# Package 'gasper'

February 28, 2024

**Type** Package

**Title** Graph Signal Processing

**Version** 1.1.6

**Description** Provides the standard operations for signal processing on graphs: graph Fourier transform, spectral graph wavelet transform, visualization tools. It also implements a data driven method for graph signal denoising/regression, for details see De Loynes, Navarro, Olivier (2019) <arxiv:1906.01882>. The package also provides an interface to the SuiteSparse Matrix Collection, <https://sparse.tamu.edu/>, a large and widely used set of sparse matrix benchmarks collected from a wide range of applications.

**URL** https://github.com/fabnavarro/gasper

**BugReports** https://github.com/fabnavarro/gasper/issues

**License** LGPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**Depends** R (>= 3.5.0)

**Imports** Rcpp, ggplot2, methods, Matrix, RSpectra, httr, curl

**LinkingTo** Rcpp, RcppArmadillo

**Suggests** knitr, kableExtra, rmarkdown, rvest

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Basile de Loynes [aut] (<https://orcid.org/0000-0002-5397-6811>), Fabien Navarro [aut, cre] (<https://orcid.org/0000-0002-4979-2745>), Baptiste Olivier [aut] (<https://orcid.org/0000-0002-5853-0341>)

**Maintainer** Fabien Navarro <fabien.navarro@math.cnrs.fr>

**Repository** CRAN

**Date/Publication** 2024-02-28 11:10:02 UTC

# R topics documented:

adjacency_mat *Compute the Adjacency Matrix of a Gaussian Weighted Graph*

## Description

adjacency_mat calculates the adjacency matrix of a Gaussian weighted graph based on the distance between points in $\mathbb{R}^3$.

## Usage

```
adjacency_mat(
  pts,
  f = function(x) {
     exp(-x^2/8)
 },
  s = 0
)
```

## Arguments

| | |
|---|---|
| pts | Matrix representing the coordinates of N points in $\mathbb{R}^3$. Each row should correspond to a point. |
| f | A scalar potential function. By default, the Gaussian potential $\exp(-x^2/8)$ is used. |
| s | Numeric threshold used to sparsify the adjacency matrix. Any value below this threshold will be set to zero. Default is 0. |

## Details

The function computes pairwise distances between each point in pts and weights the adjacency matrix based on the scalar potential f. The final adjacency matrix can be sparsified by setting values below the threshold s to zero.

## Value

A matrix representing the adjacency matrix of the Gaussian weighted graph.

## See Also

laplacian_mat for calculating the Laplacian matrix, swissroll for generating a Swiss roll dataset.

## Examples

```
pts <- swissroll(N=100, seed=0, a=1, b=4)
W <- adjacency_mat(pts)
```

analysis *Compute the Analysis Operator for a Graph Signal*

## Description

analysis computes the transform coefficients of a given graph signal using the provided frame coefficients.

## Usage

```
analysis(y, tf)
```

## Arguments

| | |
|---|---|
| y | Numeric vector or matrix representing the graph signal to analyze. |
| tf | Numeric matrix of frame coefficients. |

## Details

The analysis operator uses the frame coefficients to transform a given graph signal into its representation in the transform domain. It is defined by the linear map $T_{\mathfrak{F}} : \mathbb{R}^V \to \mathbb{R}^I$. Given a function $f \in \mathbb{R}^V$, the analysis operation is defined as:

$$T_{\mathfrak{F}} f = (\langle f, r_i \rangle)_{i \in I}$$

where $r_i$ are the frame vectors.

The transform is computed as:

$$coef = tf.y$$

## Value

coef Numeric vector or matrix of transform coefficients of the graph signal.

## See Also

synthesis, tight_frame

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Compute the spectral decomposition of L
decomp <- eigensort(L)

# Generate the tight frame coefficients using the tight_frame function
tf <- tight_frame(decomp$evalues, decomp$evectors)
```

```
# Create a random graph signal.
f <- rnorm(nrow(L))

# Compute the transform coefficients using the analysis operator
coef <- analysis(f, tf)

## End(Not run)
```

---

`betathresh` *Apply Beta Threshold to Data*

---

## Description

`betathresh` performs a generalized thresholding operation on the data `y`. The thresholding operation is parameterized by the parameter `beta`.

## Usage

```
betathresh(y, t, beta = 2)
```

## Arguments

| | |
|---|---|
| `y` | Numeric vector or matrix representing the noisy data. |
| `t` | Non-negative numeric value representing the threshold. |
| `beta` | Numeric value indicating the type of thresholding. |

## Details

The function offers flexibility by allowing for different types of thresholding based on the `beta` parameter. Soft thresholding, commonly used in wavelet-based denoising corresponds to `beta=1` . James-Stein thresholding corresponds to `beta=2`. The implementation includes a small constant for numerical stability when computing the thresholding operation.

The thresholding operator is defined as:

$$\tau(x, t) = x \max\left(1 - t^{\beta}|x|^{-\beta}, 0\right)$$

with $\beta \geq 1$.

## Value

`x` Numeric vector or matrix of the filtered result.

## References

Donoho, D. L., & Johnstone, I. M. (1995). Adapting to unknown smoothness via wavelet shrinkage. Journal of the american statistical association, 90(432), 1200-1224.

de Loynes, B., Navarro, F., & Olivier, B. (2021). Data-driven thresholding in denoising with spectral graph wavelet transform. Journal of Computational and Applied Mathematics, 389, 113319.

## Examples

```
# Define a 2x2 matrix
mat <- matrix(c(2, -3, 1.5, -0.5), 2, 2)

# Apply soft thresholding with a threshold of 1
betathresh(mat, 1, 1)
```

---

`download_graph` *Download Sparse Matrix form the SuiteSparse Matrix Collection*

---

## Description

`download_graph` allows to download sparse matrices from the SuiteSparse Matrix Collection.

## Usage

```
download_graph(matrixname, groupname, svd = FALSE, add_info = FALSE)
```

## Arguments

| | |
|---|---|
| `matrixname` | Name of the graph to download. |
| `groupname` | Name of the group that provides the graph. |
| `svd` | Logical, if `TRUE`, a ".mat" file containing the singular values of the matrix is downloaded (if available). Default is `FALSE`. |
| `add_info` | Logical, if `TRUE`, additional information about the graph will be fetched and included in the output. Default is `FALSE`. |

## Details

`download_graph` automatically converts the downloaded matrix into a sparse matrix format. If coordinates are associated with the graphs, they are downloaded and included in the output. Visit https://sparse.tamu.edu/ or see SuiteSparseData to explore groups and matrix names.

## Value

A list containing several components:

- `sA`: A sparse matrix representation of the downloaded graph.
- `xy`: Coordinates associated with the graph nodes (if available).
- `dim`: A data frame with the number of rows, columns, and numerically nonzero elements.
- `temp`: The path to the temporary directory where the matrix and downloaded files (including singular values if requested) are stored.
- `info`: Additional information about the graph (included when `add_info` is `TRUE`).

## Note

This temporary directory can be accessed, for example, via `list.files(grid1$temp)`. To open the read .mat files (containing singular values), "R.matlab" or "foreign" packages can be used. After using the downloaded data, you can delete the content of the temporary folder.

When `add_info` is set to `TRUE`, the function retrieves comprehensive information about the graph using `get_graph_info`.

## References

Davis, T. A., & Hu, Y. (2011). The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS), 38(1), 1-25.

Kolodziej, S. P., Aznaveh, M., Bullock, M., David, J., Davis, T. A., Henderson, M., Hu, Y., & Sandstrom, R. (2019). The suitesparse matrix collection website interface. Journal of Open Source Software, 4(35), 1244.

## See Also

`get_graph_info`, `SuiteSparseData`

## Examples

```
## Not run:
matrixname <- "grid1"
groupname <- "AG-Monien"
download_graph(matrixname,groupname)
list.files(grid1$temp)

## End(Not run)
```

---

# eigendec *Spectral decomposition of a symetric matrix*

---

## Description

Eigen decomposition of dense symmetric/hermitian matrix M using divide-and-conquer methods that provides slightly different results than the standard method, but is considerably faster for large matrices.

## Usage

```
eigendec(M)
```

## Arguments

| | |
|---|---|
| `M` | a matrix. |

## eigensort *Spectral Decomposition of a Symmetric Matrix*

### Description

`eigensort` performs the spectral decomposition of a symmetric matrix. The eigenvalues and eigenvectors are sorted in increasing order by eigenvalues.

### Usage

```
eigensort(M)
```

### Arguments

| | |
|---|---|
| `M` | Symmetric matrix, either sparse or dense, to be decomposed. |

### Value

A list containing:

- `evalues`: A vector of sorted eigenvalues in increasing order.
- `evectors`: A matrix of corresponding eigenvectors.

### Examples

```
A <- matrix(1, ncol=2, nrow=2)
dec <- eigensort(A)
```

## forward_gft *Compute Forward Graph Fourier Transform*

### Description

`forward_gft` computes the Graph Fourier Transform (GFT) of a given graph signal $f$.

### Usage

```
forward_gft(L, f, U = NULL)
```

### Arguments

| | |
|---|---|
| `L` | Laplacian matrix of the graph. |
| `f` | Numeric vector of the graph signal to analyze. |
| `U` | Matrix of the Eigenvectors of the Laplacian matrix. If NULL (default), the function will compute the eigendecomposition of the Laplacian. |

## Details

The GFT is the representation of the graph signal on an orthonormal basis of the graph's Laplacian matrix. It allows to analyze the frequency content of signals defined on graphs. In this context, the "frequency" of a graph signal refers to its decomposition in terms of the graph's Laplacian eigenvectors, which are similar to the harmonics of classical Fourier analysis.

The GFT of a graph signal $f$ is given by:

$$\hat{f} = U^T f$$

where $U$ denotes the matrix of eigenvectors of the graph's Laplacian.

When the eigenvectors $U$ are not provided, the function computes them using the Laplacian matrix $L$.

## Value

`hatf` Numeric vector. Graph Fourier Transform of $f$.

## References

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., & Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. Proceedings of the IEEE, 106(5), 808-828.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., & Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE signal processing magazine, 30(3), 83-98.

## See Also

`inverse_gft`

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Create a sample graph signal
f <- rnorm(nrow(L))

# Compute the forward GFT
hatf <- forward_gft(L, f)

## End(Not run)
```

`forward_sgwt` *Compute Forward Spectral Graph Wavelet Transform*

## Description

`forward_sgwt` computes the forward Spectral Graph Wavelet Transform (SGWT) for a given graph signal $f$.

## Usage

```
forward_sgwt(
  f,
  evalues,
  evectors,
  b = 2,
  filter_func = zetav,
  filter_params = list()
)
```

## Arguments

| | |
|---|---|
| `f` | Numeric vector representing the graph signal to analyze. |
| `evalues` | Numeric vector of eigenvalues of the Laplacian matrix. |
| `evectors` | Matrix of eigenvectors of the Laplacian matrix. |
| `b` | Numeric scalar that controls the number of scales in the SGWT. It must be greater than 1. |
| `filter_func` | Function used to compute the filter values. By default, it uses the `zetav` function but other frame filters can be pass. |
| `filter_params` | List of additional parameters required by `filter_func`. Default is an empty list. |

## Details

The transform is constructed based on the frame defined by the `tight_frame` function, without the need for its explicit calculation. Other filters can be passed as parameters. The SGWT provides a multi-scale analysis of graph signals.

Given a graph signal $f$ of length $N$, `forward_sgwt` computes the wavelet coefficients using SGWT.

The eigenvalues and eigenvectors of the graph Laplacian, are denoted as $\Lambda$ and $U$ respectively. The parameter $b$ controls the number of scales, and $\lambda_{\text{max}}$ is the largest eigenvalue.

For each scale $j = 0, \ldots, J$, where

$$J = \left\lfloor \frac{\log(\lambda_{\text{max}})}{\log(b)} \right\rfloor + 2$$

the wavelet coefficients are computed as:

$$\mathbf{w}_j = U\left(g_j \odot (U^T f)\right)$$

where

$$g_j(\lambda) = \sqrt{\psi_j(\lambda)}$$

and $\odot$ denotes element-wise multiplication.

The final result is a concatenated vector of these coefficients for all scales.

## Value

wc A concatenated vector of wavelet coefficients.

## Note

forward_sgwt can be adapted for other filters by passing a different filter function to the filter_func parameter.

The computation of $k_{\max}$ using $\lambda_{\max}$ and $b$ applies primarily to the default zetav filter. It can be overridden by providing it in the filter_params list for other filters.

## References

Göbel, F., Blanchard, G., von Luxburg, U. (2018). Construction of tight frames on graphs and application to denoising. In Handbook of Big Data Analytics (pp. 503-522). Springer, Cham.

Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2), 129-150.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## See Also

inverse_sgwt, tight_frame

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Compute the spectral decomposition of L
decomp <- eigensort(L)

# Create a sample graph signal
f <- rnorm(nrow(L))

# Compute the forward Spectral Graph Wavelet Transform
wc <- forward_sgwt(f, decomp$evalues, decomp$evectors)

## End(Not run)
```

## full *Conversion of Symmetric Sparse Matrix to Full Matrix*

### Description

full converts a symmetric sparse matrix, represented as sA, into a full matrix A.

### Usage

```
full(sA)
```

### Arguments

| | |
|---|---|
| sA | Symmetric sparse matrix, either in a sparse matrix format or in a three-column format, that needs to be converted into a full matrix. |

### Value

A Full matrix constructed from the symmetric sparse matrix sA.

### See Also

fullup

### Examples

```
sA <- pittsburgh$sA
A <- full(sA)
```

## fullup *Convert Symmetric Sparse Matrix to Full Matrix*

### Description

fullup converts a symmetric sparse matrix sA, stored as an upper triangular matrix, to a full matrix A.

### Usage

```
fullup(sA)
```

### Arguments

| | |
|---|---|
| sA | Matrix (sparseMatrix). Symmetric upper triangular matrix to be converted. |

### Details

This function can be used for transforming matrices that have been stored in a memory-efficient format (i.e., the upper triangle portion of a symmetric matrix) to their full format. The conversion is done either by directly transforming the sparse matrix or by leveraging the `full` function.

### Value

`A` Full symmetric matrix.

### See Also

`full`

### Examples

```
data(grid1)
A <- fullup(grid1$sA)
```

---

## get_graph_info *Retrieve Information Tables about a Specific Graph from the SuiteSparse Matrix Collection*

---

### Description

`get_graph_info` fetches the overview tables about a specified graph/matrix from the SuiteSparse Matrix Collection.

### Usage

```
get_graph_info(matrixname, groupname)
```

### Arguments

| | |
|---|---|
| `matrixname` | Name of the matrix/graph for which to fetch information. |
| `groupname` | Name of the group that provides the matrix/graph. |

### Details

The tables contain detailed information and properties about the graph/matrix, such as its size, number of non-zero elements, etc. Visit `https://sparse.tamu.edu/` of see `SuiteSparseData` to explore groups and matrix names.

### Value

A list of tables with detailed information about the specified matrix/graph:

- "Matrix Information"
- "Matrix Properties"
- "SVD Statistics" (if available)

## Note

The `rvest` package is used for parsing HTML, if it is not installed, the function will prompt for installation.

## References

Davis, T. A., & Hu, Y. (2011). The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS), 38(1), 1-25.

Kolodziej, S. P., Aznaveh, M., Bullock, M., David, J., Davis, T. A., Henderson, M., Hu, Y., & Sandstrom, R. (2019). The suitesparse matrix collection website interface. Journal of Open Source Software, 4(35), 1244.

## Examples

```
## Not run:
matrixname <- "grid1"
groupname <- "AG-Monien"
info_tables <- get_graph_info(matrixname,groupname)

# Matrix Information
info_tables[[1]]

# Matrix Properties
info_tables[[2]]

# SVD Statistics
info_tables[[3]]

## End(Not run)
#' @seealso \code{\link{download_graph}}, \code{\link{SuiteSparseData}}
```

---

`grid1` *Grid1 Graph from AG-Monien Graph Collection*

---

## Description

This dataset represents the "grid1" graph sourced from the AG-Monien Graph Collection, a collection of test graphs provided by Ralf Diekmann and Robert Preis. The AG-Monien collection encompasses graphs from various origins, including the Harwell-Boeing collection, NASA matrices, and other graphs.

## Usage

```
grid1
```

### Format

list of 3 elements

- xy: A matrix with the coordinates for each node in the graph.
- sA: A sparse matrix representation of the graph's adjacency matrix.
- dim: A numeric vector containing the numbers of rows, columns, and numerically nonzero elements in the adjacency matrix.
- temp: empty list (the path to the temporary directory where the matrix and downloaded files from download_graph function).
- info: info: Additional information about the graph.

### Source

AG-Monien Graph Collection by Ralf Diekmann and Robert Preis.

---

## GVN *Graph Von Neumann Variance Estimator*

---

### Description

GVN computes graph equivalent of the Von Neummann variance estimator.

### Usage

```
GVN(y, A, L)
```

### Arguments

| | |
|---|---|
| y | Numeric vector that represents the noisy data. |
| A | Adjacency matrix of the graph. |
| L | Laplacian matrix of the graph. |

### Details

In many real-world scenarios, the noise level $\sigma^2$ remains generally unknown. Given any function $g : \mathbb{R}_+ \to \mathbb{R}_+$, a straightforward computation gives:

$$\mathbf{E}[\widetilde{f}^T g(L)\widetilde{f}] = f^T g(L) f + \mathbf{E}[\xi^T g(L)\xi] = f^T g(L) f + \sigma^2 \mathrm{Tr}(g(L))$$

A biased estimator of the variance $\sigma^2$ can be given by:

$$\hat{\sigma}_1^2 = \frac{\widetilde{f}^T g(L)\widetilde{f}}{\mathrm{Tr}(g(L))}$$

Assuming the original graph signal is smooth enough that $f^T g(L) f$ is negligible compared to $\mathrm{Tr}(g(L))$, $\hat{\sigma}^2$ provides a reasonably accurate estimate of $\sigma^2$. For this function, a common choice is $g(x) = x$. Thanks to Dirichlet's formula, it follows:

$$\hat{\sigma}_1^2 = \frac{\widetilde{f}^T L \widetilde{f}}{\mathrm{Tr}(L)} = \frac{\sum_{i,j \in V} w_{ij} |\widetilde{f}(i) - \widetilde{f}(j)|^2}{2\mathrm{Tr}(L)}$$

This is the graph adaptation of the Von Neumann estimator, hence the term Graph Von Neumann estimator (GVN).

## Value

The Graph Von Neumann variance estimate for the given noisy data.

## References

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

von Neumann, J. (1941). Distribution of the ratio of the mean square successive difference to the variance. *Ann. Math. Statistics*, 35(3), 433–451.

## See Also

HPFVN

## Examples

```
## Not run:
data(minnesota)
A <- minnesota$A
L <- laplacian_mat(A)
x <- minnesota$xy[ ,1]
n <- length(x)
f <- sin(x)
sigma <- 0.1
noise <- rnorm(n, sd = sigma)
y <- f + noise
sigma^2
GVN(y, A, L)

## End(Not run)
```

---

HPFVN *High Pass Filter Von Neumann Estimator*

---

## Description

HPFVN computes graph extension of the Von Neummann variance estimator using finest scale coefficients (as in classical wavelet approaches).

## Usage

```
HPFVN(wcn, evalues, b, filter_func = zetav, filter_params = list())
```

## Arguments

| | |
|---|---|
| wcn | Numeric vector of noisy wavelet coefficients. |
| evalues | Numeric vector corresponding to Laplacian spectrum. |
| b | numeric parameter that control the number of scales. |
| filter_func | Function used to compute the filter values. By default, it uses the zetav function but other frame filters can be passed. |
| filter_params | List of additional parameters required by filter_func. Default is an empty list. |

## Details

The High Pass Filter Von Neumann Estimator (HPFVN) is the graph analog of the classical Von Neumann estimator, focusing on the finest scale coefficients. It leverages the characteristics of the graph signal's wavelet coefficients to estimate the variance:

$$\hat{\sigma}^2 = \frac{\sum_{i=nJ+1}^{n(J+1)} (\mathcal{W}y)_i^2}{\mathrm{Tr}\ \psi_J(L)}$$

## Note

HPFVN can be adapted for other filters by passing a different filter function to the filter_func parameter.

The computation of $k_{\max}$ using $\lambda_{\max}$ and $b$ applies primarily to the default zetav filter. It can be overridden by providing it in the filter_params list for other filters.

## References

Donoho, D. L., & Johnstone, I. M. (1994). Ideal spatial adaptation by wavelet shrinkage. biometrika, 81(3), 425-455.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

von Neumann, J. (1941). Distribution of the ratio of the mean square successive difference to the variance. *Ann. Math. Statistics*, 35(3), 433–451.

## See Also

GVN

## Examples

```
## Not run:
A <- grid1$sA
L <- laplacian_mat(A)
x <- grid1$xy[ ,1]
```

```
n <- length(x)
val1 <- eigensort(L)
evalues <- val1$evalues
evectors <- val1$evectors
f <- sin(x)
sigma <- 0.1
noise <- rnorm(n, sd = sigma)
y <- f + noise
b <- 2
wcn <- forward_sgwt(y, evalues, evectors, b=b)
sigma^2
HPFVN(wcn, evalues, b)
## End(Not run)
```

---

`inverse_gft` *Compute Inverse Graph Fourier Transform*

---

## Description

`inverse_gft` computes the Inverse Graph Fourier Transform (IGFT) of a given transformed graph signal $\hat{f}$.

## Usage

```
inverse_gft(L, hatf, U = NULL)
```

## Arguments

| | |
|---|---|
| `L` | Laplacian matrix of the graph (matrix). |
| `hatf` | Numeric vector. Graph Fourier Transform of the signal to be inverted. |
| `U` | Matrix of the eigenvectors of the Laplacian matrix. If NULL (default), the function will compute the eigendecomposition of the Laplacian. |

## Details

The IGFT enables the reconstruction of graph signals from their frequency domain representation. The "frequency" in the context of graph signal processing refers to the decomposition of the signal using the graph's Laplacian eigenvectors.

The IGFT of a transformed graph signal $\hat{f}$ is given by:

$$f = U\hat{f}$$

where $U$ represents the matrix of eigenvectors of the graph's Laplacian.

When the eigenvectors $U$ are not provided, the function computes them from the Laplacian matrix $L$.

## Value

`f` Numeric vector. Original graph signal obtained from the inverse transform of $\hat{f}$.

## References

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., & Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. Proceedings of the IEEE, 106(5), 808-828.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., & Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE signal processing magazine, 30(3), 83-98.

## See Also

forward_gft

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Create a sample graph signal
f <- rnorm(nrow(L))

# Compute the forward GFT
hatf <- forward_gft(L, f)

# Compute the forward GFT
recf <- inverse_gft(L, hatf)

## End(Not run)
```

---

# inverse_sgwt *Compute Inverse Spectral Graph Wavelet Transform*

---

## Description

inverse_sgwt computes the pseudo-inverse Spectral Graph Wavelet Transform (SGWT) for wavelet coefficients wc.

## Usage

```
inverse_sgwt(
  wc,
  evalues,
  evectors,
  b = 2,
  filter_func = zetav,
  filter_params = list()
)
```

## Arguments

| | |
|---|---|
| wc | Numeric vector representing the spectral graph wavelet coefficients to reconstruct the graph signal from. |
| evalues | Numeric vector of eigenvalues of the Laplacian matrix. |
| evectors | Matrix of eigenvectors of the Laplacian matrix. |
| b | Numeric scalar that control the number of scales in the SGWT. It must be greater than 1. |
| filter_func | Function used to compute the filter values. By default, it uses the zetav function but other frame filters can be passed. |
| filter_params | List of additional parameters required by filter_func. Default is an empty list. |

## Details

The computation corresponds to the frame defined by the tight_frame function. Other filters can be passed as parameters. Given the tightness of the frame, the inverse is simply the application of the adjoint linear transformation to the wavelet coefficients.

Given wavelet coefficients wc, inverse_sgwt reconstructs the original graph signal using the inverse SGWT.

The eigenvalues and eigenvectors of the graph Laplacian are denoted as $\Lambda$ and $U$ respectively. The parameter $b$ controls the number of scales, and $\lambda_{\text{max}}$ is the largest eigenvalue.

For each scale $j = 0, \ldots, J$, where

$$J = \left\lfloor \frac{\log(\lambda_{\text{max}})}{\log(b)} \right\rfloor + 2$$

the reconstructed signal for that scale is computed as:

$$\mathbf{f}_j = (U\mathbf{wc}_j \odot g_j)U^T$$

where

$$g_j(\lambda) = \sqrt{\psi_j(\lambda)}$$

and $\odot$ denotes element-wise multiplication.

The final result is the sum of $\mathbf{f}_j$ across all scales to reconstruct the entire graph signal.

## Value

f A graph signal obtained by applying the SGWT adjoint to wc.

## Note

inverse_sgwt can be adapted for other filters by passing a different filter function to the filter_func parameter. The computation of $k_{\text{max}}$ using $\lambda_{\text{max}}$ and $b$ applies primarily to the default zetav filter. It can be overridden by providing it in the filter_params list for other filters.

## References

Göbel, F., Blanchard, G., von Luxburg, U. (2018). Construction of tight frames on graphs and application to denoising. In Handbook of Big Data Analytics (pp. 503-522). Springer, Cham.

Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2), 129-150.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## See Also

forward_sgwt, tight_frame

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Compute the spectral decomposition of L
decomp <- eigensort(L)

# Create a sample graph signal
f <- rnorm(nrow(L))

# Compute the forward Spectral Graph Wavelet Transform
wc <- forward_sgwt(f, decomp$evalues, decomp$evectors)

# Reconstruct the graph signal using the inverse SGWT
f_rec <- inverse_sgwt(wc, decomp$evalues, decomp$evectors)

## End(Not run)
```

---

# laplacian_mat *Compute the Graph Laplacian Matrix*

---

## Description

laplacian_mat computes various forms of the graph Laplacian matrix for a given adjacency matrix W.

## Usage

```
laplacian_mat(W, type = "unnormalized")
```

## Arguments

| | |
|---|---|
| W | Adjacency matrix (dense or sparseMatrix). |
| type | Character string, type of Laplacian matrix to compute. Can be "unnormalized" (default), "normalized", or "randomwalk". |

## Details

The function supports three types of Laplacian matrices:

- Unnormalized Laplacian:
$$L = D - W$$
- Normalized Laplacian:
$$L_{norm} = I - D^{-1/2} W D^{-1/2}$$
- Random Walk Laplacian:
$$L_{rw} = I - D^{-1} W$$

Where:

- $D$ is the degree matrix, a diagonal matrix where each diagonal element $D_{ii}$ represents the sum of the weights of all edges connected to node $i$.
- $W$ is the adjacency matrix of the graph.
- $I$ is the identity matrix.

The function supports both standard and sparse matrix representations of the adjacency matrix.

## Value

L The graph Laplacian matrix.

## References

Chung, F. R. (1997). Spectral graph theory (Vol. 92). American Mathematical Soc.

## Examples

```
# Define the 3x3 adjacency matrix
W <- matrix(c(0, 1, 0,
              1, 0, 1,
              0, 1, 0), ncol=3)

# Non-sparse cases
laplacian_mat(W, "unnormalized")
laplacian_mat(W, "normalized")
laplacian_mat(W, "randomwalk")

# Convert W to a sparse matrix
W_sparse <- as(W, "sparseMatrix")

# Sparse cases
```

```
laplacian_mat(W_sparse, "unnormalized")
laplacian_mat(W_sparse, "normalized")
laplacian_mat(W_sparse, "randomwalk")
```

---

`LD_SUREthresh` *Level Dependent Stein's Unbiased Risk Estimate Thresholding*

---

## Description

Adaptive threshold selection using the Level Dependent Stein's Unbiased Risk Estimate.

## Usage

```
LD_SUREthresh(
  J,
  wcn,
  diagWWt,
  beta = 2,
  sigma,
  hatsigma = NA,
  policy = "uniform",
  keepSURE = FALSE
)
```

## Arguments

| | |
|---|---|
| `J` | Integer. The finest scale, or the highest frequency. This parameter determines the total number of scales that the function will process. |
| `wcn` | A numeric vector of noisy spectral graph wavelet coefficients that need to be thresholded. |
| `diagWWt` | Numeric vector of weights. |
| `beta` | Numeric. The type of thresholding to be used. If beta=1, soft thresholding is applied. If beta=2, James-Stein thresholding is applied (Default is 2). |
| `sigma` | Numeric. The standard deviation of the noise present in the wavelet coefficients. |
| `hatsigma` | Numeric. An optional estimator of the noise standard deviation. If provided, the function will also compute wavelet coefficient estimates using this estimator. |
| `policy` | The policy for threshold setting. It can be either "uniform" (default) or "dependent". |
| `keepSURE` | A logical flag. If `TRUE`, the function will also return a list containing the results of the SURE thresholding for each scale. |

## Details

This function applies SURE in a level dependent manner to wavelet coefficients, which aims to minimize SURE at each wavelet scale.

In the "uniform" policy, the thresholds are set based on the absolute value of the wavelet coefficients. In the "dependent" policy, the thresholds are set based on the wavelet coefficients normalized by the weights from `diagWWt`.

## Value

A list containing the wavelet coefficient estimates after applying the SURE thresholding.

- `wcLDSURE`: The wavelet coefficient estimates obtained by minimizing SURE.
- `wcLDhatSURE`: If `hatsigma` is provided, this component contains the wavelet coefficient estimates obtained using the `hatsigma` estimator.
- `lev_thresh`: If `keepSURE` is `TRUE`, this component contains a list of results similar to the output of `SUREthresh` for each scale.

## References

Donoho, D. L., & Johnstone, I. M. (1995). Adapting to unknown smoothness via wavelet shrinkage. Journal of the american statistical association, 90(432), 1200-1224.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. The annals of Statistics, 1135-1151.

## See Also

`SUREthresh` for the underlying thresholding method used at each scale.

## Examples

```
## Not run:
# Compute the Laplacian matrix and its eigen-decomposition
L <- laplacian_mat(grid1$sA)
U <- eigensort(L)

# Compute the tight frame coefficients
tf <- tight_frame(U$evalues, U$evectors)

# Generate some noisy observation
n <- nrow(L)
f <- randsignal(0.01, 3, grid1$sA)
sigma <- 0.01
noise <- rnorm(n, sd = sigma)
tilde_f <- f + noise

# Compute the transform coefficients
wcn <- forward_sgwt(f, U$evalues, U$evectors)
wcf <- forward_sgwt(f, U$evalues, U$evectors)

# Compute the weights
diagWWt <- colSums(t(tf)^2)

# Compute to optimal threshold
lmax <- max(U$evalues)
J <- floor(log(lmax)/log(b)) + 2
LD_opt_thresh_u <- LD_SUREthresh(J=J,
```

```
                                wcn=wcn,
                                diagWWt=diagWWt,
                                beta=2,
                                sigma=sigma,
                                hatsigma=NA,
                                policy = "uniform",
                                keepSURE = FALSE)

# Get the graph signal estimator
hatf_LD_SURE_u <- synthesis(LD_opt_thresh_u$wcLDSURE, tf)

## End(Not run)
```

---

`localize_gft` *Localize Kernel at a Graph Vertex Using GFT*

---

## Description

This function localizes a kernel at a specific vertex using the Graph Fourier Transform (GFT).

## Usage

```
localize_gft(i, L, evectors = NULL)
```

## Arguments

| | |
|---|---|
| `i` | Integer index of the node where to localize the kernel. |
| `L` | Laplacian matrix of the graph. |
| `evectors` | Numeric matrix of the eigenvectors of the Laplacian matrix. If NULL (default), the function will compute the eigendecomposition of the Laplacian. |

## Details

The GFT represents the signal in the graph's frequency domain through the eigendecomposition of the Laplacian matrix.

The kernel is localized by transforming an impulse signal centered at vertex $i$ using the GFT. The impulse for vertex $i$ is represented by a vector $s$ with all zeros except for a single one at the i-th position. The GFT of a signal $s$ is given by:

$$\hat{s} = U^T s$$

where $U$ is the matrix of eigenvectors of the Laplacian.

Applying the GFT to the impulse signal provides a spatial representation of the eigenvector (or kernel) associated with a specific frequency (eigenvalue) centered around vertex $i$. This depicts how the kernel influences the local neighborhood of the vertex.

## Value

s Kernel localized at vertex i using GFT.

## See Also

forward_gft,localize_sgwt

## Examples

```
## Not run:
L <- laplacian_mat(grid1$sA)
vertex_i <- sample(1:nrow(L), 1)
s <- localize_gft(vertex_i, L=L)
plot_signal(grid1, s)
s_gft <- forward_gft(L, s)
barplot(abs(s_gft), main="GFT of Localized Signal",
        xlab="Eigenvalue Index", ylab="Magnitude")

## End(Not run)
```

---

# localize_sgwt *Localize a Kernel at a Specific Vertex using SGWT*

---

## Description

This function localizes a kernel at a specific vertex using the Spectral Graph Wavelet Transform (SGWT).

## Usage

```
localize_sgwt(i, evalues, evectors, b = 2)
```

## Arguments

| | |
|---|---|
| i | Integer index of the node where to localize the kernel. |
| evalues | Numeric vector of the eigenvalues of the Laplacian matrix. |
| evectors | Numeric matrix of the eigenvectors of the Laplacian matrix. |
| b | Numeric scalar that controls the number of scales in the SGWT. It must be greater than 1. |

## Details

The SGWT offers a comprehensive understanding of graph signals by providing insights into both vertex (spatial) and spectral (frequency) domains.

The kernel is localized by transforming an impulse signal centered at vertex $i$ using the SGWT. The SGWT leverages a wavelet function $\psi(\lambda)$ to provide a multi-resolution analysis of the graph signal.

The impulse signal at vertex $i$ is a vector $f$ with a one at the i-th position and zeros elsewhere. The SGWT is given by:

$$W_f(\lambda) = f * \psi(\lambda) = U\psi(\Lambda)U^T f$$

where $U$ is the matrix of eigenvectors of the Laplacian and $\Lambda$ is the diagonal matrix of eigenvalues. The localized spatial view of the kernel's behavior around vertex $i$ is achieved by transforming this impulse signal using the above expression.

To gain insights into the spectral localization of this localized kernel, one can analyze its GFT to understand how the energy of the kernel is distributed across various graph frequencies. As SGWT scales move from coarse to fine, energy concentration of the localized kernel shifts from lower to higher graph frequencies, indicating tighter spectral localization.

## Value

`f` Kernel localized at vertex `i` using SGWT.

## See Also

`forward_sgwt`, `forward_gft`, `forward_gft`

## Examples

```
## Not run:
# Compute the Laplacian matrix and its eigen-decomposition
L <- laplacian_mat(grid1$sA)
decomp <- eigensort(L)

# Randomly select a vertex
vertex_i <- sample(1:nrow(L), 1)

f_sgwt <- localize_sgwt(vertex_i, evalues=decomp$evalues, evectors=decomp$evectors, b=2)

# Select one scale j from f_sgwt.
N <- nrow(grid1$sA)
j <- 5 # change scale j to view other scales
f <- f_sgwt[ ((j-1)*N+1):(j*N)]

# Plot the localized kernel (for the chosen scale) as a signal on the graph
plot_signal(grid1, f)

# Plot the magnitude of the GFT coefficients
barplot(abs(f_gft), main="GFT of Localized Signal",
        xlab="Eigenvalue Index", ylab="Magnitude")

## End(Not run)
```

`minnesota` *Minnesota Road Network*

## Description

A dataset representing the Minnesota road network along with two associated synthetic signals.

## Usage

```
minnesota
```

## Format

A list with 5 elements:

- `xy` A matrix indicating the spatial location of each node.
- `sA` A sparse matrix representation of the road network's adjacency matrix.
- `f1` Synthetic signal generated with parameters $\eta = 0.01$ and $k = 2$.
- `f2` Synthetic signal generated with parameters $\eta = 0.001$ and $k = 4$.
- `labels` A character vector with labels that represent various points of entry, border crossings, and notable cities within Minnesota, with some nodes possibly lacking specific location identifiers.

## Details

The Minnesota roads graph represents a planar structure consisting of 2642 vertices and 6606 edges.

The signals come from the referenced paper generated using `randsignal` with parameters $\eta = 0.01, k = 2$ and $\eta = 0.001, k = 4$.

## Source

D. Gleich. The MatlabBGL Matlab library.

## References

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

NYCdata *NYC Taxi Network Dataset*

## Description

A dataset derived from NYC taxi trip records. Additionally, the dataset includes a signal `f` that represents the total amount with added noise.

## Usage

```
NYCdata
```

## Format

A list with 2 elements:

- `A`: NYC adjacency matrix, constructed using Gaussian weights based on mean distances between locations.
- `f`: Signal representing the "total amount" with added artificial noise.

## Details

The graph constructed represents the connectivity based on taxi trips between different locations. The weights of the edges represent the frequency and distances of trips between locations.

The data comes from the methodology in the referenced paper. It is constructed from real-world data fetched from NYC taxis databases. The graph consists of 265 vertices which correspond to different LocationID (both Pick-Up and Drop-Off points). Gaussian weights are defined by

$$w_{ij} = \exp(-\tau d_{i,j}^2)$$

, where $d_{i,j}$ represents the mean distance taken on all the trips between locations $i$ and $j$ or $j$ and $i$.

The signal `f` is constructed based on the "total amount" variable from the taxi dataset, with added artificial noise.

## References

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## pittsburgh *Pittsburgh Census Tracts Network.*

### Description

A dataset representing the graph structure of 402 census tracts of Allegheny County, PA.

### Usage

```
pittsburgh
```

### Format

A list with 4 elements:

- `sA` A sparse matrix capturing the connections between spatially adjacent census tracts in Allegheny County.
- `xy` A matrix indicating the spatial location of each census tract.
- `f` Artificial signal with inhomogeneous smoothness across nodes and two sharp peaks near the center. This signal is formed using a mixture of five Gaussians in the underlying spatial coordinates.
- `y` Noisy version of the signal.

### Source

Data and associated materials were sourced from codes provided by Yu-Xiang Wang (UC Santa Barbara) and are associated with the referenced paper.

### References

Wang, Y. X., Sharpnack, J., Smola, A. J., & Tibshirani, R. J. (2016). Trend Filtering on Graphs. Journal of Machine Learning Research, 17, 1-41.

## plot_filter *Plot Tight-Frame Filters*

### Description

`plot_filter` provides a graphical representation of tight-frame filters as functions of the eigenvalues of the Laplacian matrix.

### Usage

```
plot_filter(lmax, b, N = 1000, filter_func = zetav, filter_params = list())
```

## Arguments

| | |
|---|---|
| lmax | Largest eigenvalue of the Laplacian matrix (numeric scalar). |
| b | Parameter that controls the number of scales (numeric scalar). |
| N | Number of discretization points for the x-axis. By default, N is set to 1000. |
| filter_func | Function used to compute the filter values. By default, it uses the zetav function but other frame filters can be pass. |
| filter_params | List of additional parameters required by filter_func. Default is an empty list. |

## Details

The plotted functions represent the square root of the values given by the zetav function at different scales.

This function plots the square roots of the functions forming the partition of unity, corresponding to the construction of tight frames on the graph. The square root operation is essential as it ensures the Parseval identity, making the constructed frame "tight" and preserving the energy of signals on the graph when mapped to their frame representation.

plot_filter first determines the number of scales based on the largest eigenvalue $\lambda_{\max}$ and the parameter $b$ as:

$$k_{\max} = \left\lfloor \frac{\log(\lambda_{\max})}{\log(b)} \right\rfloor + 2$$

The function then plots the square root of the values given by the zetav function over the range [0, $\lambda_{\max}$] for each scale.

## Note

plot_filter can be adapted for other filters by passing a different filter function to the filter_func parameter. The computation of $k_{\max}$ using $\lambda_{\max}$ and $b$ applies primarily to the default zetav filter. It can be overridden by providing it in the filter_params list for other filters.

## References

Coulhon, T., Kerkyacharian, G., & Petrushev, P. (2012). Heat kernel generated frames in the setting of Dirichlet spaces. Journal of Fourier Analysis and Applications, 18(5), 995-1066.

Göbel, F., Blanchard, G., von Luxburg, U. (2018). Construction of tight frames on graphs and application to denoising. In Handbook of Big Data Analytics (pp. 503-522). Springer, Cham.

Leonardi, N., & Van De Ville, D. (2013). Tight wavelet frames on multislice graphs. IEEE Transactions on Signal Processing, 61(13), 3357-3367.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## See Also

zetav

## Examples

```
plot_filter(6,2)
```

---

`plot_graph` *Plot Graph*

---

## Description

Visualizes a graph using ggplot2. It plots nodes as points and edges as segments connecting these points.

## Usage

```
plot_graph(z, size = 0.75)
```

## Arguments

| | |
|---|---|
| `z` | A list containing graph data. This list must have the following components:<br>• sA An adjacency matrix or a sparse Matrix representation of the graph.<br>• xy A matrix or dataframe containing the x and y coordinates of each node in the graph. |
| `size` | Numeric. Dot size for nodes. Default is 0.75. |

## Details

The function is primarily designed to work with the output from the `download_graph` function. This ensures that the graph visualization upholds the structure and properties of the retrieved graph. However, the function can also be utilized to visualize custom graph structures, provided they match to the input format.

## Note

If node coordinates `xy` are not provided, they will be calculated using spectral methods `spectral_coords`. For large graphs, this can be computationally intensive and may take significant time. Use with caution for large graphs if node coordinates are not supplied.

## See Also

`download_graph`, `plot_signal`, `spectral_coords`

## Examples

```
data(grid1)
plot_graph(grid1)
```

plot_signal *Plot a Signal on Top of a Given Graph*

## Description

Visualize a signal over a graph.

## Usage

```
plot_signal(z, f, size = 0.75, limits = range(f), ...)
```

## Arguments

| | |
|---|---|
| z | A list containing graph data. This list must have the following components:<br>• sA An adjacency matrix or a sparse Matrix representation of the graph.<br>• xy A matrix or dataframe containing the x and y coordinates of each node in the graph. |
| f | Signal to plot. |
| size | Numeric. Dot size for nodes. Default is 0.75. |
| limits | Set colormap limits. |
| ... | Additional arguments passed to guide_colourbar to customize the colorbar appearance (see ?guide_colourbar for more details). |

## Details

This function allows visualization of a graph signal f superimposed on the structure of a graph defined by z. It offers an intuitive way to analyze the behavior of graph signals in the vertex domain. The appearance of the colorbar can be customized by passing additional arguments that are available for ggplot2::guide_colourbar.

## Note

If node coordinates xy are not provided, they will be calculated using spectral methods spectral_coords. For large graphs, this can be computationally intensive and may take significant time. Use with caution for large graphs if node coordinates are not supplied.

## See Also

plot_graph, spectral_coords

## Examples

```
f <- rnorm(length(grid1$xy[,1]))
plot_signal(grid1, f)
```

PSNR *Compute the Peak Signal to Noise Ratio*

## Description

PSNR function computes the Peak Signal to Noise Ratio (PSNR) between two signals or images.

## Usage

```
PSNR(x, y)
```

## Arguments

| | |
|---|---|
| x | Numeric vector/matrix. Original reference signal/image. |
| y | numeric vector/matrix. Restored or noisy signal/image. |

## Details

Higher values of PSNR indicate closer similarity between the original and the compared signal or image.

The PSNR is defined by:

$$\mathrm{PSNR}(x, y) = 10 \log_{10} \left( \frac{\max(\max(x), \max(y))^2}{\mathrm{MSE}(x, y)} \right)$$

## Value

PSNR Numeric. Peak Signal to Noise Ratio.

## See Also

SNR

## Examples

```
x <- cos(seq(0, 10, length=100))
y <- x + rnorm(100, sd=0.5)
PSNR(x, y)
```

`randsignal` *Generate Random Signal with Varying Regularity*

## Description

`randsignal` constructs a random signal with specific regularity properties, utilizing the adjacency matrix `A` of the graph, a smoothness parameter `eta`, and an exponent `k`.

## Usage

```
randsignal(eta, k, A, r)
```

## Arguments

| | |
|---|---|
| `eta` | Numeric. Smoothness parameter (between 0 and 1). |
| `k` | Interger. Smoothness parameter. |
| `A` | Adjacency matrix. Must be symmetric. |
| `r` | Optional. Largest eigenvalue of `A` in magnitude (obtained using the `eigs` function from the `RSpectra` package if not provided). |

## Details

This method is inspired by the approach described in the first referenced paper.

The generated signal is formulated as $f = A^k x_\eta / r^k$ where $x_\eta$ represents Bernoulli random variables, and $r$ is the largest eigenvalue of the matrix $A$.

The power $k$ essentially captures the influence of a node's $k$-hop neighborhood in the generated signal, implying that a higher $k$ would aggregate more neighborhood information resulting in a smoother signal.

The normalization by the largest eigenvalue ensures that the signal remains bounded. This signal generation can be related to the Laplacian quadratic form that quantifies the smoothness of signals on graphs. By controlling the parameters $\eta$ and $k$, we can modulate the smoothness or regularity of the generated signal.

## Value

`f` a numeric vector representing the output signal.

## Note

While the `randsignal` function uses the adjacency matrix to parameterize and generate signals reflecting node-to-node interactions, the smoothness of these signals can subsequently be measured using the `smoothmodulus` function.

The generation is carried out in sparse matrices format in order to scale up.

## References

Behjat, H., Richter, U., Van De Ville, D., & Sörnmo, L. (2016). Signal-adapted tight frames on graphs. IEEE Transactions on Signal Processing, 64(22), 6017-6029.

de Loynes, B., Navarro, F., & Olivier, B. (2021). Data-driven thresholding in denoising with spectral graph wavelet transform. Journal of Computational and Applied Mathematics, 389, 113319.

## See Also

smoothmodulus

## Examples

```
## Not run:
# Generate a signal with smoothness parameters eta = 0.7 and k = 3
f <- randsignal(eta = 0.7, k = 3, A = grid1$sA)

## End(Not run)
```

---

# rlogo *R logo graph.*

---

## Description

A dataset containing a graph based on the R logo.

## Usage

```
rlogo
```

## Format

A list with 2 elements:

- xy: A matrix representing the coordinates for each node in the graph.
- sA: Adjacency matrix.

smoothmodulus *Modulus of Smoothness for Graph Signal*

## Description

smoothmodulus computes the modulus of smoothness (or Laplacian quadratic form) for a graph signal.

## Usage

```
smoothmodulus(f, A)
```

## Arguments

| | |
|---|---|
| f | Numeric vector representing the signal on the graph nodes |
| A | Adjacency matrix of the graph (matrix, can be either sparse or dense). |

## Details

smoothmodulus provide a measure that quantifies the smoothness of a signal on a graph. In other words, it provides a measure of how much a signal varies between adjacent nodes. This measure is analogous to the Laplacian quadratic form, which is a widely used metric in spectral graph theory for quantifying signal smoothness.

The modulus of smoothness is calculated using: $\mu(f) = 0.5 \times \sum_{(i,j)\in E} A_{ij}(f_i - f_j)^2$ where $E$ is the set of edges, $A_{ij}$ is the adjacency matrix entry for nodes i and j, and $f_i$ and $f_j$ are the signal values at nodes i and j respectively.

This metric essentially sums up the squared differences of signal values across adjacent nodes, weighted by the adjacency matrix. A high value indicates a more variable or irregular signal across the graph, while a lower value indicates a smoother signal.

## Value

A numeric scalar value indicating the modulus of smoothness for the graph signal.

## See Also

randsignal

## Examples

```
## Not run:
A <- grid1$sA
x <- grid1$xy[,1]
f <- sin(x)
smoothmodulus(f, A)

## End(Not run)
```

SNR *Compute the Signal to Noise Ratio*

## Description

SNR computes the Signal to Noise Ratio (SNR) between two signals, indicating the level of desired signal to the level of background noise.

## Usage

```
SNR(x, y)
```

## Arguments

| | |
|---|---|
| x | Numeric vector/matrix. Original reference signal. |
| y | Numeric vector/matrix. Restored or noisy signal. |

## Details

Higher values of SNR indicate a cleaner signal compared to the noise level. The SNR is computed as the ratio of the power of the signal (or the square of the Euclidean norm of the signal) to the power of the noise (or the square of the Euclidean norm of the signal difference), represented in decibels (dB).

The SNR is defined by:

$$\mathrm{SNR}(x, y) = 20 \log_{10} \left( \frac{\|x\|_2}{\|x - y\|_2} \right)$$

## Value

SNR Numeric. Signal to Noise Ratio.

## See Also

PSNR

## Examples

```
x <- cos(seq(0, 10, length=100))
y <- x + rnorm(100, sd=0.5)
SNR(x, y)
```

spectral_coords *Spectral Coordinates for Graph Drawing*

## Description

Calculates the spectral coordinates of a graph using the two smallest non-zero eigenvalues of the graph Laplacian.

## Usage

```
spectral_coords(adj_mat)
```

## Arguments

| | |
|---|---|
| adj_mat | A symmetric adjacency matrix or sparse matrix representing an undirected graph. |

## Details

The spectral_coords function implements a 2-dimensional spectral graph drawing method based on the eigenvectors of the graph Laplacian associated with its two smallest non-zero eigenvalues. Given a graph with adjacency matrix adj_mat, the graph Laplacian L is computed, which is a matrix representation that encodes the graph's topology. The Laplacian's eigenvalues and eigenvectors are calculated, and the eigenvectors corresponding to the second and third non-zero smallest eigenvalues are used to determine the coordinates of the graph's vertices in the plane.

## Value

A matrix where each row represents the spectral coordinates of a node in the graph.

## References

Chung, F. R. K. (1997). Spectral Graph Theory. American Mathematical Soc.

Hall, K. M. (1970). An r-dimensional quadratic placement algorithm. Management science, 17(3), 219-229.

## See Also

plot_graph, plot_signal

## Examples

```
## Not run:
matrixname <- "bcspwr02"
groupname <- "HB"
download_graph(matrixname,groupname)
xy <- spectral_coords(bcspwr02$sA)
bcspwr02$xy <- xy
plot_graph(bcspwr02)
```

```
## End(Not run)
```

---

`SuiteSparseData` *Matrix Data from SuiteSparse Matrix Collection*

---

## Description

This dataset represents the collection of matrices from the SuiteSparse Matrix Collection. The structure of the dataframe mirrors the structure presented on the SuiteSparse Matrix Collection website.

## Usage

```
SuiteSparseData
```

## Format

A data frame with 2893 rows and 8 columns:

`ID` Integer. Unique identifier for each matrix.

`Name` Character. Name of the matrix.

`Group` Character. Group name the matrix belongs to.

`Rows` Integer. Number of rows in the matrix.

`Cols` Integer. Number of columns in the matrix.

`Nonzeros` Integer. Number of non-zero elements in the matrix.

`Kind` Character. Kind or category of the matrix.

`Date` Character. Date when the matrix was added or updated.

## Details

The SuiteSparse Matrix Collection is a large set of sparse matrices that arise in real applications. It is widely used by the numerical linear algebra community for the development and performance evaluation of sparse matrix algorithms. The Collection covers a wide spectrum of domains, including both geometric and non-geometric domains.

## Note

Data download date: 2023-11-01 Note that the number of matrices in the SuiteSparse Matrix Collection may have increased since this download date.

## Source

SuiteSparse Matrix Collection website: https://sparse.tamu.edu/

## References

Davis, T. A., & Hu, Y. (2011). The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS), 38(1), 1-25.

Kolodziej, S. P., Aznaveh, M., Bullock, M., David, J., Davis, T. A., Henderson, M., Hu, Y., & Sandstrom, R. (2019). The suitesparse matrix collection website interface. Journal of Open Source Software, 4(35), 1244.

---

`SUREthresh` *Stein's Unbiased Risk Estimate*

---

## Description

Adaptive Threshold Selection Using Principle of Stein's Unbiased Risk Estimate (SURE).

## Usage

```
SUREthresh(
  wcn,
  thresh,
  diagWWt,
  beta = 2,
  sigma,
  hatsigma = NA,
  policy = "uniform",
  keepwc = TRUE
)
```

## Arguments

| | |
|---|---|
| `wcn` | Numeric vector of the noisy spectral graph wavelet coefficients. |
| `thresh` | Numeric vector of threshold values. |
| `diagWWt` | Numeric vector of weights typically derived from the diagonal elements of the wavelet frame matrix. |
| `beta` | A numeric value specifying the type of thresholding to be used, for example:<br>• 1 for soft thresholding.<br>• 2 for James-Stein thresholding. |
| `sigma` | A numeric value representing the standard deviation (sd) of the noise. |
| `hatsigma` | An optional numeric value providing an estimate of the noise standard deviation (default is `NA`). |
| `policy` | A character string determining the thresholding policy. Valid options include:<br>• `"uniform"` for a global threshold applied uniformly across all coefficients.<br>• `"dependent"` for threshold values that adaptively depend on the corresponding `diagWWt` weights. |
| `keepwc` | A logical value determining if the thresholded wavelet coefficients should be returned (Default is `TRUE`). |

## Details

The SUREthresh function is a data-driven approach to finding the optimal thresholding value for denoising wavelet coefficients. SURE provides a means to evaluate the denoising quality of a given thresholding function $h$. The expected risk in terms of the mean squared error (MSE) between the original coefficients $\mathbf{F}$ and their thresholded counterparts $h(\widetilde{\mathbf{F}})$, considering a noise variance $\sigma^2$, is given by:

$$\mathbb{E}\left[\|\mathbf{F} - h(\widetilde{\mathbf{F}})\|_2^2\right] = \mathbb{E}\left[-n\sigma^2 + \|\widetilde{\mathbf{F}} - h(\widetilde{\mathbf{F}})\|_2^2 + 2\sigma^2 \sum_{i,j=1}^{n(J+1)} \gamma_{ij}\partial_j h_i(\widetilde{\mathbf{F}})\right]$$

Where:

- $\widetilde{\mathbf{F}}$ are the noisy wavelet coefficients.
- $\gamma_{ij}$ represents the elements of the matrix obtained by multiplying the transpose of the wavelet transform matrix $\mathbf{\Psi}$ with itself, i.e., $\gamma_{ij} = (\mathbf{\Psi}^\top \mathbf{\Psi})_{ij}$.
- $h_i$ is the $i^{th}$ component of the thresholding function $h$.
- $n$ is the sample size.

The thresholding operator, represented by $h$ in the SUREthresh function, is obtained using this betathresh function. The SURE in the transformed domain can be explicitly stated as:

$$\mathbf{SURE}(h) = -n\sigma^2 + \sum_{i=1}^{n(J+1)} \widetilde{F}_i^2 \left(1 \wedge \frac{t_i^\beta}{|\widetilde{F}_i|^\beta}\right)^2 + 2\sum_{i=1}^{n(J+1)} \gamma_{ij}\mathbf{1}_{[t_i,\infty)}(|\widetilde{F}_i|)\left[1 + \frac{(\beta-1)t_i^\beta}{|\widetilde{F}_i|^\beta}\right].$$

GVN and HPFVN provide naive noise variance estimation.

## Value

A list containing:

- A dataframe with calculated SURE and hatSURE values.
- Minima of SURE and hatSURE and their corresponding optimal thresholds.
- Thresholded wavelet coefficients (if keepwc = TRUE).

## Note

The vector of thresholds thresh for evaluating the SURE can be effectively determined by ordering the absolute values of the noisy wavelet coefficients. This approach aligns with Donoho and Johnstone's trick in standard wavelet thresholding, where SURE typically reaches its minimum at one of these coefficients. For further details, see Donoho and Johnstone Section 2.3 and de Loynes et al. Section 3.3.

The function intentionally omits the irreducible variance term from the SURE calculations, as it doesn't affect the minimum's location.

Also, when 'keepwc = TRUE', the function provides thresholded wavelet coefficients for all evaluated threshold values, offering deeper insights into the effects of different thresholds.

## References

Donoho, D. L., & Johnstone, I. M. (1995). Adapting to unknown smoothness via wavelet shrinkage. Journal of the american statistical association, 90(432), 1200-1224.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. The annals of Statistics, 1135-1151.

## See Also

SURE_MSEthresh, GVN, HPFVN

## Examples

```
#
# See example in SURE_MSEthresh
#
```

---

# SURE_MSEthresh *Stein's Unbiased Risk Estimate with MSE*

---

## Description

Adaptive Threshold Selection Using Principle of SURE with the inclusion of Mean Squared Error (MSE) for comparison.

## Usage

```
SURE_MSEthresh(
  wcn,
  wcf,
  thresh,
  diagWWt,
  beta = 2,
  sigma,
  hatsigma = NA,
  policy = "uniform",
  keepwc = TRUE
)
```

## Arguments

| | |
|---|---|
| wcn | Numeric vector of the noisy spectral graph wavelet coefficients. |
| wcf | Numeric vector of the true spectral graph wavelet coefficients. |
| thresh | Numeric vector of threshold values. |

| | |
|---|---|
| diagWWt | Numeric vector of weights typically derived from the diagonal elements of the wavelet frame matrix. |
| beta | A numeric value specifying the type of thresholding to be used, for example:<br>• 1 for soft thresholding.<br>• 2 for James-Stein thresholding. |
| sigma | A numeric value representing the standard deviation (sd) of the noise. |
| hatsigma | An optional numeric value providing an estimate of the noise standard deviation (default is NA). |
| policy | A character string determining the thresholding policy. Valid options include:<br>• "uniform" for a global threshold applied uniformly across all coefficients.<br>• "dependent" for threshold values that adaptively depend on the corresponding ‘diagWWt‘ weights. |
| keepwc | A logical value determining if the thresholded wavelet coefficients should be returned (Default is TRUE). |

## Details

SURE_MSEthresh function extends the SUREthresh function by providing an MSE between the true coefficients and their thresholded versions for a given thresholding function $h$. This allows for a more comprehensive evaluation of the denoising quality in simulated scenarios where the true function is known.

## Value

A list containing:

- A dataframe with calculated MSE, SURE, and hatSURE values.
- Minima of SURE, hatSURE, and MSE, and their corresponding optimal thresholds.
- Thresholded wavelet coefficients (if keepwc = TRUE).

## References

Donoho, D. L., & Johnstone, I. M. (1995). Adapting to unknown smoothness via wavelet shrinkage. Journal of the american statistical association, 90(432), 1200-1224.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. The annals of Statistics, 1135-1151.

## See Also

SUREthresh, GVN, HPFVN

## Examples

```
## Not run:
# Compute the Laplacian matrix and its eigen-decomposition
L <- laplacian_mat(grid1$sA)
U <- eigensort(L)

# Compute the tight frame coefficients
tf <- tight_frame(U$evalues, U$evectors)

# Generate some noisy observation
n <- nrow(L)
f <- randsignal(0.01, 3, grid1$sA)
sigma <- 0.01
noise <- rnorm(n, sd = sigma)
tilde_f <- f + noise

# Compute the transform coefficients
wcn <- analysis(tilde_f, tf)
wcf <- analysis(f, tf)

# Compute the weights and use DJ trick for the SURE evaluation
diagWWt <- colSums(t(tf)^2)
thresh <- sort(abs(wcn))

# Compute to optimal threshold
opt_thresh_u <- SURE_MSEthresh(wcn,
                               wcf,
                               thresh,
                               diagWWt,
                               beta=2,
                               sigma,
                               NA,
                               policy = "uniform",
                               keepwc = TRUE)
# Extract corresponding wavelet coefficients
wc_oracle_u <- opt_thresh_u$wc[, opt_thresh_u$min["xminMSE"]]
wc_SURE_u <- opt_thresh_u$wc[, opt_thresh_u$min["xminSURE"]]

# Get the graph signal estimators
hatf_oracle_u <- synthesis(wc_oracle_u, tf)
hatf_SURE_u  <- synthesis(wc_SURE_u, tf)

# Compare the perfomance according to SNR measure
round(SNR(f, hatf_oracle_u), 2)
round(SNR(f, hatf_SURE_u), 2)

## End(Not run)
```

---

`swissroll` *Swiss Roll Graph Generation*

---

## Description

Generates points for a Swiss roll graph. The function maps points from the square $[0, 1]^2$ into the Swiss roll using the specified transformations.

## Usage

```
swissroll(N = 500, seed = NULL, a = 1, b = 4)
```

## Arguments

| | |
|---|---|
| N | Number of points drawn (numeric). |
| seed | Optionally specify a RNG seed for reproducible experiments (numeric). |
| a, b | Shape parameters (numeric). |

## Details

Given points $(x, y)$ within the unit square $[0, 1]^2$, the Swiss roll transformation is achieved using: $Sx = \pi\sqrt{(b^2 - a^2)x + a^2}$ and $Sy = \frac{\pi^2(b^2-a^2)y}{2}$. The transformed $(x, y)$ coordinates are then projected into 3D space to produce the characteristic rolled shape.

## Value

N x 3 array for 3d points.

## See Also

adjacency_mat

## Examples

```
## Not run:
pts <- swissroll(N=500, seed=0, a=1, b=4)
plot3D::scatter3D(pts[,1], pts[,2], pts[,3], colvar=NULL, col="red")

## End(Not run)
```

---

# synthesis *Compute the Synthesis Operator for Transform Coefficients*

---

## Description

synthesis computes the graph signal synthesis from its transform coefficients using the provided frame coefficients.

## Usage

```
synthesis(coeff, tf)
```

## Arguments

| | |
|---|---|
| coeff | Numeric vector/matrix. Transformed coefficients of the graph signal. |
| tf | Numeric matrix. Frame coefficients. |

## Details

The synthesis operator uses the frame coefficients to retrieve the graph signal from its representation in the transform domain. It is the adjoint of the analysis operator $T_{\mathfrak{F}}$ and is defined by the linear map $T_{\mathfrak{F}}^{*} : \mathbb{R}^{I} \to \mathbb{R}^{V}$. For a vector of coefficients $(c_i)_{i \in I}$, the synthesis operation is defined as:

$$T_{\mathfrak{F}}^{*}(c_i)_{i \in I} = \sum_{i \in I} c_i r_i$$

The synthesis is computed as:

$$\mathtt{y} = \mathtt{coeff}^{T}\,\mathtt{tf}$$

## Value

y Numeric vector/matrix. Synthesized graph signal.

## See Also

analysis, tight_frame

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Compute the spectral decomposition of L
decomp <- eigensort(L)

# Generate the tight frame coefficients using the tight_frame function
tf <- tight_frame(decomp$evalues, decomp$evectors)

# Create a random graph signal.
f <- rnorm(nrow(L))

# Compute the transform coefficients using the analysis operator
coef <- analysis(f, tf)

# Retrieve the graph signal using the synthesis operator
f_rec <- synthesis(coef, tf)

## End(Not run)
```

`tight_frame` *Tight-Frame Computation*

## Description

Constructs a tight-frame wavelet on graphs

## Usage

```
tight_frame(
  evalues,
  evectors,
  b = 2,
  filter_func = zetav,
  filter_params = list()
)
```

## Arguments

| | |
|---|---|
| `evalues` | Numeric vector containing the eigenvalues of the Laplacian matrix. |
| `evectors` | Matrix of the corresponding eigenvectors of the Laplacian matrix. |
| `b` | Numeric scalar. Parameter that controls the number of scales in the wavelet decomposition. |
| `filter_func` | Function used to compute the filter values. By default, it uses the `zetav` function but other frame filters can be passed. |
| `filter_params` | List of additional parameters required by filter_func. Default is an empty list. |

## Value

Matrix of the tight-frame wavelet coefficients.

## Note

`tight_frame` can be adapted for other filters by passing a different filter function to the `filter_func` parameter. The computation of $k_{\max}$ using $\lambda_{\max}$ and $b$ applies primarily to the default `zetav` filter. It can be overridden by providing it in the `filter_params` list for other filters.

## References

Coulhon, T., Kerkyacharian, G., & Petrushev, P. (2012). Heat kernel generated frames in the setting of Dirichlet spaces. Journal of Fourier Analysis and Applications, 18(5), 995-1066.

Göbel, F., Blanchard, G., von Luxburg, U. (2018). Construction of tight frames on graphs and application to denoising. In Handbook of Big Data Analytics (pp. 503-522). Springer, Cham.

Leonardi, N., & Van De Ville, D. (2013). Tight wavelet frames on multislice graphs. IEEE Transactions on Signal Processing, 61(13), 3357-3367.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## Examples

```
## Not run:
# Extract the adjacency matrix from the grid1 and compute the Laplacian
L <- laplacian_mat(grid1$sA)

# Compute the spectral decomposition of L
decomp <- eigensort(L)

# Generate the tight frame coefficients using the tight_frame function
tf <- tight_frame(decomp$evalues, decomp$evectors)

## End(Not run)
```

---

`zetav` *Evaluate Localized Tight-Frame Filter Functions*

---

## Description

`zetav` evaluates the filters associated with a specific tight-frame construction.

## Usage

```
zetav(x, k, b = 2)
```

## Arguments

| | |
|---|---|
| `x` | A vector representing the support on which to evaluate the filter |
| `k` | A scalar representing the scale index. |
| `b` | A scalar parameter that governs the number of scales (b=2 default). |

## Details

The function `zetav` evaluates the partition of unity functions $\psi$ following the methodology described in the references similar to the Littlewood-Paley type, based on a partition of unity, as proposed in the reference papers. This approach, inspired by frame theory, facilitates the construction of filter banks, ensuring effective spectral localization.

A finite collection $(\psi_j)_{j=0,\ldots,J}$ is a finite partition of unity on the compact interval $[0, \lambda_{\max}]$. It satisfies:

$$\psi_j : [0, \lambda_{\max}] \to [0, 1] \text{ for all } j \in \{1, \ldots, J\} \text{ and } \forall \lambda \in [0, \lambda_{\max}], \ \sum_{j=0}^{J} \psi_j(\lambda) = 1.$$

Let $\omega : \mathbb{R}^+ \to [0, 1]$ be a function with support in [0,1]. It's defined as:

$$\omega(x) = \begin{cases} 1 & \text{if } x \in [0, b^{-1}] \\ b \cdot \frac{x}{1-b} + \frac{b}{b-1} & \text{if } x \in (b^{-1}, 1] \\ 0 & \text{if } x > 1 \end{cases}$$

For a given $b > 1$. Based on this function $\omega$, the partition of unity functions $\psi$ are defined as:

$$\psi_0(x) = \omega(x)$$

and for all $j \geq 1$:

$$\psi_j(x) = \omega(b^{-j}x) - \omega(b^{-j+1}x)$$

where $J$ is defined by:

$$J = \left\lfloor \frac{\log \lambda_{\max}}{\log b} \right\rfloor + 2$$

Given this finite partition of unity $(\psi_j)_{j=0,\ldots,J}$, the Parseval identity implies that the following set of vectors forms a tight frame:

$$\mathfrak{F} = \left\{ \sqrt{\psi_j}(\mathcal{L})\delta_i : j = 0, \ldots, J, i \in V \right\}.$$

## Value

Returns a numeric vector of evaluated filter values.

## References

Coulhon, T., Kerkyacharian, G., & Petrushev, P. (2012). Heat kernel generated frames in the setting of Dirichlet spaces. Journal of Fourier Analysis and Applications, 18(5), 995-1066.

Göbel, F., Blanchard, G., von Luxburg, U. (2018). Construction of tight frames on graphs and application to denoising. In Handbook of Big Data Analytics (pp. 503-522). Springer, Cham.

Leonardi, N., & Van De Ville, D. (2013). Tight wavelet frames on multislice graphs. IEEE Transactions on Signal Processing, 61(13), 3357-3367.

de Loynes, B., Navarro, F., Olivier, B. (2021). Data-driven thresholding in denoising with Spectral Graph Wavelet Transform. Journal of Computational and Applied Mathematics, Vol. 389.

## Examples

```
## Not run:
  x <- seq(0, 2, by = 0.1)
  g <- zetav(x, 1, 2)
  plot(x, g, type = "l")

## End(Not run)
```

# Index