

# Package: BioGSP (via r-universe)

June 3, 2026

**Type** Package

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

**Version** 1.0.0

**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) <[doi:10.1016/j.acha.2010.04.005](https://doi.org/10.1016/j.acha.2010.04.005)>. Provides tools for multi-scale analysis of biology spatial signals, including forward and inverse transforms, energy analysis, and visualization functions tailored for biological applications. Biological application example is on Stephanie, Yao, Yuzhou (2024) <[doi:10.1101/2024.12.20.629650](https://doi.org/10.1101/2024.12.20.629650)>.

**License** GPL-3

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

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

**Encoding** UTF-8

**RoxygenNote** 7.3.2

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

**Suggests** knitr, rmarkdown, ggrepel

**VignetteBuilder** knitr

**Depends** R (>= 3.5.0)

**Config/pak/sysreqs** libgmp-dev libxml2-dev

**Repository** <https://bmineer.r-universe.dev>

**Date/Publication** 2026-01-27 22:27:38 UTC

**RemoteUrl** <https://github.com/bmineer/biogsp>

**RemoteRef** HEAD

**RemoteSha** 5ed12ed48d20cc3258e18dbb329d6e660eed9bd3

## Contents

BioGSP-package . . . . .	3
cal_laplacian . . . . .	4
checkKband . . . . .	5
codex_toy_data . . . . .	6
compare_kernel_families . . . . .	9
compute_sgwt_filters . . . . .	10
cosine_similarity . . . . .	10
demo_sgwt . . . . .	11
FastDecompositionLap . . . . .	12
find_knee_point . . . . .	13
gft . . . . .	13
hello_sgwt . . . . .	14
igft . . . . .	14
initSGWT . . . . .	15
plot_FM . . . . .	16
plot_sgwt_decomposition . . . . .	17
print.SGWT . . . . .	17
runSGCC . . . . .	18
runSGWT . . . . .	19
runSpecGraph . . . . .	20
sgwt_auto_scales . . . . .	21
sgwt_energy_analysis . . . . .	22
sgwt_forward . . . . .	22
sgwt_get_kernels . . . . .	24
sgwt_inverse . . . . .	24
simulate_checkerboard . . . . .	25
simulate_moving_circles . . . . .	26
simulate_multiscale . . . . .	27
simulate_multiscale_overlap . . . . .	28
simulate_stripe_patterns . . . . .	29
visualize_checkerboard . . . . .	30
visualize_moving_circles . . . . .	31
visualize_multiscale . . . . .	32
visualize_sgwt_kernels . . . . .	33
visualize_similarity_xy . . . . .	34
visualize_stripe_patterns . . . . .	36

## Index

38

---

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

**Examples**

```
# 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", J = 4, kernel_type = "heat")
SG <- runSpecGraph(SG, k = 8)
SG <- runSGWT(SG)

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

---

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**

```
W <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cal_laplacian(W, type = "normalized")
```

---

checkKband	<i>Check K-band limited property of signals</i>
------------	---

---

**Description**

Analyze whether signals are k-band limited by comparing low-frequency and high-frequency Fourier coefficients using eigendecomposition and statistical testing. Builds graph and computes Laplacian directly from SGWT data.

**Usage**

```
checkKband(
  SG,
  signals = NULL,
  alpha = 0.05,
  verbose = TRUE,
  k = 25,
  laplacian_type = "normalized"
)
```

**Arguments**

**SG** SGWT object with Data slot (from initSGWT)  
**signals** Character vector of signal names to analyze. If NULL, uses all signals from SG\$Data\$signals  
**alpha** Significance level for Wilcoxon test (default: 0.05)  
**verbose** Logical; if TRUE, print progress messages (default: TRUE)  
**k** Number of nearest neighbors for graph construction (default: 25)  
**laplacian\_type** Type of Laplacian ("unnormalized", "normalized", or "randomwalk") (default: "normalized")

**Value**

List containing:

**is\_kband\_limited** Logical; TRUE if all signals are k-band limited

**knee\_point\_low** Integer; knee point index for low-frequency eigenvalues

**knee\_point\_high** Integer; knee point index for high-frequency eigenvalues

**signal\_results** List with per-signal test results including p-values and Fourier coefficients

**Examples**

```
# Create example data
data <- data.frame(x = runif(100), y = runif(100),
                  signal1 = rnorm(100), signal2 = rnorm(100))

# Initialize SGWT object (no need to run runSpecGraph)
SG <- initSGWT(data, signals = c("signal1", "signal2"))

# Check k-band limited property
result <- checkKband(SG, signals = c("signal1", "signal2"), k = 30)
if (result$is_kband_limited) {
  cat("All signals are k-band limited")
}
```

---

codex\_toy\_data

*Toy CODEX Spatial Cell Type Data*

---

**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

# 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 using new workflow
SG <- initSGWT(sgwt_data, signals = "signal", J = 3, kernel_type = "heat")
SG <- runSpecGraph(SG, k = 8)
SG <- runSGWT(SG)

# View results
print(SG)
```

---

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

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

## 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**

<code>eigenvalues</code>	Eigenvalues of the graph Laplacian
<code>scales</code>	Vector of scales for the wavelets
<code>lmax</code>	Maximum eigenvalue (optional)
<code>kernel_type</code>	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**

```
SG <- demo_sgwt()
print(SG)
```

---

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

```
# 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 = 2)
```

---

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**

```

# Create example data
data <- data.frame(x = runif(50), y = runif(50), signal = rnorm(50))
SG <- initSGWT(data, signals = "signal")
SG <- runSpecGraph(SG, k = 10)
eigenvectors <- SG$Graph$eigenvectors

# Single signal - use GFT to get Fourier coefficients
fourier_coefs <- gft(data$signal, eigenvectors)
signal_reconstructed <- igft(fourier_coefs, eigenvectors)

# Multiple signals (batch processing)
signals_matrix <- cbind(data$signal, data$signal * 2)
fourier_coefs_matrix <- gft(signals_matrix, eigenvectors)
signals_reconstructed <- igft(fourier_coefs_matrix, eigenvectors)

```

---

initSGWT

*Initialize SGWT object*


---

**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**

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

**Value**

SGWT object with Data and Parameters slots initialized

**Examples**

```
# 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"))
```

---

plot\_FM

*Plot Fourier modes (eigenvectors) from SGWT object*

---

**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**

```
# Create example data
data <- data.frame(x = runif(100), y = runif(100), signal = rnorm(100))

# Plot both low and high frequency modes
SG <- initSGWT(data, signals = "signal")
SG <- runSpecGraph(SG, k = 15)
plot_FM(SG, mode_type = "both", n_modes = 4)

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

---

`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**

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

**Value**

ggplot object with combined plots

**Examples**

```
# Create and analyze example data
data <- data.frame(x = runif(100), y = runif(100), signal1 = rnorm(100))
SG <- initSGWT(data, signals = "signal1")
SG <- runSpecGraph(SG, k = 15)
SG <- runSGWT(SG)

# Plot decomposition
plots <- plot_sgwt_decomposition(SG, signal_name = "signal1")
print(plots)
```

---

`print.SGWT`*Print method for SGWT objects*

---

**Description**

Print method for SGWT objects

**Usage**

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

**Arguments**

x                   SGWT object to print  
...                  Additional arguments passed to print methods

**Value**

Invisibly returns the input SGWT object. Called for side effects (prints object summary to console).

---

runSGCC

*Run SGCC weighted similarity analysis in Fourier domain*


---

**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**

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

**Value**

Similarity analysis results computed in Fourier domain

**Examples**

```
# Create example data and compute SGWT
data <- data.frame(x = runif(100), y = runif(100),
                  signal1 = rnorm(100), signal2 = rnorm(100))
SG <- initSGWT(data, signals = c("signal1", "signal2"))
SG <- runSpecGraph(SG, k = 15)
SG <- runSGWT(SG)

# Between two signals in same SGWT object
similarity <- runSGCC("signal1", "signal2", SG = SG)
print(similarity)

# Between two SGWT objects
data2 <- data.frame(x = runif(100), y = runif(100), signal = rnorm(100))
SG2 <- initSGWT(data2, signals = "signal")
SG2 <- runSpecGraph(SG2, k = 15)
SG2 <- runSGWT(SG2)

similarity2 <- runSGCC(SG, SG2)
print(similarity2)
```

---

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**

```
# Create example data
data <- data.frame(x = runif(100), y = runif(100), signal = rnorm(100))
SG <- initSGWT(data, signals = "signal")
SG <- runSpecGraph(SG, k = 15)

# Uses batch processing by default
SG <- runSGWT(SG)

# Or force individual processing
SG2 <- initSGWT(data, signals = "signal")
SG2 <- runSpecGraph(SG2, k = 15)
SG2 <- runSGWT(SG2, use_batch = FALSE)
```

---

runSpecGraph

*Build spectral graph for SGWT object*


---

**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 <code>initSGWT()</code>
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**

```
# Create example data
data <- data.frame(x = runif(100), y = runif(100), signal = rnorm(100))
SG <- initSGWT(data, signals = "signal")

# Uses full length by default
SG <- runSpecGraph(SG, k = 30, laplacian_type = "normalized")

# Or specify custom length
SG2 <- initSGWT(data, signals = "signal")
SG2 <- runSpecGraph(SG2, k = 30, laplacian_type = "normalized",
                    length_eigenvalue = 30)
```

---

sgwt\_auto\_scales

*Generate automatic scales for SGWT*


---

**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    *Analyze SGWT energy distribution across scales in Fourier domain*

---

### 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

```
# Create and analyze example data
data <- data.frame(x = runif(100), y = runif(100), signal1 = rnorm(100))
SG <- initSGWT(data, signals = "signal1")
SG <- runSpecGraph(SG, k = 15)
SG <- runSGWT(SG)

# Analyze energy distribution
energy_analysis <- sgwt_energy_analysis(SG, signal_name = "signal1")
print(energy_analysis)
```

---

sgwt\_forward                    *Forward SGWT transform (single or batch)*

---

### 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**

```
# Create example data and compute graph
data <- data.frame(x = runif(50), y = runif(50), signal = rnorm(50))
SG <- initSGWT(data, signals = "signal", J = 3)
SG <- runSpecGraph(SG, k = 10)
eigenvectors <- SG$Graph$eigenvectors
eigenvalues <- SG$Graph$eigenvalues
scales <- SG$Parameters$scales

# Single signal
signal <- data$signal
result <- sgwt_forward(signal, eigenvectors, eigenvalues, scales)

# Multiple signals (batch processing)
signals_matrix <- cbind(data$signal, data$signal * 2, data$signal * 0.5)
result <- sgwt_forward(signals_matrix, eigenvectors, eigenvalues, scales)
```

---

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**

```
# Create example data and perform forward transform
data <- data.frame(x = runif(50), y = runif(50), signal = rnorm(50))
SG <- initSGWT(data, signals = "signal", J = 3)
SG <- runSpecGraph(SG, k = 10)
eigenvectors <- SG$Graph$eigenvectors
eigenvalues <- SG$Graph$eigenvalues
scales <- SG$Parameters$scales

# Single signal - forward transform first
original_signal <- data$signal
sgwt_decomp <- sgwt_forward(original_signal, eigenvectors, eigenvalues, scales)
inverse_result <- sgwt_inverse(sgwt_decomp, eigenvectors, original_signal)

# Multiple signals (batch processing)
original_signals_matrix <- cbind(data$signal, data$signal * 2)
sgwt_decomp <- sgwt_forward(original_signals_matrix, eigenvectors, eigenvalues, scales)
inverse_result <- sgwt_inverse(sgwt_decomp, eigenvectors, original_signals_matrix)
```

---

simulate\_checkerboard *Simulate checkerboard pattern*

---

**Description**

Generate a checkerboard pattern with alternating signals

**Usage**

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

**Arguments**

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

**Value**

Data frame with X, Y coordinates and signal\_1, signal\_2 patterns

**Examples**

```
# Generate 8x8 checkerboard with 10x10 pixel tiles
df <- simulate_checkerboard(grid_size = 8, tile_size = 10)
p <- visualize_checkerboard(df)
print(p)
```

---

simulate\_moving\_circles

*Simulate Moving Circles Pattern*

---

**Description**

Generate patterns of two circles moving toward each other horizontally. Creates mutually exclusive signals where overlapping pixels are assigned to signal\_1 (circle 1). The circles start at fixed horizontal distances from the midline and move toward the center.

**Usage**

```
simulate_moving_circles(
  grid_size = 60,
  radius_seq = 6:14,
  n_steps = 10,
  center_distance = 30,
  radius2_factor = 1.5,
  seed = NULL,
  verbose = TRUE
)
```

**Arguments**

grid_size	Size of the spatial grid (default: 60)
radius_seq	Vector of radii for circle 1 (default: 6:14)
n_steps	Number of movement steps (default: 10)
center_distance	Initial horizontal distance from midline for both centers (default: 30)
radius2_factor	Circle 2 radius = radius_seq * radius2_factor (default: 1.5)
seed	Random seed for reproducibility (default: 123)
verbose	Logical; if TRUE, show progress bar and messages (default: TRUE)

**Value**

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

**Examples**

```
# Generate moving circles patterns with default parameters
patterns <- simulate_moving_circles()

# Custom parameters
patterns <- simulate_moving_circles(
  grid_size = 80,
  radius_seq = c(8, 12, 16),
  n_steps = 8,
  center_distance = 35,
  radius2_factor = 1.2
)
```

---

simulate\_multiscale    *Simulate Multi-center Multi-scale Concentric Ring Patterns*

---

**Description**

Generate multi-center, multi-scale concentric ring simulation data. Creates patterns with inner circles and outer rings where the outer radius shrinks from a fixed starting point to a factor of the inner radius across multiple steps.

**Usage**

```
simulate_multiscale(
  grid_size = 60,
  Ra_seq = seq(2.5, 20, by = 2.5),
  n_steps = 10,
  n_centers = 1,
  outer_start = 40,
  outer_end_factor = 1.2,
  seed = NULL,
  verbose = TRUE
)
```

**Arguments**

grid_size	Size of the spatial grid (default: 60)
Ra_seq	Vector of inner circle radii (default: seq(2.5, 20, by = 2.5))
n_steps	Number of outer radius shrinkage steps (default: 10)
n_centers	Number of circle centers (default: 1)
outer_start	Fixed starting outer radius (default: 40)

outer\_end\_factor      Outer radius shrinks to this factor \* Ra (default: 1.2)  
 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 signal\_1, signal\_2 binary signals

**Examples**

```
# Generate multi-center patterns with default parameters
patterns <- simulate_multiscale()

# Custom parameters
patterns <- simulate_multiscale(
  grid_size = 80,
  Ra_seq = seq(5, 25, by = 5),
  n_steps = 8,
  n_centers = 2,
  outer_start = 50
)
```

---

simulate\_multiscale\_overlap

*Simulate Multiple Center Patterns with Fixed Centers*

---

**Description**

Generate spatial patterns with multiple circular centers at fixed positions. Similar to `simulate_multiscale` but with centers placed at fixed locations for reproducible pattern generation. Creates concentric circle patterns with inner circle A and outer ring B at various radius combinations.

**Usage**

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

**Arguments**

grid_size	Size of the spatial grid (default: 60)
n_centers	Number of pattern centers to generate. If 1, center is placed at grid center. If > 1, centers are randomly placed but fixed by seed (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 signal\_1, signal\_2 binary signals

**Examples**

```
# Generate multi-center patterns with fixed centers
patterns <- simulate_multiscale_overlap()

# Single center at grid center
patterns_single <- simulate_multiscale_overlap(n_centers = 1)

# Custom parameters with multiple centers
Ra_seq <- seq(from = 10, to = 3, length.out = 4)
Rb_seq <- seq(from = 15, to = 2, length.out = 4)
patterns <- simulate_multiscale_overlap(
  Ra_seq = Ra_seq,
  Rb_seq = Rb_seq,
  n_centers = 2,
  seed = 456
)
```

---

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**

```
# 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  
)
```

---

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**

```
df <- simulate_checkerboard(grid_size = 6, tile_size = 5)
p <- visualize_checkerboard(df, color1 = "darkblue", color2 = "lightgray")
print(p)
```

---

visualize\_moving\_circles

*Visualize Moving Circles Pattern*

---

**Description**

Visualize the simulated moving circles patterns from simulate\_moving\_circles

**Usage**

```
visualize_moving_circles(  
  sim_data,  
  bg_color = "grey90",  
  signal1_color = "#16964a",  
  signal2_color = "#2958a8",  
  show_subtitle = TRUE,  
  sort_order = c("ascending", "descending"),  
  panel_spacing = 0.1,  
  title_size = 12  
)
```

**Arguments**

<code>sim_data</code>	Output from <code>simulate_moving_circles</code> function
<code>bg_color</code>	Background color for plots (default: "grey90")
<code>signal1_color</code>	Color for signal 1 (default: "#16964a")
<code>signal2_color</code>	Color for signal 2 (default: "#2958a8")
<code>show_subtitle</code>	Logical; if TRUE (default), show parameter values in facet labels
<code>sort_order</code>	Order for sorting ("ascending" or "descending", default: "ascending")
<code>panel_spacing</code>	Control spacing between panels in lines (default: 0.1)
<code>title_size</code>	Size of title text (default: 12)

**Value**

ggplot object with faceted visualization

**Examples**

```
# Generate and visualize patterns
sim_data <- simulate_moving_circles(
  radius_seq = 6:14,
  n_steps = 10
)
plot_grid <- visualize_moving_circles(sim_data)
print(plot_grid)
```

---

`visualize_multiscale` *Visualize Multi-center Multi-scale Concentric Ring Patterns*

---

**Description**

Visualize the simulated concentric ring patterns from `simulate_multiscale`

**Usage**

```
visualize_multiscale(
  sim_data,
  Ra_seq,
  n_steps,
  bg_color = "grey90",
  signal1_color = "#16964a",
  signal2_color = "#2958a8",
  show_subtitle = TRUE,
  sort_order = c("ascending", "descending"),
  panel_spacing = 0.1,
  title_size = 12
)
```

**Arguments**

<code>sim_data</code>	Output from <code>simulate_multiscale</code> function
<code>Ra_seq</code>	Vector of Ra values used in simulation
<code>n_steps</code>	Number of steps used in simulation
<code>bg_color</code>	Background color for plots (default: "grey90")
<code>signal1_color</code>	Color for signal 1 (default: "#16964a")
<code>signal2_color</code>	Color for signal 2 (default: "#2958a8")
<code>show_subtitle</code>	Logical; if TRUE (default), show parameter values in facet labels
<code>sort_order</code>	Order for sorting ("ascending" or "descending", default: "ascending")
<code>panel_spacing</code>	Control spacing between panels in lines (default: 0.1)
<code>title_size</code>	Size of title text (default: 12)

**Value**

ggplot object with faceted visualization

**Examples**

```
# Generate and visualize patterns
sim_data <- simulate_multiscale(
  Ra_seq = seq(2.5, 20, by = 2.5),
  n_steps = 10
)
plot_grid <- visualize_multiscale(sim_data,
                                 Ra_seq = seq(2.5, 20, by = 2.5),
                                 n_steps = 10)
print(plot_grid)
```

---

`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

```

# Generate some example eigenvalues
eigenvals <- seq(0, 2, length.out = 100)

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

```

---

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**

similarity_results	List of similarity results from runSGCC function, or a single result
point_size	Size of points in the plot (default: 2)
point_color	Color of points (default: "steelblue")
add_diagonal	Whether to add diagonal reference lines (default: TRUE)
add_axes_lines	Whether to add x=0 and y=0 reference lines (default: TRUE)
title	Plot title (default: "Low-frequency vs Non-low-frequency Similarity")
show_labels	Whether to show point labels if names are available (default: FALSE)
show_names	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**

```
# Create example data and compute SGWT
data <- data.frame(x = runif(100), y = runif(100),
  signal1 = rnorm(100), signal2 = rnorm(100))
SG <- initSGWT(data, signals = c("signal1", "signal2"))
SG <- runSpecGraph(SG, k = 15)
SG <- runSGWT(SG)

# Single similarity result
sim_result <- runSGCC("signal1", "signal2", SG = SG)
plot <- visualize_similarity_xy(sim_result)
print(plot)

# Multiple similarity results (create two different analyses)
data2 <- data.frame(x = runif(100), y = runif(100),
  signal1 = rnorm(100), signal2 = rnorm(100))
SG2 <- initSGWT(data2, signals = c("signal1", "signal2"))
```

```

SG2 <- runSpecGraph(SG2, k = 15)
SG2 <- runSGWT(SG2)

sim_results <- list(
  pair1 = runSGCC("signal1", "signal2", SG = SG),
  pair2 = runSGCC("signal1", "signal2", SG = SG2)
)
plot <- visualize_similarity_xy(sim_results, show_names = TRUE)
print(plot)

```

---

```
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

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

**Value**

Combined ggplot object with all pattern visualizations

**Examples**

```
# 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)
```

# Index

- \* **CODEX**
  - codex\_toy\_data, 6
- \* **SGWT**
  - codex\_toy\_data, 6
- \* **biological-data**
  - BioGSP-package, 3
- \* **datasets**
  - codex\_toy\_data, 6
- \* **graph-theory**
  - BioGSP-package, 3
- \* **package**
  - BioGSP-package, 3
- \* **spatial-analysis**
  - BioGSP-package, 3
- \* **spatial**
  - codex\_toy\_data, 6
- \* **wavelets**
  - BioGSP-package, 3
- \_PACKAGE (BioGSP-package), 3
- BioGSP-package, 3
- cal\_laplacian, 4
- checkKband, 5
- codex\_toy\_data, 6
- compare\_kernel\_families, 9
- compute\_sgwt\_filters, 10
- cosine\_similarity, 10
- demo\_sgwt, 3, 11
- FastDecompositionLap, 12
- find\_knee\_point, 13
- gft, 13
- hello\_sgwt, 14
- igft, 14
- initSGWT, 3, 15
- plot\_FM, 16
- plot\_sgwt\_decomposition, 3, 17
- print.SGWT, 17
- runSGCC, 3, 18
- runSGWT, 3, 19
- runSpecGraph, 3, 20
- sgwt\_auto\_scales, 21
- sgwt\_energy\_analysis, 3, 22
- sgwt\_forward, 3, 22
- sgwt\_get\_kernels, 24
- sgwt\_inverse, 3, 24
- simulate\_checkerboard, 25
- simulate\_moving\_circles, 26
- simulate\_multiscale, 27
- simulate\_multiscale\_overlap, 28
- simulate\_stripe\_patterns, 29
- visualize\_checkerboard, 30
- visualize\_moving\_circles, 31
- visualize\_multiscale, 32
- visualize\_sgwt\_kernels, 33
- visualize\_similarity\_xy, 34
- visualize\_stripe\_patterns, 36