductor 3.22

**Date/Publication** 2025-05-15

**Author** Lukas M. Weber [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-3282-1730>>),  
Stephanie C. Hicks [aut] (ORCID:  
<<https://orcid.org/0000-0002-7858-0231>>)

**Maintainer** Lukas M. Weber <lmweb012@gmail.com>

## Contents

filter_genes	2
nnSVG	3
<b>Index</b>	<b>7</b>

---

filter_genes	<i>Preprocessing function to filter genes</i>
--------------	-----------------------------------------------

---

### Description

Preprocessing function to filter low-expressed genes and/or mitochondrial genes for 'nnSVG'.

### Usage

```
filter_genes(
  spe,
  filter_genes_ncounts = 3,
  filter_genes_pcspots = 0.5,
  filter_mito = TRUE
)
```

### Arguments

spe	SpatialExperiment: Input data, assumed to be formatted as a SpatialExperiment object with an assay slot named counts containing raw expression counts.
filter_genes_ncounts	numeric: Filtering parameter for low-expressed genes. Filtering retains genes containing at least filter_genes_ncounts expression counts in at least filter_genes_pcspots percent of the total number of spatial locations (spots). Defaults: filter_genes_ncounts = 3, filter_genes_pcspots = 0.5, i.e. keep genes with at least 3 counts in at least 0.5 percent of spots. Set to NULL to disable.
filter_genes_pcspots	numeric: Second filtering parameter for low-expressed genes. Set to NULL to disable. See filter_genes_ncounts for details.
filter_mito	logical: Whether to filter out mitochondrial genes, identified by gene names starting with "MT" or "mt". This requires that the rowData slot of the input object contains a column named gene_name. Default = TRUE. Set to FALSE to disable.

### Details

Preprocessing function to filter low-expressed genes and/or mitochondrial genes for 'nnSVG'.

This function can be used to filter out low-expressed genes and/or mitochondrial genes before additional preprocessing (calculating logcounts or deviance residuals) and running 'nnSVG'.

We use this function in the examples and vignettes in the 'nnSVG' package, and provide default filtering parameter values that are appropriate for 10x Genomics Visium data.

The use of this function is optional. Users can also perform filtering and preprocessing separately, and run [nnSVG](#) on a preprocessed SpatialExperiment object.

**Value**

Returns `SpatialExperiment` with filtered genes (rows) removed.

**Examples**

```
library(SpatialExperiment)
library(STexampleData)

# load example dataset from STexampleData package
spe <- Visium_humanDLPFC()

# preprocessing steps

# keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
dim(spe)

# filter low-expressed and mitochondrial genes
spe <- filter_genes(spe)
dim(spe)
```

---

nnSVG

*nnSVG*


---

**Description**

Function to run 'nnSVG' method to identify spatially variable genes (SVGs) in spatially-resolved transcriptomics data.

**Usage**

```
nnSVG(
  input,
  spatial_coords = NULL,
  X = NULL,
  assay_name = "logcounts",
  n_neighbors = 10,
  order = "AMMD",
  n_threads = 1,
  BPPARAM = NULL,
  verbose = FALSE
)
```

**Arguments**

`input` `SpatialExperiment` or numeric matrix: Input data, which can either be a `SpatialExperiment` object or a numeric matrix of values. If it is a `SpatialExperiment` object, it is assumed to have an assay slot containing either logcounts (e.g. from the `scrn` package) or deviance residuals (e.g. from the `scry` package), and a `spatialCoords` slot containing spatial coordinates of the measurements. If it is a numeric matrix, the values are assumed to already be normalized and transformed (e.g. logcounts), formatted as rows = genes and columns = spots, and

a separate numeric matrix of spatial coordinates must also be provided with the `spatial_coords` argument.

<code>spatial_coords</code>	numeric matrix: Matrix containing columns of spatial coordinates, formatted as rows = spots. This must be provided if input is provided as a numeric matrix of values, and is ignored if input is provided as a <code>SpatialExperiment</code> object. Default = NULL.
<code>X</code>	numeric matrix: Optional design matrix containing columns of covariates per spatial location, e.g. known spatial domains. Number of rows must match the number of spatial locations. Default = NULL, which fits an intercept-only model.
<code>assay_name</code>	character: If input is provided as a <code>SpatialExperiment</code> object, this argument selects the name of the assay slot in the input object containing the pre-processed gene expression values. For example, <code>logcounts</code> for log-transformed normalized counts from the <code>scran</code> package, or <code>binomial_deviance_residuals</code> for deviance residuals from the <code>scry</code> package. Default = "logcounts", or ignored if input is provided as a numeric matrix of values.
<code>n_neighbors</code>	integer: Number of nearest neighbors for fitting the nearest-neighbor Gaussian process (NNGP) model with BRISC. The default value is 10, which we recommend for most datasets. Higher numbers (e.g. 15) may give slightly improved likelihood estimates in some datasets (at the expense of slower runtime), and smaller numbers (e.g. 5) will give faster runtime (at the expense of reduced performance). Default = 10.
<code>order</code>	character: Ordering scheme to use for ordering coordinates with BRISC. Default = "AMMD" for "approximate maximum minimum distance", which is recommended for datasets with at least 65 spots. For very small datasets ( $n \leq 65$ ), "Sum_coords" can be used instead. See BRISC documentation for details. Default = "AMMD".
<code>n_threads</code>	integer: Number of threads for parallelization. Default = 1. We recommend setting this equal to the number of cores available (if working on a laptop or desktop) or around 10 or more (if working on a compute cluster).
<code>BPPARAM</code>	<code>BiocParallelParam</code> : Optional additional argument for parallelization. This argument is provided for advanced users of <code>BiocParallel</code> for further flexibility for parallelization on some operating systems. If provided, this should be an instance of <code>BiocParallelParam</code> . For most users, the recommended option is to use the <code>n_threads</code> argument instead. Default = NULL, in which case <code>n_threads</code> will be used instead.
<code>verbose</code>	logical: Whether to display verbose output for model fitting and parameter estimation from BRISC. Default = FALSE.

## Details

Function to run 'nnSVG' method to identify spatially variable genes (SVGs) in spatially-resolved transcriptomics data.

The 'nnSVG' method is based on nearest-neighbor Gaussian processes (Datta et al. 2016) and uses the BRISC algorithm (Saha and Datta 2018) for model fitting and parameter estimation. The method scales linearly with the number of spatial locations, and can be applied to datasets containing thousands or more spatial locations. For more details on the method, see our paper.

This function runs 'nnSVG' for a full dataset. The function fits a separate model for each gene, using parallelization with `BiocParallel` for faster runtime. The parameter estimates from BRISC ( $\sigma$ ,  $\tau$ ,  $\phi$ ) for each gene are stored in 'Theta' in the BRISC output.

Note that the method and this function are designed for a single tissue section. For an example of how to run nnSVG in a dataset consisting of multiple tissue sections, see the tutorial in the nnSVG package vignette.

'nnSVG' performs inference on the spatial variance parameter estimates ( $\sigma^2$ ) using a likelihood ratio (LR) test against a simpler linear model without spatial terms (i.e. without  $\tau^2$  or  $\phi$ ). The estimated LR statistics can then be used to rank SVGs. P-values are calculated from the LR statistics using the asymptotic chi-squared distribution with 2 degrees of freedom, and multiple testing adjusted p-values are calculated using the Benjamini-Hochberg method. We also calculate an effect size, defined as the proportion of spatial variance,  $\text{prop}_{sv} = \sigma^2 / (\sigma^2 + \tau^2)$ .

The function assumes the input is provided either as a `SpatialExperiment` object or a numeric matrix of values. If the input is a `SpatialExperiment` object, it is assumed to contain an assay slot containing either log-transformed normalized counts (also known as logcounts, e.g. from the `scrn` package) or deviance residuals (e.g. from the `scry` package), which have been preprocessed, quality controlled, and filtered to remove low-quality spatial locations. If the input is a numeric matrix of values, these values are assumed to already be normalized and transformed (e.g. logcounts).

## Value

If the input was provided as a `SpatialExperiment` object, the output values are returned as additional columns in the `rowData` slot of the input object. If the input was provided as a numeric matrix of values, the output is returned as a numeric matrix. The output values include spatial variance parameter estimates, likelihood ratio (LR) statistics, effect sizes (proportion of spatial variance), p-values, and multiple testing adjusted p-values.

## Examples

```
library(SpatialExperiment)
library(STexampleData)
library(scran)

### Example 1
### for more details see extended example in vignette

# load example dataset from STexampleData package
spe1 <- Visium_humanDLPFC()

# preprocessing steps
# keep only spots over tissue
spe1 <- spe1[, colData(spe1)$in_tissue == 1]
# skip spot-level quality control (already performed in this dataset)
# filter low-expressed and mitochondrial genes
spe1 <- filter_genes(spe1)
# calculate logcounts using library size factors
spe1 <- computeLibraryFactors(spe1)
spe1 <- logNormCounts(spe1)

# select small number of genes for fast runtime in this example
set.seed(123)
ix <- c(
  which(rowData(spe1)$gene_name %in% c("PCP4", "NPY")),
  sample(seq_len(nrow(spe1)), 2)
)
spe1 <- spe1[ix, ]
```

```
# run nnSVG
set.seed(123)
spe1 <- nnSVG(spe1)

# show results
rowData(spe1)

### Example 2: With covariates
### for more details see extended example in vignette

# load example dataset from SExampleData package
spe2 <- SlideSeqV2_mouseHPC()

# preprocessing steps
# remove spots with NA cell type labels
spe2 <- spe2[, !is.na(colData(spe2)$celltype)]
# skip spot-level quality control (already performed in this dataset)
# filter low-expressed and mitochondrial genes
spe2 <- filter_genes(
  spe2, filter_genes_ncounts = 1, filter_genes_pcspots = 1,
  filter_mito = TRUE
)
# calculate logcounts using library size normalization
spe2 <- computeLibraryFactors(spe2)
spe2 <- logNormCounts(spe2)

# select small number of genes for fast runtime in this example
set.seed(123)
ix <- c(
  which(rowData(spe2)$gene_name %in% c("Cpne9", "Rgs14")),
  sample(seq_len(nrow(spe2)), 2)
)
spe2 <- spe2[ix, ]

# create model matrix for cell type labels
X <- model.matrix(~ colData(spe2)$celltype)

# run nnSVG with covariates
set.seed(123)
spe2 <- nnSVG(spe2, X = X)

# show results
rowData(spe2)
```

# Index

`filter_genes`, 2

`nnSVG`, 2, 3