

# Package ‘BioGSP’

September 15, 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, knitr, rmarkdown, testthat (>= 2.1.0), remotes, dplyr, tidyr

**VignetteBuilder** knitr

**Depends** R (>= 3.5.0)

## Contents

BioGSP-package . . . . .	2
Cal_Eigen . . . . .	4
Cal_GCC . . . . .	4
cal_laplacian . . . . .	5
codex_toy_data . . . . .	6
compare_kernel_families . . . . .	8
compute_sgwt_filters . . . . .	9
cosine_similarity . . . . .	10
demo_sgwt . . . . .	10
FastDecompositionLap . . . . .	11
find_knee_point . . . . .	12
gft . . . . .	12
hello_sgwt . . . . .	13

install_and_load . . . . .	13
plot_FM . . . . .	14
plot_sgwt_decomposition . . . . .	14
SGWT . . . . .	15
sgwt-globals . . . . .	16
sgwt_auto_scales . . . . .	17
sgwt_energy_analysis . . . . .	17
sgwt_forward . . . . .	18
sgwt_get_kernels . . . . .	19
sgwt_inverse . . . . .	19
visualize_sgwt_kernels . . . . .	20

<b>Index</b>	<b>22</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

[SGWT](#) Main function for SGWT analysis  
[sgwt\\_forward](#) Forward SGWT transform  
[sgwt\\_inverse](#) Inverse SGWT transform  
[sgwt\\_energy\\_analysis](#) Energy distribution analysis  
[plot\\_sgwt\\_decomposition](#) Visualization of SGWT components  
[Cal\\_GCC](#) Graph Cross-Correlation analysis  
[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))
)

# Apply SGWT
result <- SGWT(data, signal = "signal", k = 8, J = 4)

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

## End(Not run)
```

---

Cal_Eigen	<i>Calculate eigenvalues and eigenvectors with knee detection</i>
-----------	---

---

### Description

Calculate eigenvalues and eigenvectors of a spatial graph with automatic detection of the low-frequency cutoff using knee detection

### Usage

```
Cal_Eigen(data.in = NULL, k = 25, k_fold = 15, sensitivity = 2)
```

### Arguments

<code>data.in</code>	Data frame with x and y coordinates
<code>k</code>	Number of nearest neighbors (default: 25)
<code>k_fold</code>	Eigendecomposition parameter (default: 15)
<code>sensitivity</code>	Sensitivity parameter for knee detection (default: 2)

### Value

List containing knee point, eigenvectors, and eigenvalues

### Examples

```
## Not run:
# Create spatial data
data <- data.frame(x = runif(100), y = runif(100))
result <- Cal_Eigen(data, k = 10)

## End(Not run)
```

---

Cal_GCC	<i>Calculate Graph Cross-Correlation (GCC)</i>
---------	--

---

### Description

Calculate Graph Cross-Correlation between two signals using Graph Fourier Transform

### Usage

```
Cal_GCC(
  data.in = NULL,
  knee = NULL,
  signal1 = NULL,
  signal2 = NULL,
  eigenvector = NULL
)
```

**Arguments**

data.in	Data frame containing the signals
knee	Knee point for frequency cutoff
signal1	Name of first signal column
signal2	Name of second signal column
eigenvector	Matrix of eigenvectors

**Value**

Cosine similarity value

**Examples**

```
## Not run:
# Assuming you have data with two signals and eigenvectors
# gcc_value <- Cal_GCC(data, knee = 10, signal1 = "sig1", signal2 = "sig2", eigenvector =
## 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**

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 = "mexican_hat")

## 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: c(0, 3))
<code>scale_param</code>	Scale parameter for all functions (default: 1)
<code>plot_results</code>	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 = "mexican_hat"
)
```

**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	<i>Calculate cosine similarity between two vectors</i>
-------------------	--

---

**Description**

Calculate cosine similarity between two numeric vectors

**Usage**

```
cosine_similarity(x, y)
```

**Arguments**

x	First vector
y	Second vector

**Value**

Cosine similarity value (between -1 and 1)

**Examples**

```
x <- c(1, 2, 3)
y <- c(2, 3, 4)
similarity <- cosine_similarity(x, y)
```

---

demo_sgwt	<i>Demo function for SGWT</i>
-----------	-------------------------------

---

**Description**

Demonstration function showing basic SGWT usage with synthetic data

**Usage**

```
demo_sgwt()
```

**Value**

List containing demo data, SGWT results, and energy analysis

**Examples**

```
## Not run:
demo_result <- demo_sgwt()
print(demo_result$energy)

## End(Not run)
```

---

FastDecompositionLap

*Fast eigendecomposition of Laplacian matrix*


---

## Description

Perform fast eigendecomposition using RSpectra for large matrices

## Usage

```
FastDecompositionLap(
  laplacianMat = NULL,
  k_fold = 1.5,
  which = "LM",
  sigma = NULL,
  opts = list(),
  lower = TRUE,
  ...
)
```

## Arguments

laplacianMat	Laplacian matrix
k_fold	Multiplier for number of eigenvalues to compute (default: 1.5)
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_fold = 2)

## End(Not run)
```

---

<code>find_knee_point</code>	<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**

<code>y</code>	Numeric vector of y values
<code>sensitivity</code>	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)
```

---

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

---

**Description**

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

**Usage**

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

**Arguments**

<code>signal</code>	Input signal (vector or matrix)
<code>U</code>	Matrix of eigenvectors (dense matrix preferred)

**Value**

Transformed signal in the spectral domain (vector or matrix)

---

`hello_sgw`*Hello function for SGWT package demonstration*

---

**Description**

Simple hello function to demonstrate package loading

**Usage**

```
hello_sgw()
```

**Value**

Character string with greeting

**Examples**

```
hello_sgw()
```

---

`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	<i>Plot frequency modes</i>
---------	-----------------------------

---

**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	<i>Plot SGWT decomposition results</i>
-------------------------	--

---

**Description**

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

**Usage**

```
plot_sgwt_decomposition(sgwt_result, data.in, plot_scales = NULL, ncol = 3)
```

**Arguments**

sgwt_result	SGWT result object from SGWT() function
data.in	Original data frame with spatial coordinates
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 results
plots <- plot_sgwt_decomposition(sgwt_result, data.in)
print(plots)

## End(Not run)
```

---

SGWT

*Spectral Graph Wavelet Transform (SGWT)*


---

**Description**

Main function for performing Spectral Graph Wavelet Transform analysis on spatial data. Provides a comprehensive interface for multi-scale analysis of spatial signals using graph wavelets.

**Usage**

```
SGWT (
  data.in = NULL,
  signal = NULL,
  k = 25,
  scales = NULL,
  J = 5,
  scaling_factor = 2,
  kernel_type = "mexican_hat",
  laplacian_type = "normalized",
  k_fold = 15,
  return_all = TRUE
)
```

**Arguments**

<code>data.in</code>	Data frame containing spatial coordinates and signal data. Must contain columns 'x' and 'y' for spatial coordinates.
<code>signal</code>	Character string specifying the column name of the signal to analyze, or a numeric vector of signal values.
<code>k</code>	Number of nearest neighbors for graph construction (default: 25)
<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") that defines both scaling and wavelet filters

laplacian_type	Type of graph Laplacian ("unnormalized", "normalized", or "randomwalk", default: "normalized")
k_fold	Parameter for eigendecomposition (default: 15)
return_all	Whether to return all analysis results (default: TRUE)

### Value

If return\_all = TRUE, returns a list containing:

**decomposition** SGWT decomposition results

**reconstructed\_signal** Reconstructed signal for validation

**reconstruction\_error** RMSE between original and reconstructed signal

**original\_signal** Original input signal

**graph\_info** Graph construction information (adjacency matrix, Laplacian, eigenvalues, eigenvectors)

**data** Original input data

**parameters** Analysis parameters used

If return\_all = FALSE, returns only the SGWT decomposition.

### 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:
# Generate synthetic spatial data
set.seed(123)
n_points <- 100
x_coords <- rep(1:10, each = 10) + rnorm(n_points, 0, 0.1)
y_coords <- rep(1:10, times = 10) + rnorm(n_points, 0, 0.1)
signal_data <- sin(0.5 * x_coords) * cos(0.3 * y_coords) + rnorm(n_points, 0, 0.1)

demo_data <- data.frame(x = x_coords, y = y_coords, signal = signal_data)

# Apply SGWT
result <- SGWT(data.in = demo_data, signal = "signal", k = 8, J = 4)

# View reconstruction error
print(result$reconstruction_error)

## 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**

<code>lmax</code>	Maximum eigenvalue
<code>J</code>	Number of scales
<code>scaling_factor</code>	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(sgwt_result)
```

**Arguments**

`sgwt_result`      SGWT result object from SGWT() function

**Value**

Data frame with energy analysis results

**Examples**

```
## Not run:
# Assuming you have SGWT results
energy_analysis <- sgwt_energy_analysis(sgwt_result)
print(energy_analysis)

## End(Not run)
```

sgwt\_forward

*Forward SGWT transform***Description**

Decompose signal into wavelet coefficients using SGWT

**Usage**

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

**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, filters, scales, eigenvalues, and 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 = "meyer")
result_heat <- sgwt_forward(signal, eigenvectors, eigenvalues, scales, kernel_type = "heat")

## 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 = "mexican_hat")
```

**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</i>
--------------	-------------------------------

---

**Description**

Reconstruct signal from wavelet coefficients

**Usage**

```
sgwt_inverse(sgwt_decomp)
```

**Arguments**

sgwt\_decomp    SGWT decomposition object from sgwt\_forward

**Value**

Reconstructed signal vector

**Examples**

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

## 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 = "mexican_hat",
  lmax = NULL,
  eigenvalue_range = NULL,
  resolution = 1000
)
```

## Arguments

<code>eigenvalues</code>	Vector of eigenvalues from graph Laplacian
<code>scales</code>	Vector of scales for the wavelets (if NULL, auto-generated)
<code>J</code>	Number of scales to generate if scales is NULL (default: 4)
<code>scaling_factor</code>	Scaling factor between consecutive scales (default: 2)
<code>kernel_type</code>	Type of wavelet kernel ("mexican_hat" or "meyer", default: "mexican_hat")
<code>lmax</code>	Maximum eigenvalue (optional, computed if NULL)
<code>eigenvalue_range</code>	Range of eigenvalues to plot (default: full range)
<code>resolution</code>	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 = "mexican_hat"  
  )  
  print(viz_result$plot)  
  
## End(Not run)
```

# Index

- \* **CODEX**
  - codex\_toy\_data, [6](#)
- \* **SGWT**
  - codex\_toy\_data, [6](#)
- \* **biological-data**
  - BioGSP-package, [2](#)
- \* **datasets**
  - codex\_toy\_data, [6](#)
- \* **graph-theory**
  - BioGSP-package, [2](#)
- \* **internal**
  - sgwt-globals, [16](#)
- \* **package**
  - BioGSP-package, [2](#)
- \* **spatial-analysis**
  - BioGSP-package, [2](#)
- \* **spatial**
  - codex\_toy\_data, [6](#)
- \* **wavelets**
  - BioGSP-package, [2](#)
- \_PACKAGE (BioGSP-package), 2*
- BioGSP-package, [2](#)
- Cal\_Eigen, [4](#)
- Cal\_GCC, [2, 4](#)
- cal\_laplacian, [5](#)
- codex\_toy\_data, [6](#)
- compare\_kernel\_families, [8](#)
- compute\_sgwt\_filters, [9](#)
- cosine\_similarity, [10](#)
- demo\_sgwt, [2, 10](#)
- FastDecompositionLap, [11](#)
- find\_knee\_point, [12](#)
- gft, [12](#)
- hello\_sgwt, [13](#)
- install\_and\_load, [13](#)
- plot\_FM, [14](#)
- plot\_sgwt\_decomposition, [2, 14](#)

- SGWT, [2, 15](#)
- sgwt-globals, [16](#)
- sgwt\_auto\_scales, [17](#)
- sgwt\_energy\_analysis, [2, 17](#)
- sgwt\_forward, [2, 18](#)
- sgwt\_get\_kernels, [19](#)
- sgwt\_inverse, [2, 19](#)
- visualize\_sgwt\_kernels, [20](#)