

# Package ‘BioGSP’

October 1, 2025

**Type** Package

**Title** Biological Graph Signal Processing for Spatial Data Analysis

**Version** 1.0.0

**Author** Yuzhou Chang <yuzhou.chang@osumc.edu>

**Maintainer** Yuzhou Chang <yuzhou.chang@osumc.edu>

**Description** Implementation of Graph Signal Processing (GSP) methods including Spectral Graph Wavelet Transform (SGWT) for analyzing spatial patterns in biological data. Based on Hammond, Vandergheynst, and Gribonval (2011) "Wavelets on Graphs via Spectral Graph Theory". Provides tools for multi-scale analysis of spatial signals, including forward and inverse transforms, energy analysis, and visualization functions tailored for biological applications.

**License** GPL-3

**URL** <https://github.com/BMEngineerR/BioGSP>

**BugReports** <https://github.com/BMEngineerR/BioGSP/issues>

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Imports** Matrix, igraph, RANN, RSpectra, ggplot2, patchwork, gridExtra, viridis, methods

**Suggests** remotes, dplyr, knitr, rmarkdown, ggrepel

**VignetteBuilder** knitr

**Depends** R (>= 3.5.0)

**NeedsCompilation** no

## Contents

BioGSP-package	2
cal_laplacian	4
codex_toy_data	5
compare_kernel_families	7
compute_sgwt_filters	8
cosine_similarity	9
demo_sgwt	9
FastDecompositionLap	10
find_knee_point	11

gft . . . . .	11
hello_sgwt . . . . .	12
igft . . . . .	12
initSGWT . . . . .	13
install_and_load . . . . .	14
plot_FM . . . . .	14
plot_sgwt_decomposition . . . . .	15
print.SGWT . . . . .	16
runSGCC . . . . .	16
runSGWT . . . . .	17
runSpecGraph . . . . .	18
sgwt-globals . . . . .	19
sgwt_auto_scales . . . . .	19
sgwt_energy_analysis . . . . .	19
sgwt_forward . . . . .	20
sgwt_get_kernels . . . . .	21
sgwt_inverse . . . . .	21
simulate_checkerboard . . . . .	22
simulate_multiscale . . . . .	23
simulate_stripe_patterns . . . . .	24
visualize_checkerboard . . . . .	25
visualize_multiscale . . . . .	25
visualize_sgwt_kernels . . . . .	26
visualize_similarity_xy . . . . .	27
visualize_stripe_patterns . . . . .	29
<b>Index</b>	<b>31</b>

---

BioGSP-package	<i>BioGSP: Biological Graph Signal Processing for Spatial Data Analysis</i>
----------------	---

---

## Description

The BioGSP package provides a comprehensive implementation of Graph Signal Processing (GSP) methods including Spectral Graph Wavelet Transform (SGWT) for analyzing spatial patterns in biological data. This implementation is based on Hammond, Vandergheynst, and Gribonval (2011) "Wavelets on Graphs via Spectral Graph Theory".

## Details

The package enables multi-scale analysis of spatial signals by:

- Building graphs from spatial coordinates using k-nearest neighbors
- Computing graph Laplacian eigendecomposition for spectral analysis
- Designing wavelets in the spectral domain using various kernel functions
- Decomposing signals into scaling and wavelet components at multiple scales
- Providing reconstruction capabilities with error analysis
- Offering comprehensive visualization and analysis tools

## Main Functions

`initSGWT` Initialize SGWT object with data and parameters  
`runSpecGraph` Build graph and compute eigendecomposition  
`runSGWT` Perform forward and inverse SGWT transforms  
`runSGCC` Calculate weighted similarity between signals  
`sgwt_forward` Forward SGWT transform  
`sgwt_inverse` Inverse SGWT transform  
`sgwt_energy_analysis` Energy distribution analysis  
`plot_sgwt_decomposition` Visualization of SGWT components  
`demo_sgwt` Demonstration with synthetic data

## Applications

The BioGSP package is particularly useful for:

- Spatial biology: Analyzing cell distribution patterns in tissue imaging (CODEX, Visium, etc.)
- Single-cell genomics: Spatial transcriptomics and proteomics analysis
- Neuroscience: Brain connectivity and signal analysis
- Pathology: Tumor microenvironment and tissue architecture analysis
- Developmental biology: Spatial pattern formation and cell fate mapping
- Immunology: Immune cell spatial organization and interactions

## Author(s)

BioGSP Development Team

## References

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

## See Also

Useful links:

- <https://github.com/BMEngineer/BioGSP>
- Report bugs at <https://github.com/BMEngineer/BioGSP/issues>

## Examples

```
## Not run:  
# Load the package  
library(BioGSP)  
  
# Run a quick demo  
demo_result <- demo_sgwt()  
  
# Generate synthetic data  
set.seed(123)  
n <- 100
```

```

data <- data.frame(
  x = runif(n, 0, 10),
  y = runif(n, 0, 10),
  signal = sin(runif(n, 0, 2*pi))
)

# New workflow: Initialize -> Build Graph -> Run SGWT
SG <- initSGWT(data, signals = "signal", k = 8, J = 4, kernel_type = "heat")
SG <- runSpecGraph(SG)
SG <- runSGWT(SG)

# Analyze results
energy_analysis <- sgwt_energy_analysis(SG)
print(energy_analysis)

## End(Not run)

```

---

cal\_laplacian

---

*Calculate Graph Laplacian Matrix*


---

## Description

Compute unnormalized, normalized, or random-walk Laplacian from an adjacency matrix.

## Usage

```
cal_laplacian(W, type = c("unnormalized", "normalized", "randomwalk"))
```

## Arguments

W	A square adjacency matrix (can be dense or sparse).
type	Type of Laplacian to compute: "unnormalized", "normalized", or "randomwalk".

## Value

Laplacian matrix of the same class as input.

## Examples

```

## Not run:
W <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cal_laplacian(W, type = "normalized")

## End(Not run)

```

---

codex_toy_data	<i>Toy CODEX Spatial Cell Type Data</i>
----------------	---

---

**Description**

A synthetic dataset mimicking CODEX multiplexed imaging data for demonstrating Spectral Graph Wavelet Transform (SGWT) analysis on spatial cell type distributions. The dataset contains spatial coordinates and cell type annotations for multiple immune cell populations arranged in realistic spatial clusters.

**Usage**

```
data(codex_toy_data)
```

**Format**

A data frame with 18604 rows and 5 columns:

- cellLabel** Character. Unique identifier for each cell
- Y\_cent** Numeric. Y coordinate of cell centroid (0-115 range)
- X\_cent** Numeric. X coordinate of cell centroid (0-116 range)
- Annotation5** Character. Full descriptive cell type name
- ROI\_num** Character. Region of interest identifier ("ROI\_0" through "ROI\_15")

**Details**

The dataset contains 16 regions of interest (ROI\_0 through ROI\_15) with different spatial patterns and varying cell counts (945-1497 cells per ROI). Each ROI represents a distinct tissue region with unique spatial arrangements of the same cell types.

ROI Distribution:

- **ROI\_0**: 952 cells
- **ROI\_1**: 945 cells
- **ROI\_2**: 1155 cells
- **ROI\_3**: 1421 cells
- **ROI\_4**: 1096 cells
- **ROI\_5**: 1420 cells
- **ROI\_6-ROI\_15**: 958-1497 cells each

Cell types across all ROIs include:

- **BCL6- B Cell** (~3719 cells): Primary B cell population
- **CD4 T** (~4092 cells): Helper T cells - largest population
- **CD8 T** (~3346 cells): Cytotoxic T cells
- **DC** (~2233 cells): Dendritic cells
- **M1** (~1490 cells): M1 macrophages
- **CD4 Treg** (~1490 cells): Regulatory T cells

- **BCL6+ B Cell** (~931 cells): Activated B cells
- **Endothelial** (~746 cells): Vascular cells
- **M2** (~370 cells): M2 macrophages
- **Myeloid** (~186 cells): Other myeloid cells
- **Other** (~1 cells): Miscellaneous cell types

This synthetic data is designed to demonstrate:

- Spatial clustering patterns of different cell types
- Multi-scale spatial analysis using SGWT
- Cross-cell type correlation analysis
- Graph construction and eigenvalue analysis
- Wavelet decomposition of spatial signals

### Source

Generated synthetically using clustered normal distributions with realistic parameters based on real CODEX data characteristics.

### Examples

```
# Load the toy dataset
data(codex_toy_data)

# Examine the structure
str(codex_toy_data)
head(codex_toy_data)

# Summary of cell types
table(codex_toy_data$Annotation5)

# Summary by ROI
table(codex_toy_data$ROI_num)
table(codex_toy_data$ROI_num, codex_toy_data$Annotation5)

# Quick visualization of spatial distribution
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(codex_toy_data, aes(x = X_cent, y = Y_cent, color = Annotation5)) +
    geom_point(size = 0.8, alpha = 0.7) +
    facet_wrap(~ROI_num, scales = "free") +
    labs(title = "Toy CODEX Spatial Cell Distribution by ROI",
         x = "X Coordinate", y = "Y Coordinate") +
    theme_minimal() +
    scale_y_reverse()
}

# Basic SGWT analysis example
## Not run:
# Focus on BCL6+ B Cell cells in ROI_1 for SGWT analysis
bcl6nb_data <- codex_toy_data[codex_toy_data$Annotation5 == "BCL6+ B Cell" &
                             codex_toy_data$ROI_num == "ROI_1", ]

# Create binned representation
```

```

library(dplyr)
binned_data <- codex_toy_data %>%
  filter(Annotation5 == "BCL6- B Cell", ROI_num == "ROI_1") %>%
  mutate(
    x_bin = cut(X_cent, breaks = 20, labels = FALSE),
    y_bin = cut(Y_cent, breaks = 20, labels = FALSE)
  ) %>%
  group_by(x_bin, y_bin) %>%
  summarise(cell_count = n(), .groups = 'drop')

# Prepare for SGWT
complete_grid <- expand_grid(x_bin = 1:20, y_bin = 1:20)
sgwt_data <- complete_grid %>%
  left_join(binned_data, by = c("x_bin", "y_bin")) %>%
  mutate(
    cell_count = ifelse(is.na(cell_count), 0, cell_count),
    x = x_bin,
    y = y_bin,
    signal = cell_count / max(cell_count, na.rm = TRUE)
  ) %>%
  select(x, y, signal)

# Apply SGWT
sgwt_result <- SGWT(data.in = sgwt_data,
  signal = "signal",
  k = 8,
  J = 3,
  kernel_type = "heat")

## End(Not run)

```

---

compare\_kernel\_families

*Compare different kernel families*

---

## Description

Visualize and compare different kernel families (both scaling and wavelet filters)

## Usage

```

compare_kernel_families(
  x_range = c(0, 3),
  scale_param = 1,
  plot_results = TRUE
)

```

## Arguments

<code>x_range</code>	Range of x values to evaluate (default: <code>c(0, 3)</code> )
<code>scale_param</code>	Scale parameter for all functions (default: 1)
<code>plot_results</code>	Whether to plot the comparison (default: <code>TRUE</code> )

**Value**

Data frame with x values and kernel values for each family

**Examples**

```
comparison <- compare_kernel_families()
comparison <- compare_kernel_families(x_range = c(0, 5), scale_param = 1.5)
```

---

```
compute_sgwt_filters
```

*Compute SGWT filters*

---

**Description**

Compute wavelet and scaling function coefficients in the spectral domain

**Usage**

```
compute_sgwt_filters(eigenvalues, scales, lmax = NULL, kernel_type = "heat")
```

**Arguments**

eigenvalues	Eigenvalues of the graph Laplacian
scales	Vector of scales for the wavelets
lmax	Maximum eigenvalue (optional)
kernel_type	Kernel family that defines both scaling and wavelet filters (default: "mexican_hat", options: "mexican_hat", "meyer", "heat")

**Value**

List of filters (scaling function + wavelets)

**Examples**

```
eigenvals <- c(0, 0.1, 0.5, 1.0, 1.5)
scales <- c(2, 1, 0.5)
filters <- compute_sgwt_filters(eigenvals, scales)
filters_meyer <- compute_sgwt_filters(eigenvals, scales, kernel_type = "meyer")
filters_heat <- compute_sgwt_filters(eigenvals, scales, kernel_type = "heat")
```



---

cosine\_similarity    *Calculate cosine similarity between two vectors*

---

### Description

Calculate cosine similarity between two numeric vectors with numerical stability

### Usage

```
cosine_similarity(x, y, eps = 1e-12)
```

### Arguments

x	First vector
y	Second vector
eps	Small numeric for numerical stability when norms are near zero (default 1e-12)

### Value

Cosine similarity value (between -1 and 1)

### Examples

```
x <- c(1, 2, 3)
y <- c(2, 3, 4)
similarity <- cosine_similarity(x, y)
# With custom eps for numerical stability
similarity2 <- cosine_similarity(x, y, eps = 1e-10)
```

---

demo\_sgwt    *Demo function for SGWT*

---

### Description

Demonstration function showing basic SGWT usage with synthetic data using the new workflow:  
initSGWT -> runSpecGraph -> runSGWT

### Usage

```
demo_sgwt(verbose = TRUE)
```

### Arguments

verbose	Logical; if TRUE, show progress messages and results (default: TRUE)
---------	--

### Value

SGWT object with complete analysis

**Examples**

```
## Not run:
SG <- demo_sgw()
print(SG)

## End(Not run)
```

---

FastDecompositionLap

*Fast eigendecomposition of Laplacian matrix*


---

**Description**

Perform fast eigendecomposition using RSpectra for large matrices

**Usage**

```
FastDecompositionLap(
  laplacianMat = NULL,
  k_eigen = 25,
  which = "LM",
  sigma = NULL,
  opts = list(),
  lower = TRUE,
  ...
)
```

**Arguments**

laplacianMat	Laplacian matrix
k_eigen	Number of eigenvalues to compute (default: 25)
which	Which eigenvalues to compute ("LM", "SM", etc.)
sigma	Shift parameter for eigenvalue computation
opts	Additional options for eigenvalue computation
lower	Whether to compute from lower end of spectrum
...	Additional arguments

**Value**

List with eigenvalues (evalues) and eigenvectors (evectors)

**Examples**

```
## Not run:
# Create a Laplacian matrix and decompose
L <- matrix(c(2, -1, -1, -1, 2, -1, -1, -1, 2), nrow = 3)
decomp <- FastDecompositionLap(L, k_eigen = 25)

## End(Not run)
```

---

find_knee_point	<i>Find knee point in a curve</i>
-----------------	-----------------------------------

---

**Description**

Simple knee point detection using the maximum curvature method

**Usage**

```
find_knee_point(y, sensitivity = 1)
```

**Arguments**

y	Numeric vector of y values
sensitivity	Sensitivity parameter (not used in this simple implementation)

**Value**

Index of the knee point

**Examples**

```
y <- c(1, 2, 3, 10, 11, 12) # curve with a knee
knee_idx <- find_knee_point(y)
```

---

gft	<i>Graph Fourier Transform</i>
-----	--------------------------------

---

**Description**

Compute the Graph Fourier Transform (GFT) of a signal using Laplacian eigenvectors.

**Usage**

```
gft(signal, U)
```

**Arguments**

signal	Input signal (vector or matrix)
U	Matrix of eigenvectors (dense matrix preferred)

**Value**

Transformed signal in the spectral domain (vector or matrix)

---

hello\_sgwt

Hello function for SGWT package demonstration

---

### Description

Simple hello function to demonstrate package loading

### Usage

```
hello_sgwt()
```

### Value

Character string with greeting

### Examples

```
hello_sgwt()
```

---

igft

Inverse Graph Fourier Transform

---

### Description

Compute the Inverse Graph Fourier Transform (IGFT) of spectral coefficients using Laplacian eigenvectors.

### Usage

```
igft(fourier_coeffs, U)
```

### Arguments

```
fourier_coeffs
```

Input Fourier coefficients (vector or matrix)

```
U
```

Matrix of eigenvectors (dense matrix preferred)

### Value

Reconstructed signal in the vertex domain (vector or matrix)

### Examples

```
## Not run:
# Single signal
signal_reconstructed <- igft(fourier_coeffs, eigenvectors)

# Multiple signals (batch processing)
signals_reconstructed <- igft(fourier_coeffs_matrix, eigenvectors)

## End(Not run)
```

---

initSGWT	<i>Initialize SGWT object</i>
----------	-------------------------------

---

## Description

Build an SGWT object with Data and Parameters slots, validate inputs.

## Usage

```
initSGWT(
  data.in,
  x_col = "x",
  y_col = "y",
  signals = NULL,
  scales = NULL,
  J = 5,
  scaling_factor = 2,
  kernel_type = "heat"
)
```

## Arguments

<code>data.in</code>	Data frame containing spatial coordinates and signal data
<code>x_col</code>	Character string specifying the column name for X coordinates (default: "x")
<code>y_col</code>	Character string specifying the column name for Y coordinates (default: "y")
<code>signals</code>	Character vector of signal column names to analyze. If NULL, all non-coordinate columns are used.
<code>scales</code>	Vector of scales for the wavelets. If NULL, scales are auto-generated.
<code>J</code>	Number of scales to generate if scales is NULL (default: 5)
<code>scaling_factor</code>	Scaling factor between consecutive scales (default: 2)
<code>kernel_type</code>	Kernel family ("mexican_hat", "meyer", or "heat") (default: "heat")

## Value

SGWT object with Data and Parameters slots initialized

## Examples

```
## Not run:
# Initialize SGWT object
data <- data.frame(x = runif(100), y = runif(100),
                  signal1 = rnorm(100), signal2 = rnorm(100))
SG <- initSGWT(data, signals = c("signal1", "signal2"))

## End(Not run)
```

---

install_and_load	<i>Install and load packages</i>
------------------	----------------------------------

---

**Description**

Utility function to install and load packages from CRAN or GitHub

**Usage**

```
install_and_load(packages)
```

**Arguments**

packages	Named vector where names are package names and values are source URLs
----------	---

**Value**

NULL (side effect: installs and loads packages)

**Examples**

```
## Not run:
packages <- c("ggplot2" = "ggplot2", "devtools" = "r-lib/devtools")
install_and_load(packages)

## End(Not run)
```

---

plot_FM	<i>Plot Fourier modes (eigenvectors) from SGWT object</i>
---------	---

---

**Description**

Plot low-frequency and high-frequency Fourier modes (eigenvectors) from the graph Laplacian eigendecomposition in an SGWT object

**Usage**

```
plot_FM(SG, mode_type = "both", n_modes = 6, ncol = 3, point_size = 1.5)
```

**Arguments**

SG	SGWT object with Graph slot computed (from runSpecGraph)
mode_type	Type of modes to plot: "low", "high", or "both" (default: "both")
n_modes	Number of modes to plot for each type (default: 6)
ncol	Number of columns in plot layout (default: 3)
point_size	Size of points in the plot (default: 1.5)

**Value**

Combined plot of Fourier modes

**Examples**

```
## Not run:
# Plot both low and high frequency modes
SG <- initSGWT(data) %>% runSpecGraph()
plot_FM(SG, mode_type = "both", n_modes = 4)

# Plot only low frequency modes
plot_FM(SG, mode_type = "low", n_modes = 8)

## End(Not run)
```

---

```
plot_sgwt_decomposition
```

*Plot SGWT decomposition results*

---

**Description**

Visualize SGWT decomposition components including original signal, scaling function, wavelet coefficients, and reconstructed signal

**Usage**

```
plot_sgwt_decomposition(SG, signal_name = NULL, plot_scales = NULL, ncol = 3)
```

**Arguments**

SG	SGWT object with Forward and Inverse results computed
signal_name	Name of signal to plot (default: first signal)
plot_scales	Which wavelet scales to plot (default: first 4)
ncol	Number of columns in the plot layout (default: 3)

**Value**

ggplot object with combined plots

**Examples**

```
## Not run:
# Assuming you have SGWT object
plots <- plot_sgwt_decomposition(SG_object, signal_name = "signal1")
print(plots)

## End(Not run)
```

---

<code>print.SGWT</code>	<i>Print method for SGWT objects</i>
-------------------------	--------------------------------------

---

### Description

Print method for SGWT objects

### Usage

```
## S3 method for class 'SGWT'
print(x, ...)
```

### Arguments

<code>x</code>	SGWT object to print
<code>...</code>	Additional arguments passed to print methods

---

<code>runSGCC</code>	<i>Run SGCC weighted similarity analysis in Fourier domain</i>
----------------------	--

---

### Description

Calculate energy-normalized weighted similarity between two signals using Fourier domain coefficients directly (no vertex domain reconstruction). Excludes DC component and uses energy-based weighting consistent with Parseval's theorem.

### Usage

```
runSGCC(
  signal1,
  signal2,
  SG = NULL,
  eps = 1e-12,
  validate = TRUE,
  return_parts = TRUE,
  low_only = FALSE
)
```

### Arguments

<code>signal1</code>	Either a signal name (character) for SG object, or SGWT Forward result, or SGWT object
<code>signal2</code>	Either a signal name (character) for SG object, or SGWT Forward result, or SGWT object
<code>SG</code>	SGWT object (required if signal1/signal2 are signal names)
<code>eps</code>	Small numeric for numerical stability (default: 1e-12)
<code>validate</code>	Logical; if TRUE, check consistency (default: TRUE)
<code>return_parts</code>	Logical; if TRUE, return detailed components (default: TRUE)
<code>low_only</code>	Logical; if TRUE, compute only low-frequency similarity (default: FALSE)



**Value**

Similarity analysis results computed in Fourier domain

**Examples**

```
## Not run:
# Between two signals in same SGWT object
similarity <- runSGCC("signal1", "signal2", SG = SG_object)

# Between two SGWT objects
similarity <- runSGCC(SG_object1, SG_object2)

## End(Not run)
```

---

runSGWT

---

*Run SGWT forward and inverse transforms for all signals*


---

**Description**

Perform SGWT analysis on all signals in the SGWT object. Uses batch processing for multiple signals when possible for efficiency. Assumes Graph slot is populated by runSpecGraph().

**Usage**

```
runSGWT(SG, use_batch = TRUE, verbose = TRUE)
```

**Arguments**

SG	SGWT object with Graph slot populated
use_batch	Whether to use batch processing for multiple signals (default: TRUE)
verbose	Whether to print progress messages (default: TRUE)

**Value**

Updated SGWT object with Forward and Inverse slots populated

**Examples**

```
## Not run:
SG <- initSGWT(data)
SG <- runSpecGraph(SG)
SG <- runSGWT(SG) # Uses batch processing by default
SG <- runSGWT(SG, use_batch = FALSE) # Force individual processing

## End(Not run)
```

---

runSpecGraph	<i>Build spectral graph for SGWT object</i>
--------------	---

---

## Description

Generate Graph slot information including adjacency matrix, Laplacian matrix, eigenvalues, and eigenvectors.

## Usage

```
runSpecGraph(
  SG,
  k = 25,
  laplacian_type = "normalized",
  length_eigenvalue = NULL,
  verbose = TRUE
)
```

## Arguments

SG	SGWT object from initSGWT()
k	Number of nearest neighbors for graph construction (default: 25)
laplacian_type	Type of graph Laplacian ("unnormalized", "normalized", or "randomwalk") (default: "normalized")
length_eigenvalue	Number of eigenvalues/eigenvectors to compute (default: NULL, uses full length)
verbose	Whether to print progress messages (default: TRUE)

## Value

Updated SGWT object with Graph slot populated

## Examples

```
## Not run:
SG <- initSGWT(data)
# Uses full length by default
SG <- runSpecGraph(SG, k = 30, laplacian_type = "normalized")
# Or specify custom length
SG <- runSpecGraph(SG, k = 30, laplacian_type = "normalized",
  length_eigenvalue = 30)

## End(Not run)
```

---

sgwt-globals	<i>Global variables used in ggplot2 aesthetics</i>
--------------	--

---

**Description**

This file declares global variables used in ggplot2 aesthetics to avoid R CMD check NOTEs about undefined global functions or variables.

---

sgwt_auto_scales	<i>Generate automatic scales for SGWT</i>
------------------	---

---

**Description**

Generate logarithmically spaced scales for SGWT

**Usage**

```
sgwt_auto_scales(lmax, J = 5, scaling_factor = 2)
```

**Arguments**

lmax	Maximum eigenvalue
J	Number of scales
scaling_factor	Scaling factor between consecutive scales

**Value**

Vector of scales

**Examples**

```
scales <- sgwt_auto_scales(lmax = 2.0, J = 5, scaling_factor = 2)
```

---

sgwt_energy_analysis	<i>Analyze SGWT energy distribution across scales in Fourier domain</i>
----------------------	---

---

**Description**

Calculate and analyze energy distribution across different scales using Fourier domain coefficients directly (consistent with Parseval's theorem). Excludes DC component for more accurate energy analysis.

**Usage**

```
sgwt_energy_analysis(SG, signal_name = NULL)
```

**Arguments**

SG                      SGWT object with Forward results computed  
 signal\_name      Name of signal to analyze (default: first signal)

**Value**

Data frame with energy analysis results computed in Fourier domain

**Examples**

```
## Not run:
# Assuming you have SGWT object
energy_analysis <- sgwt_energy_analysis(SG_object, signal_name = "signal1")
print(energy_analysis)

## End(Not run)
```

---

sgwt_forward	<i>Forward SGWT transform (single or batch)</i>
--------------	---

---

**Description**

Transform signal(s) to spectral domain and apply SGWT filters. Handles both single signals (vector) and multiple signals (matrix) efficiently. Stores original and filtered Fourier coefficients for analysis.

**Usage**

```
sgwt_forward(
  signal,
  eigenvectors,
  eigenvalues,
  scales,
  lmax = NULL,
  kernel_type = "heat"
)
```

**Arguments**

signal              Input signal vector OR matrix where each column is a signal (n\_vertices x n\_signals)  
 eigenvectors      Eigenvectors of the graph Laplacian  
 eigenvalues      Eigenvalues of the graph Laplacian  
 scales              Vector of scales for the wavelets  
 lmax                Maximum eigenvalue (optional)  
 kernel\_type      Kernel family that defines both scaling and wavelet filters (default: "heat")

**Value**

List containing:

**fourier\_coefficients** List with original and filtered Fourier coefficients  
**filters** Filter bank used

**Examples**

```
## Not run:
# Single signal
result <- sgwt_forward(signal, eigenvectors, eigenvalues, scales)

# Multiple signals (batch processing)
signals_matrix <- cbind(signal1, signal2, signal3)
result <- sgwt_forward(signals_matrix, eigenvectors, eigenvalues, scales)

## End(Not run)
```

---

sgwt_get_kernels	<i>Get a unified kernel family (low-pass and band-pass) by kernel_type</i>
------------------	--

---

**Description**

Returns a pair of functions implementing the scaling (low-pass) and wavelet (band-pass) kernels for a given kernel family. This enforces consistency: a single `kernel_type` defines both filters.

**Usage**

```
sgwt_get_kernels(kernel_type = "heat")
```

**Arguments**

`kernel_type` Kernel family name ("mexican\_hat", "meyer", or "heat")

**Value**

A list with two functions: `list(scaling = function(x, scale_param), wavelet = function(x, scale_param))`

---

sgwt_inverse	<i>Inverse SGWT transform (single or batch)</i>
--------------	---

---

**Description**

Reconstruct signal(s) from filtered Fourier coefficients using inverse GFT. Handles both single signals and multiple signals efficiently. Returns detailed inverse transform results including low-pass, band-pass approximations, reconstructed signal(s), and reconstruction error(s).

**Usage**

```
sgwt_inverse(sgwt_decomp, eigenvectors, original_signal = NULL)
```

**Arguments**

`sgwt_decomp` SGWT decomposition object from `sgwt_forward`  
`eigenvectors` Eigenvectors of the graph Laplacian (for inverse GFT)  
`original_signal` Original signal vector OR matrix (`n_vertices` x `n_signals`) for error calculation (optional)

**Value**

List containing:

**vertex\_approximations** Named list with inverse-transformed signals in vertex domain:

- `low_pass`: Low-pass (scaling) approximation
- `wavelet_1`, `wavelet_2`, etc.: Band-pass (wavelet) approximations by scale

**reconstructed\_signal** Full reconstructed signal (vector or matrix)

**reconstruction\_error** RMSE (scalar for single signal, vector for multiple signals)

**Examples**

```
## Not run:
# Single signal
inverse_result <- sgwt_inverse(sgwt_decomp, eigenvectors, original_signal)

# Multiple signals (batch processing)
inverse_result <- sgwt_inverse(sgwt_decomp, eigenvectors, original_signals_matrix)

## End(Not run)
```

---

```
simulate_checkerboard
      Simulate checkerboard pattern
```

---

**Description**

Generate a checkerboard pattern with alternating signals

**Usage**

```
simulate_checkerboard(grid_size = 8, tile_size = 1)
```

**Arguments**

<code>grid_size</code>	Number of tiles per row/column (default: 8)
<code>tile_size</code>	Resolution of each tile in pixels per side (default: 1)

**Value**

Data frame with X, Y coordinates and `signal_1`, `signal_2` patterns

**Examples**

```
## Not run:
# Generate 8x8 checkerboard with 10x10 pixel tiles
df <- simulate_checkerboard(grid_size = 8, tile_size = 10)
p <- visualize_checkerboard(df)
print(p)

## End(Not run)
```

---

simulate\_multiscale

*Simulate Multiple Center Patterns*


---

## Description

Generate spatial patterns with multiple circular centers at different scales. Creates concentric circle patterns with inner circle A and outer ring B at various radius combinations.

## Usage

```
simulate_multiscale(
  grid_size = 60,
  n_centers = 3,
  Ra_seq = c(10, 5, 1),
  Rb_seq = c(10, 5, 1),
  seed = 123,
  verbose = TRUE
)
```

## Arguments

grid_size	Size of the spatial grid (default: 60)
n_centers	Number of pattern centers to generate (default: 3)
Ra_seq	Vector of inner circle radii (default: c(10, 5, 1))
Rb_seq	Vector of outer ring radii (default: c(10, 5, 1))
seed	Random seed for reproducible center placement (default: 123)
verbose	Logical; if TRUE, show progress bar and messages (default: TRUE)

## Value

List of data frames, each containing X, Y coordinates and circleA, circleB binary signals

## Examples

```
## Not run:
# Generate multi-center patterns with default parameters
patterns <- simulate_multiscale()

# Custom parameters
Ra_seq <- seq(from = 10, to = 3, length.out = 6)
Rb_seq <- seq(from = 20, to = 3, length.out = 6)
patterns <- simulate_multiscale(Ra_seq = Ra_seq, Rb_seq = Rb_seq, n_centers = 3)

## End(Not run)
```

---

```
simulate_stripe_patterns
      Simulate Stripe Patterns
```

---

### Description

Generate stripe patterns with two parallel stripes separated by a gap. Creates rotatable stripe patterns with configurable gap, width, and rotation angle.

### Usage

```
simulate_stripe_patterns(
  grid_size = 100,
  gap_seq = c(10),
  width_seq = c(5),
  theta_seq = c(0),
  eps = 1e-09,
  verbose = TRUE
)
```

### Arguments

grid_size	Size of the spatial grid (default: 100)
gap_seq	Vector of gap distances between stripe centers (default: c(10))
width_seq	Vector of stripe widths (default: c(5))
theta_seq	Vector of rotation angles in degrees (default: c(0))
eps	Small numeric value for open boundary conditions to avoid overlap at stripe edges (default: 1e-9)
verbose	Logical; if TRUE, show progress messages (default: TRUE)

### Value

List of data frames, each containing X, Y coordinates and signal\_1, signal\_2 binary signals

### Examples

```
## Not run:
# Generate stripe patterns with default parameters
patterns <- simulate_stripe_patterns()

# Custom parameters
patterns <- simulate_stripe_patterns(
  grid_size = 80,
  gap_seq = c(10, 20),
  width_seq = c(5, 10, 20),
  theta_seq = c(0, 30, 60),
  eps = 1e-9,
  verbose = TRUE
)

## End(Not run)
```



---

```
visualize_checkerboard
```

*Visualize checkerboard pattern*

---

**Description**

Create a visualization of checkerboard pattern data

**Usage**

```
visualize_checkerboard(df, color1 = "black", color2 = "white")
```

**Arguments**

df	Data frame with X, Y coordinates and signal_1, signal_2 columns
color1	Color for signal_1 tiles (default: "black")
color2	Color for signal_2 tiles (default: "white")

**Value**

ggplot object showing the checkerboard pattern

**Examples**

```
## Not run:
df <- simulate_checkerboard(grid_size = 6, tile_size = 5)
p <- visualize_checkerboard(df, color1 = "darkblue", color2 = "lightgray")
print(p)

## End(Not run)
```

---

```
visualize_multiscale
```

*Visualize Multiple Center Simulation Results*

---

**Description**

Create visualization plots for multiple center simulation patterns

**Usage**

```
visualize_multiscale(
  sim_data,
  Ra_seq,
  Rb_seq,
  bg_color = "grey",
  signal1_color = "red",
  signal2_color = "blue",
  show_title = TRUE
)
```

**Arguments**

<code>sim_data</code>	Output from <code>simulate_multiscale</code> function
<code>Ra_seq</code>	Vector of Ra values used in simulation
<code>Rb_seq</code>	Vector of Rb values used in simulation
<code>bg_color</code>	Background color for plots (default: "grey")
<code>signal1_color</code>	Color for signal 1 (default: "red")
<code>signal2_color</code>	Color for signal 2 (default: "blue")
<code>show_title</code>	Logical; if TRUE (default), add titles to plots with Ra and Rb values

**Value**

Combined ggplot object with all pattern visualizations

**Examples**

```
## Not run:
# Generate and visualize patterns
Ra_seq <- seq(from = 10, to = 3, length.out = 6)
Rb_seq <- seq(from = 20, to = 3, length.out = 6)
sim_data <- simulate_multiscale(Ra_seq = Ra_seq, Rb_seq = Rb_seq, n_centers = 3)
plot_grid <- visualize_multiscale(sim_data, Ra_seq, Rb_seq)
print(plot_grid)

## End(Not run)
```

---

```
visualize_sgwt_kernels
```

*Visualize SGWT kernels and scaling functions*

---

**Description**

Visualize the scaling function and wavelet kernels used in SGWT based on the eigenvalue spectrum and selected parameters

**Usage**

```
visualize_sgwt_kernels(
  eigenvalues,
  scales = NULL,
  J = 4,
  scaling_factor = 2,
  kernel_type = "heat",
  lmax = NULL,
  eigenvalue_range = NULL,
  resolution = 1000
)
```

**Arguments**

**eigenvalues**    Vector of eigenvalues from graph Laplacian  
**scales**         Vector of scales for the wavelets (if NULL, auto-generated)  
**J**                Number of scales to generate if scales is NULL (default: 4)  
**scaling\_factor**    Scaling factor between consecutive scales (default: 2)  
**kernel\_type**    Type of wavelet kernel ("mexican\_hat" or "meyer", default: "mexican\_hat")  
**lmax**            Maximum eigenvalue (optional, computed if NULL)  
**eigenvalue\_range**    Range of eigenvalues to plot (default: full range)  
**resolution**     Number of points for smooth curve plotting (default: 1000)

**Value**

List containing the filter visualization plot and filter values

**Examples**

```
## Not run:
# Generate some example eigenvalues
eigenvals <- seq(0, 2, length.out = 100)

# Visualize kernels with specific parameters
viz_result <- visualize_sgwt_kernels(
  eigenvalues = eigenvals,
  J = 4,
  scaling_factor = 2,
  kernel_type = "heat"
)
print(viz_result$plot)

## End(Not run)
```

---

```
visualize_similarity_xy
```

*Visualize similarity in low vs non-low frequency space*

---

**Description**

Create a scatter plot with low-frequency similarity (c\_low) on x-axis and non-low-frequency similarity (c\_nonlow) on y-axis from runSGCC results

**Usage**

```
visualize_similarity_xy(
  similarity_results,
  point_size = 2,
  point_color = "steelblue",
  add_diagonal = TRUE,
  add_axes_lines = TRUE,
```

```

    title = "Low-frequency vs Non-low-frequency Similarity",
    show_labels = FALSE,
    show_names = FALSE
  )

```

### Arguments

<code>similarity_results</code>	List of similarity results from runSGCC function, or a single result
<code>point_size</code>	Size of points in the plot (default: 2)
<code>point_color</code>	Color of points (default: "steelblue")
<code>add_diagonal</code>	Whether to add diagonal reference lines (default: TRUE)
<code>add_axes_lines</code>	Whether to add x=0 and y=0 reference lines (default: TRUE)
<code>title</code>	Plot title (default: "Low-frequency vs Non-low-frequency Similarity")
<code>show_labels</code>	Whether to show point labels if names are available (default: FALSE)
<code>show_names</code>	Whether to display data point names as text labels using ggrepel (default: FALSE). If more than 50 points, randomly samples 50 for labeling. Requires ggrepel package.

### Value

ggplot object showing similarity space visualization

### Examples

```

## Not run:
# Single similarity result
sim_result <- runSGCC("signal1", "signal2", SG = SG_object)
plot <- visualize_similarity_xy(sim_result)
print(plot)

# Multiple similarity results
sim_results <- list(
  pair1 = runSGCC("signal1", "signal2", SG = SG_object1),
  pair2 = runSGCC("signal1", "signal2", SG = SG_object2)
)
plot <- visualize_similarity_xy(sim_results, show_names = TRUE)
print(plot)

# Show both labels and names (for comparison)
plot_both <- visualize_similarity_xy(sim_results, show_labels = TRUE, show_names = TRUE)
print(plot_both)

# With many data points (>50), names will be randomly sampled
# install.packages("ggrepel") # Required for show_names = TRUE
plot_many <- visualize_similarity_xy(many_sim_results, show_names = TRUE)
print(plot_many)

## End(Not run)

```

---

visualize\_stripe\_patterns

*Visualize Stripe Pattern Simulation Results*


---

## Description

Create visualization plots for stripe pattern simulation results

## Usage

```
visualize_stripe_patterns (
  sim_data,
  gap_seq,
  width_seq,
  theta_seq,
  bg_color = "grey",
  signal1_color = "#1f6f8b",
  signal2_color = "#e67e22",
  overlap_color = "#7a4dbf",
  show_title = TRUE
)
```

## Arguments

sim_data	Output from simulate_stripe_patterns function
gap_seq	Vector of gap values used in simulation
width_seq	Vector of width values used in simulation
theta_seq	Vector of theta (rotation angle) values used in simulation
bg_color	Background color for plots (default: "grey")
signal1_color	Color for signal 1 (default: "#1f6f8b")
signal2_color	Color for signal 2 (default: "#e67e22")
overlap_color	Color for overlapping regions (default: "#7a4dbf")
show_title	Logical; if TRUE (default), add titles to plots with parameter values

## Value

Combined ggplot object with all pattern visualizations

## Examples

```
## Not run:
# Generate and visualize patterns
sim_data <- simulate_stripe_patterns(
  grid_size = 80,
  gap_seq = c(10, 20),
  width_seq = c(5, 10, 20),
  theta_seq = c(0, 30, 60)
```

```
)  
plot_grid <- visualize_stripe_patterns(sim_data,  
                                     gap_seq = c(10, 20),  
                                     width_seq = c(5, 10, 20),  
                                     theta_seq = c(0, 30, 60))  
  
print(plot_grid)  
  
## End(Not run)
```

# Index

- \* **CODEX**
  - codex\_toy\_data, 5
- \* **SGWT**
  - codex\_toy\_data, 5
- \* **biological-data**
  - BioGSP-package, 2
- \* **datasets**
  - codex\_toy\_data, 5
- \* **graph-theory**
  - BioGSP-package, 2
- \* **internal**
  - sgwt-globals, 19
- \* **package**
  - BioGSP-package, 2
- \* **spatial-analysis**
  - BioGSP-package, 2
- \* **spatial**
  - codex\_toy\_data, 5
- \* **wavelets**
  - BioGSP-package, 2
- \_PACKAGE (*BioGSP-package*), 2
- BioGSP-package, 2
- cal\_laplacian, 4
- codex\_toy\_data, 5
- compare\_kernel\_families, 7
- compute\_sgwt\_filters, 8
- cosine\_similarity, 9
- demo\_sgwt, 3, 9
- FastDecompositionLap, 10
- find\_knee\_point, 11
- gft, 11
- hello\_sgwt, 12
- igft, 12
- initSGWT, 3, 13
- install\_and\_load, 14
- plot\_FM, 14
- plot\_sgwt\_decomposition, 3, 15
- print.SGWT, 16
- runSGCC, 3, 16
- runSGWT, 3, 17
- runSpecGraph, 3, 18
- sgwt-globals, 19
- sgwt\_auto\_scales, 19
- sgwt\_energy\_analysis, 3, 19
- sgwt\_forward, 3, 20
- sgwt\_get\_kernels, 21
- sgwt\_inverse, 3, 21
- simulate\_checkerboard, 22
- simulate\_multiscale, 23
- simulate\_stripe\_patterns, 24
- visualize\_checkerboard, 25
- visualize\_multiscale, 25
- visualize\_sgwt\_kernels, 26
- visualize\_similarity\_xy, 27
- visualize\_stripe\_patterns, 29