Package 'BioGSP'

September 28, 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

Encoding UTF-8

RoxygenNote 7.2.3

Imports Matrix, igraph, RANN, RSpectra, ggplot2, patchwork, ggpubr, viridis, methods

Suggests spdep, spatialEco, sp, gasper, egg, testthat (>= 2.1.0), remotes, dplyr, tidyr, knitr, rmarkdown

VignetteBuilder knitr

Depends R (>= 3.5.0)

NeedsCompilation no

Contents

`BioGSP-package` *BioGSP: Biological Graph Signal Processing for Spatial Data Analysis*

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.

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` *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
## 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

`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

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()
```

Value

SGWT object with complete analysis

Examples

```
## Not run:
SG <- demo_sgwt()
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_neighbor = 25,
  which = "LM",
  sigma = NULL,
  opts = list(),
  lower = TRUE,
  ...
)
```

Arguments

laplacianMat	Laplacian matrix
k_neighbor	Number of neighbors for eigenvalue computation (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_neighbor = 25)

## End(Not run)
```

---

find_knee_point *Find knee point in a curve*

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 *Graph Fourier Transform*

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()
```

`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,
  k = 25,
  scales = NULL,
  J = 5,
  scaling_factor = 2,
  kernel_type = "heat",
  laplacian_type = "normalized"
)
```

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.
`k`	Number of nearest neighbors for graph construction (default: 25)
`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")
`laplacian_type`	Type of graph Laplacian ("unnormalized", "normalized", or "randomwalk") (default: "normalized")

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` *Install and load packages*

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` *Plot frequency modes*

Description

Plot frequency modes from graph Fourier analysis

Usage

```
plot_FM(input = NULL, FM_idx = c(1:20), ncol = 5)
```

Arguments

`input`	Input data (currently not used, for future compatibility)
`FM_idx`	Indices of frequency modes to plot (default: 1:20)
`ncol`	Number of columns in plot layout (default: 5)

Value

Combined plot of frequency modes

Examples

```
## Not run:
# This function requires specific data structure (df_hex_combine)
# plot_FM(FM_idx = 1:10, ncol = 5)

## 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)
```

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

runSGCC *Run SGCC weighted similarity analysis*

Description

Calculate energy-normalized weighted similarity between two signals from SGWT Forward results or between two SGWT objects.

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

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. Assumes Graph slot is populated by runSpecGraph().

Usage

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

Arguments

SG	SGWT object with Graph slot populated
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)

## End(Not run)
```

runSpecGraph *Build spectral graph for SGWT object*

Description

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

Usage

```
runSpecGraph(SG, verbose = TRUE)
```

Arguments

SG	SGWT object from initSGWT()
verbose	Whether to print progress messages (default: TRUE)

Value

Updated SGWT object with Graph slot populated

Examples

```
## Not run:
SG <- initSGWT(data)
SG <- runSpecGraph(SG)

## End(Not run)
```

sgwt-globals *Global variables used in ggplot2 aesthetics*

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 *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*

Description

Calculate and analyze energy distribution across different scales in the SGWT decomposition

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

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 *Forward SGWT transform*

Description

Decompose signal into inverse-transformed signals (vertex domain) using SGWT. Also 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
`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: "mexican_hat", options: "mexican_hat", "meyer", "heat")

Value

List containing:

coefficients Inverse-transformed signals in vertex domain (scaling + wavelet scales)

fourier_coefficients List with original and filtered Fourier coefficients

filters Filter bank used

scales Scales used

eigenvalues Graph Laplacian eigenvalues

eigenvectors Graph Laplacian eigenvectors

Examples

```
## Not run:
# Assuming you have eigenvalues, eigenvectors, and a signal
result <- sgwt_forward(signal, eigenvectors, eigenvalues, scales)
result_meyer <- sgwt_forward(signal, eigenvectors, eigenvalues, scales, kernel_type = "me
result_heat <- sgwt_forward(signal, eigenvectors, eigenvalues, scales, kernel_type = "hea

## End(Not run)
```

---

sgwt_get_kernels *Get a unified kernel family (low-pass and band-pass) by kernel_type*

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 *Inverse SGWT transform*

Description

Reconstruct signal from inverse-transformed signals (coefficients). Returns detailed inverse transform results including low-pass, band-pass approximations, reconstructed signal, and reconstruction error.

Usage

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

Arguments

sgwt_decomp	SGWT decomposition object from sgwt_forward
original_signal	Original signal for error calculation (optional)

Value

List containing:

low_pass_approximation Low-pass (scaling) approximation

band_pass_approximations List of band-pass (wavelet) approximations by scale

reconstructed_signal Full reconstructed signal

reconstruction_error RMSE if original_signal provided

Examples

```
## Not run:
# Assuming you have an SGWT decomposition
inverse_result <- sgwt_inverse(sgwt_decomp, original_signal)

## 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
)
```

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)

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_ringpattern

Simulate Concentric Ring Patterns

Description

Generate concentric ring patterns with dynamic outer ring movement. Creates a solid inner circle with a moving outer ring that closes in over time.

Usage

```
simulate_ringpattern(
  grid_size = 60,
  radius_seq = seq(2.5, 20, by = 2.5),
  n_movements = 10,
  center_x = NULL,
  center_y = NULL
)
```

Arguments

grid_size	Size of the spatial grid (default: 60)
radius_seq	Vector of inner circle radii to simulate (default: seq(2.5, 20, by = 2.5))
n_movements	Number of movement steps for the outer ring (default: 10)
center_x	X coordinate of pattern center (default: grid_size/2)
center_y	Y coordinate of pattern center (default: grid_size/2)

Value

List of data frames, each containing X, Y coordinates, movement indicators, and signal_1 (solid circle), signal_2 (concentric ring) binary signals

Examples

```
## Not run:
# Generate concentric ring patterns with default parameters
patterns <- simulate_ringpattern()

# Custom parameters
```

```
radius_seq <- seq(2.5, 20, by = 2.5)
patterns <- simulate_ringpattern(radius_seq = radius_seq, n_movements = 5)

## 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

`sim_data`	Output from simulate_multiscale function
`Ra_seq`	Vector of Ra values used in simulation
`Rb_seq`	Vector of Rb values used in simulation
`bg_color`	Background color for plots (default: "grey")
`signal1_color`	Color for signal 1 (default: "red")
`signal2_color`	Color for signal 2 (default: "blue")
`show_title`	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_ringpattern` *Visualize Concentric Ring Simulation Results*

Description

Create visualization plots for concentric ring simulation patterns

Usage

```
visualize_ringpattern(
  sim_data,
  radius_seq,
  bg_color = "grey",
  signal1_color = "#16964a",
  signal2_color = "#2958a8"
)
```

Arguments

`sim_data`	Output from simulate_ringpattern function
`radius_seq`	Vector of radius values used in simulation
`bg_color`	Background color for plots (default: "grey")
`signal1_color`	Color for signal 1 (default: "#16964a")
`signal2_color`	Color for signal 2 (default: "#2958a8")

Value

Combined ggplot object with all pattern visualizations

Examples

```
## Not run:
# Generate and visualize patterns
radius_seq <- seq(2.5, 20, by = 2.5)
sim_data <- simulate_ringpattern(radius_seq = radius_seq)
plot_grid <- visualize_ringpattern(sim_data, radius_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)
```

Index