

# Package ‘dandelionR’

May 20, 2025

**Title** Single-cell Immune Repertoire Trajectory Analysis in R

**Version** 1.1.0

**Description** dandelionR is an R package for performing single-cell immune repertoire trajectory analysis, based on the original python implementation. It provides the necessary functions to interface with scRepertoire and a custom implementation of an absorbing Markov chain for pseudo-time inference, inspired by the Palantir Python package.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**biocViews** Software, ImmunoOncology, SingleCell

**Collate** 'check.R' 'constructMarkovChain.R' 'dandelionR.R' 'data.R'  
'determMultiscaleSpace.R' 'terminalStateFromMarkovChain.R'  
'differentiationProbabilities.R' 'filterCells.R' 'getPbs.R'  
'projectProbability.R' 'maxMinSampling.R' 'minMaxScale.R'  
'markovProbability.R' 'miloUmap.R' 'projectPseudotimeToCell.R'  
'setupVdjPseudobulk.R' 'splitCTgene.R' 'vdjPseudobulk.R'

**Imports** BiocGenerics, bluster, destiny, igraph, MASS, Matrix, methods,  
miloR, purrr, rlang, S4Vectors, SingleCellExperiment, spam,  
stats, SummarizedExperiment, uwot

**Suggests** BiocStyle, knitr, rmarkdown, RColorBrewer, scater,  
scRepertoire, testthat

**VignetteBuilder** knitr

**URL** <https://www.github.com/tuonglab/dandelionR/>

**BugReports** <https://www.github.com/tuonglab/dandelionR/issues>

**Depends** R (>= 4.4.0)

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

**git\_branch** devel

**git\_last\_commit** e0d38d2

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

**Repository** Bioconductor 3.22

**Date/Publication** 2025-05-20

**Author** Jiawei Yu [aut] (ORCID: <<https://orcid.org/0009-0005-9170-7881>>),  
 Nicholas Borchering [aut] (ORCID:  
 <<https://orcid.org/0000-0003-1427-6342>>),  
 Kelvin Tuong [aut, cre] (ORCID:  
 <<https://orcid.org/0000-0002-6735-6808>>)

**Maintainer** Kelvin Tuong <[z.tuong@uq.edu.au](mailto:z.tuong@uq.edu.au)>

## Contents

.addColData . . . . .	3
.allowedChain . . . . .	3
.calDif . . . . .	4
.classCheck . . . . .	4
.collapse_nested_list . . . . .	5
.constructMarkovChain . . . . .	5
.determExtractColN . . . . .	6
.determineMultiscaleSpace . . . . .	6
.determTerminal . . . . .	7
.extractVdj . . . . .	7
.featureSpaceConstruct . . . . .	8
.filterCells . . . . .	8
.filterProductivity . . . . .	9
.filterUnmapped . . . . .	9
.findNewWaypoints . . . . .	10
.generateExtractColumn . . . . .	11
.generateExtractName . . . . .	11
.getPbs . . . . .	12
.getPbsCol . . . . .	12
.getPbsPerCol . . . . .	13
.KNNind . . . . .	13
.maxMinSampling . . . . .	14
.minMaxScale . . . . .	14
.normalizeFeatureSpace . . . . .	15
.normalizePerVDJ . . . . .	15
.packFeatureSpace . . . . .	16
.removeEdge . . . . .	17
.subsetSce . . . . .	17
.terminalStateFromMarkovChain . . . . .	18
.typeCheck . . . . .	18
.waypiontsPerCol . . . . .	19
chainAssign . . . . .	19
dandelionR . . . . .	20
demo_airr . . . . .	21
demo_sce . . . . .	21
differentiationProbabilities . . . . .	22
formatVdj . . . . .	23
markovProbability . . . . .	23
miloUmap . . . . .	25
projectProbability . . . . .	26
projectPseudotimeToCell . . . . .	27
sce_vdj . . . . .	29

<code>.addColData</code>	3
<code>setupVdjPseudobulk</code> . . . . .	30
<code>splitCTgene</code> . . . . .	33
<code>vdjPseudobulk</code> . . . . .	33

**Index** **36**

`.addColData`                    *add the calculated probability to the colData*

**Description**

add the calculated probability to the colData

**Usage**

```
.addColData(probabilities_proj, terminal_state, milo, verbose)
```

**Arguments**

<code>probabilities_proj</code>	the probabilities need to be stored
<code>terminal_state</code>	Integer. The index of the terminal state in the Markov chain, passed from markovProbability
<code>milo</code>	the milo object provided by user
<code>verbose</code>	logical, print warnings.

**Value**

a Milo object with probabilities and pseudotime in its colData slot

`.allowedChain`                    *filtering cell without allowed chain status*

**Description**

filtering cell without allowed chain status

**Usage**

```
.allowedChain(sce, allowed_chain_status, verbose)
```

**Arguments**

<code>sce</code>	SingleCellExperiment object input
<code>allowed_chain_status</code>	the chain needs to be retain, passed from setupVdjPseudobulk
<code>verbose</code>	logical, print messages. Default is TRUE.

**Value**

SingleCellExperiment object with allowed chain status

---

`.calDif` *function help to reconstruct diffusion distance using Reduce*

---

### Description

function help to reconstruct diffusion distance using Reduce

### Usage

```
.calDif(dfm, i, j, eigenvector, lambda_t, K)
```

### Arguments

<code>dfm</code>	an <code>nrow(eigenvectors) x nrow(eigenvectors)</code> matrix need to be filled with diffusion distance with iteration
<code>i</code>	integer the row selected in this iteration
<code>j</code>	integer the col selected in this iteration
<code>eigenvector</code>	numeric vector, the eigenvector from diffusion map
<code>lambda_t</code>	eigenvalues to the power of <code>t</code> (diffusion time)
<code>K</code>	The number of the eigenvectors to be used in calculation

### Value

updated diffusion distance matrix after one iteration

---

`.classCheck` *.classCheck*

---

### Description

check whether the input is with the correct class

### Usage

```
.classCheck(input, must)
```

### Arguments

<code>input</code>	the input need to be check
<code>must</code>	the type we need

### Value

whether or not the input is the correct class

---

*.collapse\_nested\_list* *Collapse a nested list*

---

**Description**

Collapse a nested list

**Usage**

```
.collapse_nested_list(input_list)
```

**Arguments**

*input\_list*      input nested list.

**Value**

collapsed list

---

*.constructMarkovChain* *.constructMarkovChain*

---

**Description**

Markov chain construction

**Usage**

```
.constructMarkovChain(wp_data, knn., pseudotime, waypoints, vb)
```

**Arguments**

*wp\_data*            Multi scale data of the waypoints  
*knn.*                Number of nearest neighbors for graph construction  
*pseudotime*        pseudotime ordering of cells  
*waypoints*          integer vector, index of selected waypoint used to  
*vb*                  whether to print messages

**Value**

transition matrix of the markov chain

---

`.determExtractColN`     *determine the columns in the colData where the main VDJ information is stored*

---

**Description**

determine the columns in the colData where the main VDJ information is stored

**Usage**

```
.determExtractColN(extract_cols, mode_option, milo)
```

**Arguments**

<code>extract_cols</code>	names of columns in the colData where the main VDJ information stores, passed from vdjPseudobulk
<code>mode_option</code>	Specifies the mode for extracting V(D)J genes
<code>milo</code>	Milo or SingleCellExperiment object provided by user

**Value**

a character vector stores the names of columns in the colData where the main VDJ information stores

---

`.determineMultiscaleSpace`  
*.determineMultiscaleSpace*

---

**Description**

`.determineMultiscaleSpace`

**Usage**

```
.determineMultiscaleSpace(diffusionmap, n_eigs = NULL)
```

**Arguments**

<code>diffusionmap</code>	DiffusionMap object
<code>n_eigs</code>	integer, default is NULL. Number of eigen vectors to use. <ul style="list-style-type: none"> <li>If is not specified, the number of eigen vectors will be determined using the eigen gap.</li> </ul>

**Value**

dataframe

---

*.determTerminal*      *.determTerminal*

---

**Description**

function in Reduce to provide waypoints

**Usage**

`.determTerminal(terminal_states, i, dm_boudaries, wp_data)`

**Arguments**

`terminal_states`      integer vector to store the generated waypoint index  
`i`      iteration index  
`dm_boudaries`      index of the maxium or minium value of transition matrix per row  
`wp_data`      Multi scale data of the waypoints

**Value**

integer vector store the index of waypoints serve as terminal state

---

*.extractVdj*      *Specify the columns which store VDJ information, and extract the main chain from it*

---

**Description**

Specify the columns which store VDJ information, and extract the main chain from it

**Usage**

`.extractVdj(sce, extract_cols, mode_option, verbose)`

**Arguments**

`sce`      SingleCellExperiment object input  
`extract_cols`      The setupVdjPseutobulk transfered parameter given by user to specify the VDJ information columns  
`mode_option`      see document of setupVdjPseudobulk for detailed explanation  
`verbose`      logical, print messages. Default is TRUE.

**Value**

SingleCellExperiment objects with column stores the information of the main VDJ information in colData slot

---

```
.featureSpaceConstruct
```

*Construct VDJ feature space*

---

**Description**

Construct VDJ feature space

**Usage**

```
.featureSpaceConstruct(milo, extract_cols, pbs)
```

**Arguments**

milo	Milo or SingleCellExperiment object provided by user
extract_cols	columns of names where to extract the VDJ information
pbs	cell x pseudobulk adjacent matrix

**Value**

constructed feature space

---

```
.filterCells
```

*.filterCells*

---

**Description**

Helper function that identifies filter\_pattern hits in determined column of sce, and then either removes the offending cells or masks the matched values with a uniform value of '(column's name)\_missing'

**Usage**

```
.filterCells(
  sce,
  col_n,
  filter_pattern = "", |None|No_contig",
  remove_missing = TRUE
)
```

**Arguments**

sce	SingleCellExperiment object, adata in python data after combineTCR, contain both vdj and seq
col_n	mode for extraction the V(D)J genes.
filter_pattern	character string, optional ', None No_contig' by default
remove_missing	bool, True by default <ul style="list-style-type: none"> <li>• If TRUE, will remove cells with contigs matching the filter from the object.</li> <li>• If FALSE, will mask them with a uniform value dependent on the column name.</li> </ul>



### Value

filtered SingleCellExperiment object according to the parameter.

---

.filterProductivity     *filer out cell with unproductive chain*

---

### Description

filer out cell with unproductive chain

### Usage

```
.filterProductivity(  
  sce,  
  mode_option,  
  productive_cols,  
  productive_vj,  
  productive_vdj,  
  verbose  
)
```

### Arguments

sce	SingleCellExperiment input
mode_option	check setupVdjPseudobulk for detailed explanation
productive_vj	If TRUE, retains cells where the main VJ chain is productive.
productive_vdj	If TRUE, retains cells where the main VDJ chain is productive.
verbose	logical, print messages. Default is TRUE.

### Value

SingleCellExperiment object after filtering on productive chain

---

.filterUnmapped     *Filter out cell with unclear mapping in VDJ information*

---

### Description

Filter out cell with unclear mapping in VDJ information

**Usage**

```
.filterUnmapped(
  sce,
  mode_option,
  check_vj_mapping,
  check_vdj_mapping,
  main_cols,
  check_extract_cols_mapping,
  remove_missing,
  verbose
)
```

**Arguments**

sce	SingleCellExperiment object input
mode_option	see document of setupVdjPseudobulk for explanation
check_vj_mapping	logical vector to set whether to check V and J gene in VJ chain, passed from setupVdjPseudobulk
check_vdj_mapping	logical vector to set whether to check V, D and J gene in VDJ chain, passed from setupVdjPseudobulk
main_cols	column names in colData in which The information of main chain stores
check_extract_cols_mapping	character vector,the names of columns that needs to be checked, passed from setupVdjPseudobulk
remove_missing	option for removing the unclear mappin or just mask it, passed from setupVdjPseudobulk
verbose	logical, print messages.

**Value**

filtered SingleCellExperiment object

---

*.findNewWaypoints*      *function used in Reduce to find new waypoint in an iteration*

---

**Description**

function used in Reduce to find new waypoint in an iteration

**Usage**

```
.findNewWaypoints(iterdists, k, vecs, ind, datas)
```

### Arguments

iterdists	a list containing both waypoints detected in the former iterations and the distance matrix used to find waypoints
k	the iteration number
vecs	a numeric vector used to calculate distance of waypoints to each points
ind	colnames

### Value

a list containing updated distance matrix and new waypoints

---

.generateExtractColumn

*Check whether the columns with specified names exist, if not, create them with CTgene columns*

---

### Description

Check whether the columns with specified names exist, if not, create them with CTgene columns

### Usage

```
.generateExtractColumn(sce, extract_cols, verbose)
```

### Arguments

sce	SingleCellExperiment object input
extract_cols	column names we aim to extract information from
verbose	logical, print messages. Default is TRUE.

### Value

SingleCellExperiment with columns containing VDJ information in the names we've specified.

---

.generateExtractName *Generate the name of columns with given parameter*

---

### Description

Generate the name of columns with given parameter

### Usage

```
.generateExtractName(sce, mode_option, verbose)
```

**Arguments**

sce	SingleCellExperiment object input
mode_option	see document of setupVdjPseudobulk for explanation
verbose	logical, print messages. Default is TRUE.

**Value**

a vector of colnames we need to perform main chain extraction

---

<i>.getPbs</i>	<i>.getPbs</i>
----------------	----------------

---

**Description**

Helper function to ensure we have cells by pseudobulks matrix which we can use for pseudobulking.

**Usage**

```
.getPbs(pbs, col_to_bulk, milo, verbose = TRUE)
```

**Arguments**

pbs	pbs parameter provided by vdjPseudobulk(), cells by pseudobulks matrix or NULL
col_to_bulk	col_to_bulk parameter provided by vdjPseudobulk(), column's name of colData from milo
milo	SingleCellExperiment object
verbose	logical, whether to print messages

**Value**

a cell x pseudobulk matrix

---

<i>.getPbsCol</i>	<i>.getPbsCol</i>
-------------------	-------------------

---

**Description**

Helper function to create the new pseudobulk object's coldata.

**Usage**

```
.getPbsCol(pbs, col_to_take, milo)
```

**Arguments**

pbs	dgeMatrix, cell x pseudobulk binary matrix
col_to_take	character vector, names of colData of milo that need to be processed
milo	Milo or SingleCellExperiment object

**Value**

`pbs_col`, a DataFrame which will be passed to the new `SingleCellExperiment` object as `colData` of `vdj` x pseudobulk assays

---

<code>.getPbsPerCol</code>	<code>.getPbsPerCol</code>
----------------------------	----------------------------

---

**Description**

function used in Reduce to get the PbsCol

**Usage**

```
.getPbsPerCol(pbs.col, anno.col, milo, pbs)
```

**Arguments**

<code>pbs.col</code>	DataFrame object used to store the result of each iteration
<code>anno.col</code>	colname where to generate the metadata from
<code>milo</code>	<code>milo</code> or <code>SingleCellExperiment</code> objects provided by user
<code>pbs</code>	<code>dgeMatrix</code> , cell x pseudobulk binary matrix

**Value**

DataFrame object, serve as part of metadata of the new `milo` object

---

<code>.KNNind</code>	<i>Calculate the weight adjacent matricks of knn graph and its index</i>
----------------------	--

---

**Description**

Calculate the weight adjacent matricks of knn graph and its index

**Usage**

```
.KNNind(wp_data, knn.)
```

**Arguments**

<code>wp_data</code>	Multi scale data of the waypoints
<code>knn.</code>	Number of nearest neighbors for graph construction

**Value**

a list containing the weight adjacent matrix and index

---

<code>.maxMinSampling</code>	<i>.maxMinSampling</i>
------------------------------	------------------------

---

**Description**

function for max min sampling of waypoints

**Usage**

```
.maxMinSampling(datas, num_waypoints, verbose = TRUE)
```

**Arguments**

<code>datas</code>	data matrix along which to sample the waypoints, usually diffusion components
<code>num_waypoints</code>	number of waypoints to sample
<code>verbose</code>	logical, print progress

**Value**

Series representing the sampled waypoints

---

<code>.minMaxScale</code>	<i>minMaxScale</i>
---------------------------	--------------------

---

**Description**

scale the value to range 0 to 1

**Usage**

```
.minMaxScale(data)
```

**Arguments**

<code>data</code>	dataframe need to be scale
-------------------	----------------------------

**Value**

scaled value

---

.normalizeFeatureSpace  
*Normalize Feature Space*

---

### Description

Make sure the sum of each V, D, and J gene within a pseudobulk equals to 1

### Usage

```
.normalizeFeatureSpace(  
  pseudo_vdj_feature,  
  extract_cols,  
  min_count,  
  renormalize,  
  milo  
)
```

### Arguments

pseudo_vdj_feature	constructed feature space
extract_cols	names of columns to extract the VDJ information
min_count	the minim count of a V/D/J gene
renormalize	Whether to renormalize the matrix
milo	Milo or SingleCellExperiment object provided by user

### Value

normalized VDJ feature space

---

.normalizePerVDJ      *function to normalize a specific kind of VDJ gene in feature space*

---

### Description

function to normalize a specific kind of VDJ gene in feature space

### Usage

```
.normalizePerVDJ(  
  pseudo_vdj_feature,  
  col_n,  
  renormalize,  
  define.mask,  
  milo,  
  min_count  
)
```

**Arguments**

<code>pseudo_vdj_feature</code>	constructed feature space
<code>col_n</code>	name of column to extract the VDJ information
<code>renormalize</code>	Whether to renormalize the matrix
<code>define.mask</code>	logical vector determine whether the V/D/J gene should be masked when normalizing
<code>mi lo</code>	Milo or SingleCellExperiment object provided by user
<code>min_count</code>	the minim count of a V/D/J gene

**Value**

feature space normalized on specifed V/D/J gene

---

`.packFeatureSpace`      *Pack the normalized feature space into new Milo object*

---

**Description**

The metadata will derived from the original milo

**Usage**

```
.packFeatureSpace(pbs, col_to_take, milo, pseudo_vdj_feature)
```

**Arguments**

<code>pbs</code>	cell x pseudobulk adjacent matrix
<code>col_to_take</code>	Optional character or vector of characters. Specifies names of colData of milo that need to identify the most common value for each pseudobulk
<code>mi lo</code>	Milo or SingleCellExperiment object provided by user
<code>pseudo_vdj_feature</code>	VDJ feature space

**Value**

Milo object with VDJ feature space stored in its assay



---

.removeEdge	<i>function used in Reduce to remove KNN's backward edges except for edges that are within the computed standard deviation</i>
-------------	--

---

**Description**

function used in Reduce to remove KNN's backward edges except for edges that are within the computed standard deviation

**Usage**

```
.removeEdge(Knn, i, rem_edges)
```

**Arguments**

Knn	weight KNN adjacent matrix
i	the iteration number
rem_edges	the edges that need to be removes

**Value**

an updated matrix after one round of iteration

---

.subsetSce	<i>Subset sce with given parameter</i>
------------	--

---

**Description**

Subset sce with given parameter

**Usage**

```
.subsetSce(sce, subsetby, groups, verbose)
```

**Arguments**

sce	SingleCellExperiment object input
subsetby	subsetby Character. Name of a colData column for subsetting. given by setupVdjPseudobulk.
groups	Character vector. Specifies the subset condition for filtering. given by setupVdjPseudobulk.
verbose	logical, print messages. Default is TRUE.

**Value**

subsetting SingleCellExperiment object

---

`.terminalStateFromMarkovChain`

*Determine terminal states using Markov chain if end states are not provided.*

---

### Description

Determine terminal states using Markov chain if end states are not provided.

### Usage

```
.terminalStateFromMarkovChain(
  Transmat,
  wp_data,
  pseudotime,
  waypoints,
  verbose
)
```

### Arguments

<code>Transmat</code>	Transition matrix
<code>wp_data</code>	Multi scale data of the waypoints
<code>pseudotime</code>	numeric vector, pseudotime of each pseudobulk
<code>waypoints</code>	integer vector, waypoint selected to construct markov chain.
<code>verbose</code>	Boolean, whether to print messages/warnings.

### Value

`terminal_state`

---

`.typeCheck`

*.typeCheck*

---

### Description

check whether the input has the correct type

### Usage

```
.typeCheck(input, must)
```

### Arguments

<code>input</code>	the input need to be check
<code>must</code>	the type we need

### Value

whether or not the input is the correct type

---

.waypiontsPerCol      *find the waypoints according to certain columns of data*

---

**Description**

find the waypoints according to certain columns of data

**Usage**

.waypiontsPerCol(waypoints, ind, datas, no.iterations)

**Arguments**

waypoints      integer vector used to store waypoints  
ind              columns' colnames  
datas            scaled diffusionmap

**Value**

a numeric vector containing waypoints' index

---

chainAssign      *Assign the V(D)J gene to the right chain.*

---

**Description**

Assign the V(D)J gene to the right chain.

**Usage**

chainAssign(vec, num)

**Arguments**

vec              vector of V(D)J genes to assign to the right chain.  
num              number of genes to return. should be 2(vj) or 3(vdj)

**Value**

list contain vector of VJ + VDJ of the cell input

---

dandelionR

*dandelionR: Single-cell immune repertoire trajectory analysis*

---

## Description

dandelionR is an R package for performing single-cell immune repertoire trajectory analysis, based on the original python implementation. It provides the necessary functions to interface with scRepertoire and a custom implementation of an absorbing Markov chain for pseudotime inference, inspired by the Palantir Python package.

## Main functions

- `setupVdjPseudobulk`: Preprocess V(D)J Data for Pseudobulk Analysis.
- `vdjPseudobulk`: Generate Pseudobulk V(D)J Feature Space.
- `markovProbability`: Markov Chain Construction and Probability Calculation.
- `projectPseudotimeToCell`: Project Pseudotime and Branch Probabilities to Single Cells.

## Vignettes

See the package vignettes for detailed workflows: `vignette('dandelionR')`

## Installation

To install from Bioconductor, use:

```
if (!requireNamespace('BiocManager', quietly = TRUE))
  install.packages('BiocManager')
BiocManager::install('dandelionR')
```

## Author(s)

**Maintainer:** Kelvin Tuong <z.tuong@uq.edu.au> ([ORCID](#))

Authors:

- Jiawei Yu <jiawei.yu@uq.edu.au> ([ORCID](#))
- Nicholas Borcharding <borcherding@wustl.edu> ([ORCID](#))

## See Also

Useful links:

- <https://www.github.com/tuonglab/dandelionR/>
- Report bugs at <https://www.github.com/tuonglab/dandelionR/issues>

---

`demo_airr`*Example AIRR Dataset for V(D)J Analysis*

---

### Description

The `demo_airr` object is a list of AIRR data frames from a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.

This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.

For details, see the original publication at <https://www.nature.com/articles/s41587-023-01734-7>.

The original files are available at <https://github.com/zktuong/dandelion-demo-files>.

### Usage

```
data(demo_airr)
```

### Format

A `SingleCellExperiment` object with the following slots:

`list` List of `DataFrames` containing the standardised AIRR data for each sample.

For information of AIRR rearrangements, see the AIRR Community standards at <https://docs.airr-community.org/>.

### Source

Suo et al., 2024, *Nature Biotechnology*.

<https://www.nature.com/articles/s41587-023-01734-7>.

### Examples

```
data(demo_airr)
```

---

`demo_sce`*Example SCE Dataset that does not contain V(D)J information*

---

### Description

The `demo_sce` object is a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.

This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.

For details, see the original publication at <https://www.nature.com/articles/s41587-023-01734-7>.

The original Lymphoid cells data in h5ad format is available at <https://developmental.cellatlas.io/fetal-immune>.

### Usage

```
data(demo_sce)
```

**Format**

A SingleCellExperiment object with the following slots:

`colData` A minimal DataFrame containing metadata about each sample, corresponding to obs in AnnData (Python). The following columns are relevant for vignette usage:

`anno_lvl_2_final_clean` Cell type annotations.

`int_colData` A DataFrame containing additional assay metadata important for further analysis. Includes:

- `X_scvi`: A dimensionality reduction matrix from the scVI model.
- `UMAP`: A UMAP reduction matrix.

**Source**

Suo et al., 2024, *Nature Biotechnology*.

<https://www.nature.com/articles/s41587-023-01734-7>.

**Examples**

```
data(demo_sce)
```

---

differentiationProbabilities

*Compute Branch Probabilities Using Markov Chain*

---

**Description**

This function calculates branch probabilities for differentiation trajectories based on a Markov chain constructed from waypoint data and pseudotime ordering.

**Usage**

```
differentiationProbabilities(
  wp_data,
  terminal_states = NULL,
  knn = 30L,
  pseudotime,
  waypoints,
  verbose = TRUE
)
```

**Arguments**

<code>wp_data</code>	A multi-scale data matrix or data frame representing the waypoints.
<code>terminal_states</code>	Integer vector. Indices of the terminal states. Default is NULL.
<code>knn</code>	Integer. Number of nearest neighbors for graph construction. Default is 30L.
<code>pseudotime</code>	Numeric vector. Pseudotime ordering of cells.
<code>waypoints</code>	Integer vector. Indices of selected waypoints used to construct the Markov chain.
<code>verbose</code>	Boolean, whether to print messages/warnings.

**Value**

A numeric matrix or data frame containing branch probabilities for each waypoint.

---

formatVdj	<i>Change the format of splitCTgene output.</i>
-----------	---

---

**Description**

Change the format of splitCTgene output.

**Usage**

```
formatVdj(gene_list)
```

**Arguments**

gene\_list      list containing the output from splitCTgene.

**Value**

list contain vector of VJ + VDJ information of the cell input

---

markovProbability	<i>Markov Chain Construction and Probability Calculation</i>
-------------------	--

---

**Description**

This function preprocesses data, constructs a Markov chain, and calculates transition probabilities based on pseudotime information.

**Usage**

```
markovProbability(
  milo,
  diffusionmap,
  terminal_state = NULL,
  root_cell,
  knn = 30L,
  diffusiontime = NULL,
  pseudotime_key = "pseudotime",
  scale_components = TRUE,
  num_waypoints = 500,
  n_eigs = NULL,
  verbose = TRUE
)
```

**Arguments**

<code>milo</code>	A Milo or SingleCellExperiment object. This object should have pseudotime stored in <code>colData</code> , which will be used to calculate probabilities. If pseudotime is available in <code>milo</code> , it takes precedence over the value provided through the <code>diffusiontime</code> parameter.
<code>diffusionmap</code>	A DiffusionMap object corresponding to the <code>milo</code> object. Used for Markov chain construction.
<code>terminal_state</code>	Integer. The index of the terminal state in the Markov chain.
<code>root_cell</code>	Integer. The index of the root state in the Markov chain.
<code>knn</code>	Integer. The number of nearest neighbors for graph construction. Default is 30L.
<code>diffusiontime</code>	Numeric vector. If pseudotime is not stored in <code>milo</code> , this parameter can be used to provide pseudotime values to the function.
<code>pseudotime_key</code>	Character. The name of the column in <code>colData</code> that contains the inferred pseudotime.
<code>scale_components</code>	Logical. If TRUE, the components will be scaled before constructing the Markov chain. Default is FALSE.
<code>num_waypoints</code>	Integer. The number of waypoints to sample when constructing the Markov chain. Default is 500L.
<code>n_eigs</code>	integer, default is NULL. Number of eigen vectors to use. <ul style="list-style-type: none"> <li>• If is not specified, the number of eigen vectors will be determined using the eigen gap.</li> </ul>
<code>verbose</code>	Logical. If TRUE, print progress. Default is TRUE.

**Value**

`milo` or `SingleCellExperiment` object with pseudotime, probabilities in its `colData`

**Examples**

```
data(sce_vdj)
# downsample to first 2000 cells
sce_vdj <- sce_vdj[, 1:2000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
  already.productive = FALSE,
  allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
set.seed(100)
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
  k = 50, d = 20,
  reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
  reduced_dims = "X_scvi",
  d = 20
)

# Construct Pseudobulked VDJ Feature Space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lvl_2_final_clean")
```



```

pb.milo <- scater::runPCA(pb.milo, assay.type = "Feature_space")

# Define root and branch tips
pca <- t(as.matrix(SingleCellExperiment::reducedDim(pb.milo, type = "PCA")))
branch.tips <- c(which.min(pca[, 2]), which.max(pca[, 2]))
names(branch.tips) <- c("CD8+T", "CD4+T")
root <- which.min(pca[, 1])

# Construct Diffusion Map
dm <- destiny::DiffusionMap(t(pca), n_pcs = 10, n_eigs = 5)
dif.pse <- destiny::DPT(dm, tips = c(root, branch.tips), w_width = 0.1)

# Markov Chain Construction
pb.milo <- markovProbability(
  milo = pb.milo,
  diffusionmap = dm,
  diffusiontime = dif.pse[[paste0("DPT", root)]],
  terminal_state = branch.tips,
  root_cell = root,
  pseudotime_key = "pseudotime"
)

```

---

miloUmap

*Perform UMAP on the Adjacency Matrix of a Milo Object*


---

## Description

This function uses `uwot::umap` to perform UMAP dimensionality reduction on the adjacency matrix of the KNN graph in a Milo object.

## Usage

```

miloUmap(
  milo,
  slot_name = "UMAP_knngraph",
  n_neighbors = 50L,
  metric = "euclidean",
  min_dist = 0.3,
  ...
)

```

## Arguments

<code>milo</code>	the milo object with knn graph that needed to conduct umap on.
<code>slot_name</code>	character, with default 'UMAP_knngraph'. <ul style="list-style-type: none"> <li>The slot name in <code>reduceDim</code> where the result store</li> </ul>
<code>n_neighbors</code>	integer, with default 50L. <ul style="list-style-type: none"> <li>the size of local neighborhood (in terms of number of neighboring sample points) used for manifold approximation.</li> <li>Here, the goal is to create large enough neighborhoods to capture the local manifold structure to allow for hypersampling.</li> </ul>

metric character, with default 'euclidean'

- the choice of metric used to measure distance to find nearest neighbors. Default is 'euclidean'.

min\_dist numeric, with default 0.3

- the minimum distance between points in the low dimensional space

... other parameters passed to uwot::umap

**Value**

milo object with umap reduction

**Examples**

```
data(sce_vdj)
# downsample to just 1000 cells
sce_vdj <- sce_vdj[, 1:1000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
  already.productive = FALSE,
  allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
  k = 50, d = 20,
  reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
  reduced_dims = "X_scvi", d = 20
)

# Construct UMAP on Milo Neighbor Graph
milo_object <- miloUmap(milo_object)
```

---

projectProbability      *Project Probabilities from Markov Chain to Pseudobulks*

---

**Description**

This function projects probabilities calculated from a Markov chain onto each pseudobulk based on a diffusion distance matrix.

**Usage**

```
projectProbability(
  diffusionmap,
  waypoints,
  probabilities,
  t = 1,
  verbose = TRUE
)
```

**Arguments**

diffusionmap	diffusion map, used to reconstruct diffusion distance matrix
waypoints	Integer vector. Indices of the waypoints used in the Markov chain.
probabilities	Numeric vector. Probabilities associated with the waypoints, calculated from the Markov chain.
t	Numeric. The diffusion time to be used in the projection.
verbose	Boolean, whether to print messages/warnings.

**Value**

each pseudobulk's probabilities

---

projectPseudotimeToCell

*Project Pseudotime and Branch Probabilities to Single Cells*

---

**Description**

This function projects pseudotime and branch probabilities from pseudobulk data to single-cell resolution (milo). The results are stored in the colData of the milo object.

**Usage**

```
projectPseudotimeToCell(
  milo,
  pb_milo,
  term_states = NULL,
  pseudotime_key = "pseudotime",
  suffix = "",
  verbose = TRUE
)
```

**Arguments**

milo	A SingleCellExperiment or Milo object. Represents single-cell data where pseudotime and branch probabilities will be projected.
pb_milo	A pseudobulk Milo object. Contains aggregated branch probabilities and pseudotime information to be transferred to single cells.
term_states	Named vector of terminal states, with branch probabilities to be transferred. The names should correspond to branches of interest.
pseudotime_key	Character. The column name in colData of pb_milo that contains the pseudotime information which was used in the markovProbability function. Default is "pseudotime".
suffix	Character. A suffix to be added to the new column names in colData. Default is an empty string ('').
verbose	Boolean, whether to print messages/warnings.

**Value**

subset of milo or SingleCellExperiment object where cell that do not belong to any neighbourhood are removed and projected pseudotime information stored colData

**Examples**

```

data(sce_vdj)
# downsample to first 2000 cells
sce_vdj <- sce_vdj[, 1:2000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
  already.productive = FALSE,
  allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
set.seed(100)
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
  k = 50, d = 20,
  reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
  reduced_dims = "X_scvi",
  d = 20
)

# Construct Pseudobulked VDJ Feature Space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lvl_2_final_clean")
pb.milo <- scater::runPCA(pb.milo, assay.type = "Feature_space")

# Define root and branch tips
pca <- t(as.matrix(SingleCellExperiment::reducedDim(pb.milo, type = "PCA")))
branch.tips <- c(which.min(pca[, 2]), which.max(pca[, 2]))
names(branch.tips) <- c("CD8+T", "CD4+T")
root <- which.min(pca[, 1])

# Construct Diffusion Map
dm <- destiny::DiffusionMap(t(pca), n_pcs = 10, n_eigs = 5)
dif.pse <- destiny::DPT(dm, tips = c(root, branch.tips), w_width = 0.1)

# Markov Chain Construction
pb.milo <- markovProbability(
  milo = pb.milo,
  diffusionmap = dm,
  diffusiontime = dif.pse[[paste0("DPT", root)]],
  terminal_state = branch.tips,
  root_cell = root,
  pseudotime_key = "pseudotime"
)
# Project Pseudobulk Data
projected_milo <- projectPseudotimeToCell(
  milo_object,
  pb.milo,
  branch.tips,
  pseudotime_key = "pseudotime"
)

```

---

`sce_vdj`*Example Dataset for V(D)J Analysis*

---

## Description

The `sce_vdj` object is a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.

This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.

For details, see the original publication at <https://www.nature.com/articles/s41587-023-01734-7>.

## Usage

```
data(sce_vdj)
```

## Format

A `SingleCellExperiment` object with the following slots:

`colData` A `DataFrame` containing metadata about each sample, corresponding to `obs` in `AnnData` (Python). The following columns are relevant for vignette usage:

`productive_(mode)_VDJ`, `productive_(mode)_VJ` Factors indicating whether the heavy or light chain is productive. `mode` refers to the extraction mode for V(D)J genes and can be one of:

- 'abT': TCR alpha-beta
- 'gdT': TCR gamma-delta
- 'B': BCR

`Gene segment fields` Gene segment annotations with column names in the format `(v/d/j)_call_(mode)_(VDJ/chain)`.  
Examples include:

- `v_call_abT_VDJ`: V gene for TCR alpha-beta VDJ recombination
- `d_call_abT_VJ`: D gene for TCR alpha-beta VJ recombination

`chain_status` A factor describing the receptor chain's status.

`anno_lvl_2_final_clean` Cell type annotations.

`int_colData` A `DataFrame` containing additional assay metadata important for further analysis.  
Includes:

- `X_scvi`: A dimensionality reduction matrix from the scVI model.
- `UMAP`: A UMAP reduction matrix.

## Source

Suo et al., 2024, *Nature Biotechnology*.

<https://www.nature.com/articles/s41587-023-01734-7>.

## Examples

```
data(sce_vdj)
```

---

 setupVdjPseudobulk     *Preprocess V(D)J Data for Pseudobulk Analysis*


---

### Description

This function preprocesses single-cell V(D)J sequencing data for pseudobulk analysis. It filters data based on productivity and chain status, subsets data, extracts main V(D)J genes, and removes unmapped entries.

### Usage

```
setupVdjPseudobulk(
  sce,
  mode_option = c("abT", "gdT", "B"),
  already.productive = TRUE,
  productive_cols = NULL,
  productive_vj = TRUE,
  productive_vdj = TRUE,
  allowed_chain_status = NULL,
  subsetby = NULL,
  groups = NULL,
  extract_cols = NULL,
  filter_unmapped = TRUE,
  check_vj_mapping = c(TRUE, TRUE),
  check_vdj_mapping = c(TRUE, FALSE, TRUE),
  check_extract_cols_mapping = NULL,
  remove_missing = TRUE,
  verbose = TRUE
)
```

### Arguments

sce	A SingleCellExperiment object. V(D)J data should be contained in colData for filtering.
mode_option	Optional character. Specifies the mode for extracting V(D)J genes. If NULL, extract_cols must be specified. Default is NULL.
already.productive	Logical. Whether the data has already been filtered for productivity. If TRUE, skips productivity filtering. Default is FALSE.
productive_cols	Character vector. Names of colData columns used for productivity filtering. Default is NULL.
productive_vj	Logical. If TRUE, retains cells where the main VJ chain is productive. Default is TRUE.
productive_vdj	Logical. If TRUE, retains cells where the main VDJ chain is productive. Default is TRUE.
allowed_chain_status	Character vector. Specifies chain statuses to retain. Valid options include `c('single pair', 'Extra pair', 'Extra pair-exception', 'Orphan VDJ', 'Orphan VDJ-exception')`. Default is NULL.

subsetby	Character. Name of a colData column for subsetting. Default is NULL.
groups	Character vector. Specifies the subset condition for filtering. Default is NULL.
extract_cols	Character vector. Names of colData columns where V(D)J information is stored, used instead of the standard columns. Default is NULL.
filter_unmapped	Logic. Whether to filter unmapped data. Default is TRUE.
check_vj_mapping	Logic vector. Whether to check for VJ mapping. Default is c(TRUE, TRUE). <ul style="list-style-type: none"> <li>• If the first element is TRUE, function will filter the unmapped data in V gene of the VJ chain</li> <li>• If the second element is TRUE, function will filter the unmapped data in J gene of the VJ chain</li> </ul>
check_vdj_mapping	Logic vector. Specifies columns to check for VDJ mapping. Default is c(TRUE, FALSE, 'TRUE'). <ul style="list-style-type: none"> <li>• If the first element is TRUE, function will filter the unmapped data in V gene of the VDJ chain</li> <li>• If the second element is TRUE, function will filter the unmapped data in D gene of the VDJ chain</li> <li>• If the third element is TRUE, function will filter the unmapped data in J gene of the VDJ chain</li> </ul>
check_extract_cols_mapping	Character vector. Specifies columns related to extract_cols for mapping checks. Default is NULL.
remove_missing	Logical. If TRUE, removes cells with contigs matching the filter. If FALSE, masks them with uniform values. Default is TRUE.
verbose	Logical. Whether to print messages. Default is TRUE.

## Details

The function performs the following preprocessing steps:

- **Productivity Filtering:**
  - Skipped if `already.productive = TRUE`.
  - Filters cells based on productivity using `productive_cols` or standard colData columns named `productive_{mode_option}_{type}` (where `type` is 'VDJ' or 'VJ').
  - *mode\_option*
    - \* function will check colData(s) named `productive_{mode_option}_{type}`, where `type` should be 'VDJ' or 'VJ' or both, depending on values of `productive_vj` and `productive_vdj`.
    - \* If set as NULL, the function needs the option 'extract\_cols' to be specified
  - *productive\_cols*
    - \* must be specified when productivity filtering is need to conduct and `mode_option` is NULL.
    - \* where VDJ/VJ information is stored so that this will be used instead of the standard columns.
  - *productive\_vj, productive\_vdj*
    - \* If TRUE, cell will only be kept if the main V(D)J chain is productive
- **Chain Status Filtering:**

- Retains cells with chain statuses specified by `allowed_chain_status`.
- **Subsetting:**
  - Conducted only if both `subsetby` and `groups` are provided.
  - Retains cells matching the groups condition in the `subsetby` column.
- **Main V(D)J Extraction:**
  - Uses `extract_cols` to specify custom columns for extracting V(D)J information.
- **Unmapped Data Filtering:**
  - decided to removes or masks cells based on `filter_unmapped`.
  - Checks specific columns for unclear mappings using `check_vj_mapping`, `check_vdj_mapping`, or `check_extract_cols_mapping`.
  - *filter\_unmapped*
    - \* pattern to be filtered from object.
    - \* If is set to be NULL, the filtering process will not start
  - *check\_vj\_mapping, check\_vdj\_mapping*
    - \* only `colData` specified by these arguments (`check_vj_mapping` and `check_vdj_mapping`) will be checked for unclear mappings
  - *check\_extract\_cols\_mapping, related to extract\_cols*
    - \* Only `colData` specified by the argument will be checked for unclear mapping, the `colData` should first specified by `extract_cols`
  - `remove_missing`
    - \* If TRUE, will remove cells with contigs matching the filter from the object.
    - \* If FALSE, will mask them with a uniform value dependent on the column name.

## Value

filtered `SingleCellExperiment` object

## Examples

```
# load data
data(sce_vdj)
# check the dimension
dim(sce_vdj)
# filtered the data
sce_vdj <- setupVdjPseudobulk(
  sce = sce_vdj,
  mode_option = "abT", # set the mode to alpha-beta TCR
  allowed_chain_status = c("Single pair", "Extra pair"),
  already.productive = FALSE
) # need to filter the unproductive cells
# check the remaining dim
dim(sce_vdj)
```



---

splitCTgene	<i>Split the V(D)J genes from CTgene column and store them separately.</i>
-------------	--

---

**Description**

Split the V(D)J genes from CTgene column and store them separately.

**Usage**

```
splitCTgene(sce)
```

**Arguments**

sce	SingleCellExperiment object after conducting scRepertoire::combineTCR()
-----	---

**Value**

list contain vector of VJ & VDJ genes from each cell

---

vdjPseudobulk	<i>Generate Pseudobulk V(D)J Feature Space</i>
---------------	--

---

**Description**

This function creates a pseudobulk V(D)J feature space from single-cell data, aggregating V(D)J information into pseudobulk groups. It supports input as either a Milo object or a SingleCellExperiment object.

**Usage**

```
vdjPseudobulk(
  milo,
  pbs = NULL,
  col_to_bulk = NULL,
  extract_cols = c("v_call_abT_VDJ_main", "j_call_abT_VDJ_main", "v_call_abT_VJ_main",
    "j_call_abT_VJ_main"),
  mode_option = c("abT", "gdT", "B"),
  col_to_take = NULL,
  normalise = TRUE,
  renormalize = FALSE,
  min_count = 1L,
  verbose = TRUE
)
```

**Arguments**

miilo	A Milo or SingleCellExperiment object containing V(D)J data.
pbs	Optional. A binary matrix with cells as rows and pseudobulk groups as columns. <ul style="list-style-type: none"> <li>• If miilo is a Milo object, this parameter is not required.</li> <li>• If miilo is a SingleCellExperiment object, either pbs or col_to_bulk must be provided.</li> </ul>
col_to_bulk	Optional character or character vector. Specifies colData column(s) to generate pbs. If multiple columns are provided, they will be combined. Default is NULL. <ul style="list-style-type: none"> <li>• If miilo is a Milo object, this parameter is not required.</li> <li>• If miilo is a SingleCellExperiment object, either pbs or col_to_bulk must be provided.</li> </ul>
extract_cols	Character vector. Specifies column names where V(D)J information is stored. Default is c('v_call_abT_VDJ_main', 'j_call_abT_VDJ_main', 'v_call_abT_VJ_main', 'j_call_abT_VJ_main').
mode_option	Character. Specifies the mode for extracting V(D)J genes. Must be one of c('B', 'abT', 'gdT'). Default is 'abT'. <ul style="list-style-type: none"> <li>• Note: This parameter is considered only when extract_cols = NULL.</li> <li>• If NULL, uses column names such as v_call_VDJ instead of v_call_abT_VDJ.</li> </ul>
col_to_take	Optional character or vector of characters. Specifies names of colData of miilo that need to identify the most common value for each pseudobulk. Default is NULL.
normalise	Logical. If TRUE, scales the counts of each V(D)J gene group to 1 for each pseudobulk. Default is TRUE.
renormalize	Logical. If TRUE, rescales the counts of each V(D)J gene group to 1 for each pseudobulk after removing 'missing' calls. Useful when setupVdjPseudobulk() was run with remove_missing = FALSE. Default is FALSE.
min_count	Integer. Sets pseudobulk counts in V(D)J gene groups with fewer than this many non-missing calls to 0. Default is 1.
verbose	Logical. If TRUE, prints messages and warnings. Default is TRUE.

**Details**

This function aggregates V(D)J data into pseudobulk groups based on the following logic:

- **Input Requirements:**
  - If miilo is a Milo object, neither pbs nor col\_to\_bulk is required.
  - If miilo is a SingleCellExperiment object, the user must provide either pbs or col\_to\_bulk.
- **Normalization:**
  - When normalise = TRUE, scales V(D)J counts to 1 for each pseudobulk group.
  - When renormalize = TRUE, rescales the counts after removing 'missing' calls.
- **Mode Selection:**
  - If extract\_cols = NULL, the function relies on mode\_option to determine which V(D)J columns to extract.
- **Filtering:**
  - Uses min\_count to filter pseudobulks with insufficient counts for V(D)J groups.

**Value**

SingleCellExperiment object

**Examples**

```
data(sce_vdj)
sce_vdj <- setupVdjPseudobulk(sce_vdj,
  already.productive = FALSE,
  allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
  k = 50, d = 20,
  reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
  reduced_dims = "X_scvi",
  d = 20
)

# Construct pseudobulked VDJ feature space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lv1_2_final_clean")
```

# Index

- \* **datasets**
  - demo\_airr, [21](#)
  - demo\_sce, [21](#)
  - sce\_vdj, [29](#)
- \* **internal**
  - .KNNind, [13](#)
  - .addColData, [3](#)
  - .allowedChain, [3](#)
  - .calDif, [4](#)
  - .classCheck, [4](#)
  - .collapse\_nested\_list, [5](#)
  - .constructMarkovChain, [5](#)
  - .determExtractColN, [6](#)
  - .determTerminal, [7](#)
  - .determineMultiscaleSpace, [6](#)
  - .extractVdj, [7](#)
  - .featureSpaceConstruct, [8](#)
  - .filterCells, [8](#)
  - .filterProductivity, [9](#)
  - .filterUnmapped, [9](#)
  - .findNewWaypoints, [10](#)
  - .generateExtractColumn, [11](#)
  - .generateExtractName, [11](#)
  - .getPbs, [12](#)
  - .getPbsCol, [12](#)
  - .getPbsPerCol, [13](#)
  - .maxMinSampling, [14](#)
  - .minMaxScale, [14](#)
  - .normalizeFeatureSpace, [15](#)
  - .normalizePerVDJ, [15](#)
  - .packFeatureSpace, [16](#)
  - .removeEdge, [17](#)
  - .subsetSce, [17](#)
  - .terminalStateFromMarkovChain, [18](#)
  - .typeCheck, [18](#)
  - .waypiontsPerCol, [19](#)
  - chainAssign, [19](#)
  - dandelionR, [20](#)
  - dandelionR-package (dandelionR), [20](#)
  - demo\_airr, [21](#)
  - demo\_sce, [21](#)
  - differentiationProbabilities, [22](#)
  - formatVdj, [23](#)
  - markovProbability, [20, 23](#)
  - miUmap, [25](#)
  - projectProbability, [26](#)
  - projectPseudotimeToCell, [20, 27](#)
  - .calDif, [4](#)
  - .classCheck, [4](#)
  - .collapse\_nested\_list, [5](#)
  - .constructMarkovChain, [5](#)
  - .determExtractColN, [6](#)
  - .determTerminal, [7](#)
  - .determineMultiscaleSpace, [6](#)
  - .extractVdj, [7](#)
  - .featureSpaceConstruct, [8](#)
  - .filterCells, [8](#)
  - .filterProductivity, [9](#)
  - .filterUnmapped, [9](#)
  - .findNewWaypoints, [10](#)
  - .generateExtractColumn, [11](#)
  - .generateExtractName, [11](#)
  - .getPbs, [12](#)
  - .getPbsCol, [12](#)
  - .getPbsPerCol, [13](#)
  - .maxMinSampling, [14](#)
  - .minMaxScale, [14](#)
  - .normalizeFeatureSpace, [15](#)
  - .normalizePerVDJ, [15](#)
  - .packFeatureSpace, [16](#)
  - .removeEdge, [17](#)
  - .subsetSce, [17](#)
  - .terminalStateFromMarkovChain, [18](#)
  - .typeCheck, [18](#)
  - .waypiontsPerCol, [19](#)

sce\_vdj, [29](#)  
setupVdjPseudobulk, [20](#), [30](#)  
splitCTgene, [33](#)  
  
vdjPseudobulk, [20](#), [33](#)