**HSI-Processing**

Константин Пухкий

# HSI Processing documentation

Добро пожаловать в документацию по обработке ГСИ!



Содержание:

[Анализ сигнатур 3](#_6c7d2b4a8969bae08dac20ce20f09a02)

[Функции для анализа: 3](#_d4909823c74dbcf26a9787b5c69b3c9d)

[get\_centroids\_and\_medoids() 3](#_1abea44ee3ea3c20eee6d6b2836c5240)

[get\_cross\_correlation\_matrix() 4](#_e23b18a4906c1020ec32b8c0673b861f)

[Кластеризация гиперспектральных данных 5](#_7f94f983b61999748812bce1c4899e57)

[Методы кластеризации: 5](#_bb285c6040be8ba9b131d735c51918ae)

[CosClust 5](#_423f5ce1455a9ab1cbe9f0e545a0a82d)

[HDBSCAN 6](#_1d74e8e33fbd8e36ff91d7c9fe60d394)

[KMeans 8](#_59bb4e7c74801a00531f58d7903d93f4)

[SCH 9](#_4fdac6a1f7075331b59a79143096b71f)

[Визуализация данных 11](#_d8f4f0ac4c5885b24f7d1d4662984f3d)

[Методы визуализации данных 11](#_19059d940363d6e774eebe0765ac67c4)

[show\_correlation\_matrix() 11](#_a10476116649286cd76979a7eb8d393a)

[show\_curves() 12](#_1012daddc9c5f3c17b4d794289afd953)

[show\_image() 13](#_4e52acd4786a740159c06caa866dfe18)

[Методы обработки гиперспектральных данных 14](#_70d3bf93886ef50d5f3c6853f91bae16)

[Методы обработки: 14](#_2f2ceed68632fa9c356a17f9678a76b0)

[normalize() 14](#_cb386b59ae47c7a80774c5fbe8bcee44)

[rayleigh\_scattering() 15](#_97dc421cfdfc245c4b74628401c899e5)

[sigma\_maximum\_filter() 15](#_4677bdaea19204070f7095ed22618a23)

[Функции для чтения гиперспектральных данных 16](#_0c24d9c6519995c2b1a3347b5f6183b9)

[Функции для чтения: 16](#_bbda4f7e042a171ef7ca537202e127e2)

[open\_AVIRIS() 16](#_5318c0050d7fa92fedb69b3de5305f1e)

[open\_ENVI() 17](#_bc205aff12278a750929008f8c3cd706)

[open\_ERDAS() 17](#_9df7df39cf8e59e4c9348b4439541ad2)

[open\_SpyFile() 17](#_d011059ab784b30dd2b9efba1f60b5b8)

[open\_TIF() 18](#_b9793d781dfdfc5082e9e480cf2267dd)

[Сентез цветных изображений 18](#_6a05ace1161008304031845629f5f5ec)

[Методы для синтеза цветных изображений: 18](#_95703a84eeb4c30d4f31240acc605668)

[hsi\_synthesize\_rgb() 18](#_5caa67a6d5409b442dc9d4443ea99aaf)

[simple\_synthesize\_rgb() 19](#_288617f63f11a8ac11bc461f1f614d44)

[labels\_to\_rgb() 20](#_915760189d4def6e1fbb4afc161b159b)

[Класс с предопределенным набором цветов: 21](#_1fab9f3a13e34ea58a75f04edcd1d1b7)

[Color 21](#_eaa74bf6e8ec889951e27070db177ae0)

[Разложение на эмпирические моды (EMD) 22](#_942dcf191fd286491b2d81eb49215480)

[Methods for decomposition into EMD 23](#_dbc7e03c86827bd1891254f8f788ec6d)

[SWEMD() 23](#_2120079f72df660b2db6dad4b0e92fbf)

[SWEMD\_signal() 24](#_cad30461ba5030c791e57c04dc72546e)

## Анализ сигнатур

Данный набор функций предоставляет основные инструменты для анализа и обработки данных в контекстах кластеризации и корреляционного анализа. Эти утилиты включают операции для вычисления центроидов и медиан, создания матриц кросс-корреляции и дополнительные методы, которые помогают в понимании структуры и взаимосвязей внутри наборов данных.

### Функции для анализа:

|  |
| --- |
| **hsip.analysis.analysis.get\_centroids\_and\_medoids**(*labels*, *data*, *metric='cosine'*) |
| Вычисление центроидов и медианов для кластеров в размеченных данных.  Функция вычисляет центроиды и медианы для каждого кластера в предоставленных данных, учитывая метки кластеров. Центроиды — это среднее значение всех точек в кластере, а медианы определяются как точка кластера, наиболее близкая к центроиду, в соответствии с заданной метрикой расстояния. Параметры labels : np.ndarray  1D массив меток кластеров для точек данных. Точки с одинаковой меткой считаются принадлежащими одному и тому же кластеру.  data : np.ndarray  2D массив формы (n\_samples, n\_features), содержащий точки данных.  metric : str, optional  Метрика расстояния, используемая для вычисления медианов. По умолчанию „cosine“. Другие допустимые метрики включают „euclidean“, „manhattan“ и другие, поддерживаемые функцией scipy.spatial.distance.cdist. Возвращаемые значения centroids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий медианы кластеров. Примеры Вычисление центроидов и медианов для косинусного сходства  **>>> import** **numpy** **as** **np** **>>> from** **hsip.analysis.analysis** **import** get\_centroids\_and\_medoids **>>>** labels = np.array([0, 0, 1, 1, 2, 2]) **>>>** data = np.array([ **...**  [1, 2], [2, 3], **...**  [3, 4], [4, 5], **...**  [5, 6], [6, 7] **...** ]) **>>>** centroids, medoids = get\_centroids\_and\_medoids(labels, data, metric='cosine') **>>>** print("Центроиды:") **>>>** print(centroids) Центроиды: [[1.5 2.5]  [3.5 4.5]  [5.5 6.5]] **>>>** print("Медианы:") **>>>** print(medoids) Медианы: [[1. 2.]  [3. 4.]  [5. 6.]] |

|  |
| --- |
| **hsip.analysis.analysis.get\_cross\_correlation\_matrix**(*data: ndarray*, *metric: str = 'euclidean'*) |
| Вычисление матрицы перекрёстной корреляции для заданного набора данных.  Эта функция вычисляет попарные расстояния между всеми строками входных данных с использованием указанной метрики расстояния и возвращает полученную матрицу перекрёстной корреляции. Параметры data : np.ndarray  2D массив формы (n\_samples, n\_features), содержащий входные данные, где n\_samples — количество точек данных, а n\_features — количество признаков.  metric : str, optional  Метрика расстояния, используемая для вычисления попарных расстояний. По умолчанию „euclidean“. Поддерживаемые метрики включают „euclidean“, „manhattan“, „cosine“ и другие, доступные в scipy.spatial.distance.cdist. Возвращаемые значения cross\_corr\_mat : np.ndarray  2D массив формы (n\_samples, n\_samples), содержащий попарные расстояния между всеми точками данных во входных данных. Примеры Вычисление матрицы перекрёстной корреляции с метрикой Евклида по умолчанию  **>>> import** **numpy** **as** **np** **>>> from** **hsip.analysis.analysis** **import** get\_cross\_correlation\_matrix **>>>** data = np.array([[0, 1], [1, 2], [2, 3], [3, 4]]) **>>>** cross\_corr\_mat = get\_cross\_correlation\_matrix(data) **>>>** print(cross\_corr\_mat) [[0. 1.41421356 2.82842712 4.24264069]  [1.41421356 0. 1.41421356 2.82842712]  [2.82842712 1.41421356 0. 1.41421356]  [4.24264069 2.82842712 1.41421356 0. ]] |

## Кластеризация гиперспектральных данных

Одним из основных методов для обработки ГСИ является кластеризация.

### Методы кластеризации:

|  |
| --- |
| *class* **hsip.clustering.clustering.CosClust**(*threshold: float = 0.9*, *verbose=True*) |
| Базовые классы: object  Кластеризационный алгоритм на основе косинусного сходства. Объединяет образцы в кластеры, основываясь на пороговом значении косинусного сходства, и назначает метки каждому образцу. Параметры threshold : float, optional  Порог косинусного сходства для определения принадлежности к кластеру. По умолчанию 0.9.  verbose : bool, optional  Если установлено в True, отображает процесс и дополнительную информацию во время кластеризации. По умолчанию True. Атрибуты threshold : float  Порог косинусного сходства для кластеризации.  labels : np.ndarray or None  Метки кластеров, назначенные каждому образцу. Инициализируется как None, заполняется после вызова метода fit.  reference\_set : list  Список эталонных образцов, представляющих каждый кластер.  centroids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий медианы кластеров. Методы fit(source\_data)  Выполняет кластеризацию входных данных и возвращает метки кластеров. Примеры **>>> import** **numpy** **as** **np** **>>> from** **hsip.clustering.clustering** **import** CosClust **>>>** data = np.random.rand(100, 50) *# 100 образцов, каждый с 50 признаками* **>>>** model = CosClust(threshold=0.8, verbose=**True**) **>>>** labels = model.fit(data) **>>>** print(labels) array([0, 1, 0, 2, ..., 1])   |  | | --- | | **fit**(*source\_data: ndarray*) | | Выполняет кластеризацию заданных данных на основе косинусного сходства. Параметры source\_data : np.ndarray  2D массив формы (n\_samples, n\_features), где n\_samples — количество образцов, а n\_features — количество признаков каждого образца. Возвращаемые значения np.ndarray  1D массив формы (n\_samples,), содержащий метки кластеров для каждого образца. Заметки Алгоритм выполняется в два этапа:   1. Начальная кластеризация на основе порогового значения косинусного сходства. 2. Корректировка меток на основе эталонного набора представителей кластеров.  * Образцы, которым не удалось назначить кластер, маркируются как -1. | |

|  |
| --- |
| *class* **hsip.clustering.clustering.HDBSCAN**(*\*\*kwargs*) |
| Базовые классы: object  Обёртка для алгоритма кластеризации HDBSCAN.  Класс HDBSCAN упрощает использование алгоритма кластеризации HDBSCAN, инкапсулируя основную функциональность класса hdbscan.HDBSCAN. Он предоставляет удобный интерфейс для кластеризации данных и получения меток кластеров. Атрибуты labels : np.ndarray или None  Метки кластеров, назначенные каждой точке данных после обучения модели. Изначально установлено в None.  centroids : np.ndarray  Двумерный массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  Двумерный массив формы (n\_clusters, n\_features), содержащий медианы кластеров. Методы fit(source\_data: np.ndarray) -> np.ndarray  Обучает модель кластеризации HDBSCAN на предоставленных данных и вычисляет метки кластеров. Параметры min\_cluster\_size : int, по умолчанию=5  Минимальный размер кластеров.  min\_samples : int, необязательный  Минимальное количество точек в окружении, чтобы точка считалась ядром кластера.  cluster\_selection\_epsilon : float, по умолчанию=0.0  Пороговое расстояние для выбора кластера. Примеры **>>> import** **numpy** **as** **np** **>>>** source\_data = np.random.rand(100, 5) *# Пример данных* **>>>** model = HDBSCAN(min\_cluster\_size=10) **>>>** labels = model.fit(source\_data) **>>>** print(labels) [0 0 1 -1 2 2 1 1 3 ...]   |  | | --- | | **fit**(*source\_data: ndarray*) | | Выполняет кластеризацию методом HDBSCAN на входных данных. Параметры source\_data : np.ndarray  Массив данных для кластеризации. Размерность массива (n\_samples, n\_features), где n\_samples — количество объектов, n\_features — количество признаков. Возвращаемое значение np.ndarray  Массив меток кластеров для каждого объекта в данных. Размерность: (n\_samples,). | |

|  |
| --- |
| *class* **hsip.clustering.clustering.KMeans**(*centroids: list | ndarray | None = None*, *n\_clusters: int | None = None*, *metric: str = 'euclidean'*, *verbose: bool = True*, *max\_iter: int = 300*) |
| Базовые классы: object  Класс KMeans  Класс для кластеризации данных методом K-средних с возможностью задания начальных центроидов, настройки количества кластеров, выбора метрики и ограничения числа итераций. Атрибуты labels : np.ndarray  Метки кластеров для каждого объекта в данных после выполнения алгоритма.  centroids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий медианы кластеров.  n\_clusters : int  Количество указанных кластеров.  metric : str  Метрика расстояния, используемая для расчёта расстояний. Доступные значения: „euclidean“, „cosine“.  verbose : bool  Флаг вывода прогресса выполнения алгоритма. По умолчанию True.  max\_iter : int  Максимальное количество итераций алгоритма. По умолчанию 300. Методы fit(source\_data)  Выполняет кластеризацию методом K-средних на входных данных. Параметры centroids : list | np.ndarray, optional  Начальные центроиды. Если не указаны, центроиды инициализируются случайным выбором из данных.  n\_clusters : int, optional  Количество кластеров. Требуется, если не заданы центроиды.  metric : str, optional  Метрика для расчёта расстояний между точками и центроидами. Поддерживаются „euclidean“ и „cosine“. По умолчанию „euclidean“.  verbose : bool, optional  Вывод информации о ходе выполнения. По умолчанию True.  max\_iter : int, optional  Максимальное число итераций. По умолчанию 300. Пример Кластеризация данных с 3 кластерами:  **>>> import** **numpy** **as** **np** **>>>** data = np.random.rand(100, 2) *# Случайные точки* **>>>** kmeans = KMeans(n\_clusters=3) **>>>** labels = kmeans.fit(data)  Задание начальных центроидов:  **>>>** initial\_centroids = [[0.1, 0.1], [0.5, 0.5], [0.9, 0.9]] **>>>** kmeans = KMeans(centroids=initial\_centroids) **>>>** labels = kmeans.fit(data)  Вывод центроидов:  **>>>** print(kmeans.centroids)   |  | | --- | | **fit**(*source\_data: ndarray*) | | Выполняет кластеризацию методом K-средних на входных данных. Параметры source\_data : np.ndarray  Массив данных для кластеризации. Размерность массива (n\_samples, n\_features), где n\_samples — количество объектов, n\_features — количество признаков. Возвращаемое значение np.ndarray  Массив меток кластеров для каждого объекта в данных. Размерность: (n\_samples,). | |

|  |
| --- |
| *class* **hsip.clustering.clustering.SCH**(*linkage\_method: str = 'complete'*, *linkage\_metric: str = 'cosine'*, *linkage\_optimal\_ordering: bool = False*, *fcluster\_t: float = 0.25*, *fcluster\_criterion: str = 'distance'*, *fcluster\_depth: int = 2*) |
| Базовые классы: object  Класс для выполнения иерархической кластеризации с использованием методов linkage и fcluster из библиотеки SciPy.  Класс SCH предоставляет интерфейс для иерархической кластеризации с настраиваемыми параметрами объединения и кластеризации. Атрибуты labels : np.ndarray or None  Метки кластеров, присвоенные каждой точке данных после обучения модели. Изначально установлено в None.  linkage\_method : str  Метод объединения, используемый для иерархической кластеризации. Поддерживаемые методы включают «single», «complete», «average», «weighted», «centroid», «median», и «ward».  linkage\_metric : str  Метрика расстояния, используемая для вычисления попарных расстояний между точками данных. Популярные метрики: «euclidean», «cosine», «cityblock», «hamming».  linkage\_optimal\_ordering : bool  Если True, матрица объединений будет упорядочена для минимизации расстояний между последовательными листьями.  fcluster\_t : float  Порог для формирования плоских кластеров. Значение t зависит от fcluster\_criterion.  fcluster\_criterion : str  Критерий для формирования плоских кластеров. Поддерживаемые критерии: «inconsistent», «distance», «maxclust».  fcluster\_depth : int  Максимальная глубина для расчета несогласованности, если fcluster\_criterion=»inconsistent». Игнорируется для других критериев.  centroids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий медианы кластеров. Методы fit(source\_data: np.ndarray) -> np.ndarray  Обучает модель иерархической кластеризации на предоставленных данных и вычисляет метки кластеров. Параметры linkage\_method : str, optional  Метод объединения для кластеризации. По умолчанию «complete».  linkage\_metric : str, optional  Метрика расстояния для кластеризации. По умолчанию «cosine».  linkage\_optimal\_ordering : bool, optional  Перестраивать ли матрицу объединений для оптимального порядка листьев. По умолчанию False.  fcluster\_t : float, optional  Порог для формирования плоских кластеров. По умолчанию 0.25.  fcluster\_criterion : str, optional  Критерий для формирования плоских кластеров. По умолчанию «distance».  fcluster\_depth : int, optional  Глубина для расчета несогласованности, если fcluster\_criterion=»inconsistent». По умолчанию 2. Примеры **>>> import** **numpy** **as** **np** **>>> from** **hsip.clustering.clustering** **import** SCH **>>>** source\_data = np.random.rand(10, 5) **>>>** model = SCH() **>>>** labels = model.fit(source\_data) **>>>** print(labels) [1 1 2 2 3 3 4 4 5 5]   |  | | --- | | **fit**(*source\_data: ndarray*) | | Выполняет кластеризацию методом иерархичексой кластеризации на входных данных. Параметры source\_data : np.ndarray  Массив данных для кластеризации. Размерность массива (n\_samples, n\_features), где n\_samples — количество объектов, n\_features — количество признаков. Возвращаемое значение np.ndarray  Массив меток кластеров для каждого объекта в данных. Размерность: (n\_samples,). | |

## Визуализация данных

Для правильного анализа промежуточных и конечных результатов, особенно при работе с большими массивами данных, удобнее всего рассматривать их в графической форме. Для этого предоставлен функционал, использующий модуль matplotlib, с готовой надстройкой, которая упрощает использование и облегчает интерпретацию входных данных.

### Методы визуализации данных

|  |
| --- |
| **hsip.graphics.graphics.show\_correlation\_matrix**(*corr\_matrix: ndarray*, *label\_axis: list*, *path: str | None = None*) |
| Отображение матрицы корреляции с аннотированными значениями.  Эта функция визуализирует матрицу корреляции с аннотированными значениями, где каждый элемент матрицы отображается на соответствующей позиции с точностью до двух знаков после запятой. Параметры corr\_matrix : np.ndarray  Квадратная матрица корреляции, которая будет отображена в виде тепловой карты.  label\_axis : list  Список меток для осей X и Y, который используется для подписей в графике.  path : str, optional  Путь, по которому сохранить изображение. Если параметр не указан (по умолчанию None), то изображение будет показано в окне. Возвращает None Пример **>>> import** **numpy** **as** **np** **>>>** corr\_matrix = np.array([[1, 0.8], [0.8, 1]]) **>>>** label\_axis = ['A', 'B'] **>>>** show\_correlation\_matrix(corr\_matrix, label\_axis) |

|  |
| --- |
| **hsip.graphics.graphics.show\_curves**(*curves: ndarray*, *labels: list | ndarray | None = None*, *colors: list | ndarray | None = None*, *xlabel: str = 'Номер канала'*, *ylabel: str = 'Величина'*, *title: str | None = None*, *path: str | None = None*, *scale\_img: float = 4.0*) |
| Отображение набора кривых с возможностью настройки меток, цветов и сохранения.  Функция визуализирует набор кривых, где каждая строка массива curves соответствует одной кривой. Можно задавать метки для легенды, цвета кривых и параметры отображения. Параметры curves : np.ndarray  Двумерный массив размером (n\_curves, n\_points), где каждая строка представляет одну кривую. Если передан одномерный массив, он автоматически преобразуется в массив с одной строкой.  labels : list | np.ndarray, optional  Метки для каждой кривой. Если не указаны (по умолчанию None), кривые будут отображены без легенды.  colors : list | np.ndarray, optional  Цвета для кривых. Если не указаны (по умолчанию None), используются заранее заданные цвета. Количество цветов должно совпадать с количеством кривых.  xlabel : str, optional  Метка оси X (по умолчанию „Номер канала“).  ylabel : str, optional  Метка оси Y (по умолчанию „Величина“).  title : str, optional  Заголовок графика (по умолчанию None).  path : str, optional  Путь для сохранения изображения. Если не указан (по умолчанию None), изображение отображается в окне.  scale\_img : float, optional  Коэффициент масштабирования размера изображения (по умолчанию 4.0). Возвращает None Пример Отображение трёх кривых с разными метками и цветами:  **>>> import** **numpy** **as** **np** **>>>** curves = np.random.rand(3, 100) **>>>** labels = ['Кривая 1', 'Кривая 2', 'Кривая 3'] **>>>** colors = [[1, 0, 0], [0, 1, 0], [0, 0, 1]] **>>>** show\_curves(curves, labels=labels, colors=colors, title='Пример графика')  Сохранение графика в файл:  **>>>** show\_curves(curves, path='curves\_plot.png') |

|  |
| --- |
| **hsip.graphics.graphics.show\_image**(*image: ndarray*, *path: str | None = None*, *color\_bar: bool = False*, *scale\_img: float = 1.0*) |
| Отображение изображения с возможностью сохранения и масштабирования.  Функция визуализирует переданное изображение. Поддерживается отображение цветных RGB изображений и одноканальных изображений с настройкой цветовой шкалы. Также можно сохранить изображение в файл, указав путь в параметре path. Параметры image : np.ndarray  Массив изображения. Для RGB изображения тип данных должен быть uint8, а размерность массива — (height, width, 3). Для одноканального изображения допускаются другие типы данных и размерность (height, width).  path : str, optional  Путь для сохранения изображения. Если не указан (по умолчанию None), изображение будет отображено в окне.  color\_bar : bool, optional  Если True, для одноканальных изображений добавляется цветовая шкала (по умолчанию False). Для RGB изображений этот параметр игнорируется.  scale\_img : float, optional  Коэффициент масштабирования изображения. Например, при значении 2.0 изображение будет отображено в два раза крупнее (по умолчанию 1.0). Возвращает None Пример Отображение и сохранение RGB изображения:  **>>> import** **numpy** **as** **np** **>>>** rgb\_image = np.random.randint(0, 255, (100, 100, 3), dtype=np.uint8) **>>>** show\_image(rgb\_image, path='rgb\_image.png', scale\_img=1.5)  Отображение одноканального изображения с цветовой шкалой:  **>>>** grayscale\_image = np.random.rand(100, 100) **>>>** show\_image(grayscale\_image, color\_bar=**True**) |

## Методы обработки гиперспектральных данных

Работа с HSI часто требует наличия функциональных возможностей как для предварительной, так и для последующей обработки данных.

### Методы обработки:

|  |
| --- |
| **hsip.processing.processing.normalize**(*array: ndarray*) |
| Оболочка для алгоритма кластеризации HDBSCAN.  Класс HDBSCAN упрощает использование алгоритма HDBSCAN, инкапсулируя основную функциональность класса hdbscan.HDBSCAN. Он предоставляет удобный интерфейс для кластеризации данных и получения меток кластеров. Атрибуты labels : np.ndarray or None  Метки кластеров, присвоенные каждой точке данных после обучения модели. Изначально установлено в None.  centroids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий центроиды кластеров.  medoids : np.ndarray  2D массив формы (n\_clusters, n\_features), содержащий медианы кластеров. Методы fit(source\_data: np.ndarray) -> np.ndarray  Обучает модель HDBSCAN на предоставленных данных и вычисляет метки кластеров. Параметры min\_cluster\_size : int, optional  Минимальный размер кластеров. По умолчанию 5.  min\_samples : int, optional  Минимальное количество точек в окрестности, чтобы точка считалась ядровой.  cluster\_selection\_epsilon : float, optional  Порог расстояния для выбора кластеров. По умолчанию 0.0. Примеры **>>> import** **numpy** **as** **np** **>>> from** **hsip.clustering.clustering** **import** HDBSCAN **>>>** source\_data = np.random.rand(100, 5) **>>>** model = HDBSCAN(min\_cluster\_size=10) **>>>** labels = model.fit(source\_data) **>>>** print(labels) [0 0 1 -1 2 2 1 1 3 ...] |

|  |
| --- |
| **hsip.processing.processing.rayleigh\_scattering**(*spectral\_data: ndarray*, *inplace=False*, *verbose=True*) |
| Вычисление сигнатуры рассеяния Рэлея для спектрального набора данных. Параметры spectral\_data : np.ndarray  Многомерный массив NumPy, где последняя размерность представляет спектральные диапазоны.  inplace : bool, default=False  Если True, функция изменяет spectral\_data, вычитая вычисленную сигнатуру Рэлея.  verbose : bool, verbose=True  Если True, отображает индикатор прогресса. Возвращаемые значения np.ndarray  1D массив, содержащий сигнатуру рассеяния Рэлея для каждого спектрального диапазона. Примеры Применение фильтра Рэлея к 3D спектральному набору данных:  **>>> import** **numpy** **as** **np** **>>> from** **hsip.processing.processing** **import** rayleigh\_scattering **>>>** data = np.random.rand(100, 100, 10) \* 10 *# Пример спектральных данных* **>>>** rayleigh\_signature = rayleigh\_scattering(data, inplace=**True**) **>>>** rayleigh\_signature *# Сигнатура рассеяния Рэлея* array([...]) |

|  |
| --- |
| **hsip.processing.processing.sigma\_maximum\_filter**(*spectral\_data: ndarray*, *sigma: float = 3*, *thresholds: ndarray | None = None*) |
| Применяет сигма-ориентированный максимальный фильтр к входным спектральным данным, ограничивая значения на основе вычисленного порога: среднее значение + сигма \* стандартное отклонение. Параметры spectral\_data : np.ndarray  Входной массив спектральных данных. Ожидается, что последняя размерность соответствует спектральным диапазонам.  sigma : float, optional  Множитель для стандартного отклонения, используемый при расчете порога. По умолчанию 3.  thresholds : np.ndarray, optional  Массив для хранения вычисленных порогов. Если передан, его форма должна совпадать с последней размерностью spectral\_data. Если None, пороги вычисляются и возвращаются внутренне. По умолчанию None. Возвращаемые значения np.ndarray  Отфильтрованные спектральные данные с ограниченными значениями на основе вычисленных порогов. Примеры Применение сигма-фильтра к 3D спектральному набору данных:  **>>> import** **numpy** **as** **np** **>>> from** **hsip.processing.processing** **import** sigma\_maximum\_filter **>>>** data = np.random.rand(100, 100, 10) \* 10 *# Пример спектральных данных* **>>>** thresholds = np.zeros(data.shape[-1:], dtype=np.float32) **>>>** result\_with\_thresholds = sigma\_maximum\_filter(data, sigma=2, thresholds=thresholds) **>>>** thresholds *# Обновленные пороги* array([...]) |

## Функции для чтения гиперспектральных данных

Гиперспектральные данные распространяются в разных форматах. Используя эти функции, можно читать HSI в следующих форматах: SpyFile, ENVI, AVIRIS, ERDAS/Lan, Tif.

Все функции чтения, кроме чтения в формате TIF, являются обертками для функций из [библиотеки Spectral Python](https://www.spectralpython.net/).

### Функции для чтения:

|  |
| --- |
| **hsip.reader.reader.open\_AVIRIS**(*path\_rfl: str*, *path\_spc: str*) |
| Открывает гиперспектральное изображение в формате AVIRIS. Параметры path\_rfl : str  Путь к файлу данных отражения AVIRIS (.rfl).  path\_spc : str  Путь к файлу спектральной калибровки AVIRIS (.spc). Возвращаемые значения SpectralLibrary  Объект SpyFile, содержащий гиперспектральные данные AVIRIS. Примеры **>>> from** **hsip.reader.reader** **import** open\_AVIRIS **>>>** hsi = open\_AVIRIS("example.rfl", "example.spc") **>>>** print(hsi) SpyFile: [shape=(100, 100, 224), dtype=float32] |

|  |
| --- |
| **hsip.reader.reader.open\_ENVI**(*path\_hdr: str*, *path\_img: str*) |
| Открывает гиперспектральное изображение в формате ENVI и конвертирует его в массив NumPy. Параметры path\_hdr : str  Путь к заголовочному файлу ENVI (.hdr).  path\_img : str  Путь к изображению ENVI (.img). Возвращаемые значения np.ndarray  Массив NumPy с гиперспектральными данными, с типом данных dtype=float. Примеры **>>> from** **hsip.reader.reader** **import** open\_ENVI **>>>** hsi = open\_ENVI("example.hdr", "example.img") **>>>** print(hsi.shape) (100, 100, 224) # Примерные размеры |

|  |
| --- |
| **hsip.reader.reader.open\_ERDAS**(*path: str*) |
| Открывает гиперспектральное изображение в формате ERDAS Imagine. Параметры path : str  Путь к файлу ERDAS Imagine, который необходимо открыть. Возвращаемые значения SpectralLibrary  Объект SpyFile, содержащий гиперспектральные данные. Примеры **>>> from** **hsip.reader.reader** **import** open\_ERDAS **>>>** hsi = open\_ERDAS("example.img") **>>>** print(hsi) SpyFile: [shape=(100, 100, 224), dtype=float32] |

|  |
| --- |
| **hsip.reader.reader.open\_SpyFile**(*path\_lan: str*) |
| Открывает гиперспектральное изображение в формате SpyFile. Параметры path\_lan : str  Путь к LAN-файлу, который необходимо открыть. Возвращаемые значения SpectralLibrary  Объект SpyFile, содержащий гиперспектральные данные. Примеры **>>> from** **hsip.reader.reader** **import** open\_SpyFile **>>>** hsi = open\_SpyFile("example.lan") **>>>** print(hsi) SpyFile: [shape=(100, 100, 224), dtype=float32] |

|  |
| --- |
| **hsip.reader.reader.open\_TIF**(*path\_tif: str*) |
| Открывает гиперспектральное изображение в формате GeoTIFF. Параметры path\_tif : str  Путь к файлу GeoTIFF, который необходимо открыть. Возвращаемые значения np.ndarray  Массив NumPy, содержащий гиперспектральные данные. Примеры **>>> from** **hsip.reader.reader** **import** open\_TIF **>>>** hsi = open\_TIF("example.tif") **>>>** print(hsi.shape) (100, 100, 224) |

## Сентез цветных изображений

Как и гиперспектральные изображения, цветные изображения также состоят из нескольких каналов. Эти функции позволяют синтезировать цветные изображения из гиперспектральных данных, максимально приближенными к реальности. Кроме того, функционал позволяет создавать цветные изображения, состоящие из наиболее информативных каналов, которые можно использовать для семантической кластеризации.

### Методы для синтеза цветных изображений:

|  |
| --- |
| **hsip.rgb.rgb.hsi\_synthesize\_rgb**(*spectral\_data: ndarray*, *rgb\_bands: list | ndarray | None = None*, *wavelengths: list | ndarray | None = None*) |
| Синтезирует RGB-изображение из гиперспектральных данных. Параметры spectral\_data : np.ndarray  Гиперспектральные данные изображения, ожидается, что это 3D массив формы (height, width, bands).  rgb\_bands : list of int, необязательный  Список, содержащий индексы полос для каналов красного, зеленого и синего цветов, в указанном порядке. Должен иметь длину 3. Если указан, wavelengths игнорируется.  wavelengths : list of float, необязательный  Список, содержащий длины волн, соответствующие каждой полосе в spectral\_data. Если указан, выбираются наиболее близкие длины волн к 650 нм (красный), 550 нм (зеленый) и 450 нм (синий). Возвращает np.ndarray  3D массив формы (height, width, 3), представляющий синтезированное RGB-изображение. Примеры Использование индексов полос напрямую:  **>>> from** **hsip.rgb.rgb** **import** hsi\_synthesize\_rgb **>>>** spectral\_data = np.random.rand(100, 100, 224) *# Пример гиперспектральных данных* **>>>** rgb\_bands = [50, 100, 150] *# Пример полос для красного, зеленого, синего* **>>>** rgb\_image = synthesize\_rgb(spectral\_data, rgb\_bands=rgb\_bands) **>>>** print(rgb\_image.shape) (100, 100, 3)  Использование длин волн:  **>>>** wavelengths = np.linspace(400, 700, 224) *# Пример данных о длинах волн* **>>>** rgb\_image = synthesize\_rgb(spectral\_data, wavelengths=wavelengths) **>>>** print(rgb\_image.shape) (100, 100, 3) |

|  |
| --- |
| **hsip.rgb.rgb.simple\_synthesize\_rgb**(*band\_data: list*, *sig\_max\_filt: float | None = None*) |
| Генерирует RGB-изображение из трех гиперспектральных полос.  Функция синтезирует простое RGB-изображение из трех предоставленных полос, соответствующих каналам красного, зеленого и синего. Опционально, она применяет сигма максимум-фильтр к каждой полосе для снижения шума перед нормализацией и масштабированием. Параметры band\_data : list of np.ndarray  Список, содержащий ровно три 2D массива, каждый из которых представляет одну полосу данных для каналов красного, зеленого и синего. Каждый массив должен иметь одинаковую форму.  sig\_max\_filt : float, необязательный  Значение сигма для применения функции sigma\_maximum\_filter к каждой полосе. Если указано, оно используется для снижения шума перед синтезом RGB-изображения. Возвращает rgb\_image : np.ndarray  3D массив формы (height, width, 3), представляющий синтезированное RGB-изображение. Выходной тип — np.uint8 с пиксельными значениями, масштабированными в диапазон [0, 255]. Примеры Генерация RGB-изображения с применением сигма максимум-фильтрации:  **>>> import** **numpy** **as** **np** **>>> from** **hsip.rgb.rgb** **import** simple\_synthesize\_rgb **>>>** band\_red = np.random.rand(100, 100) **>>>** band\_green = np.random.rand(100, 100) **>>>** band\_blue = np.random.rand(100, 100) **>>>** band\_data = [band\_red, band\_green, band\_blue] **>>>** rgb\_image = simple\_synthesize\_rgb(band\_data, sig\_max\_filt=3) **>>>** print(rgb\_image.shape) (100, 100, 3) |

|  |
| --- |
| **hsip.rgb.labels.labels\_to\_rgb**(*labels: ndarray*, *RGB\_image: ndarray | None = None*) |
| Преобразует метки в маске в представление в виде изображения RGB.  Эта функция генерирует RGB-изображение, где каждой уникальной метке из входного массива labels присваивается определенный цвет. Если предоставлено RGB\_image, цвет каждой метки вычисляется как среднее значение цветов для соответствующих областей в RGB\_image. Если изображение не предоставлено, используются заранее определенные цвета из глобального набора colors\_set. Внимание: в наборе colors\_set только 17 цветов, и если уникальных меток больше, то цвета для некоторых из них будут повторяться! Параметры labels : np.ndarray  2D или 3D массив, представляющий маску меток. Каждое уникальное значение соответствует различному классу.  RGB\_image : np.ndarray, необязательный  3D массив формы (height, width, 3), представляющий существующее RGB-изображение. Если предоставлено, цвет для каждой метки вычисляется как среднее значение RGB для соответствующих регионов в этом изображении. Возвращает np.ndarray  RGB-изображение формы (height, width, 3) с присвоенными цветами для каждой метки. Примеры Пример 1: Использование заранее определенного набора цветов:  **>>> from** **hsip.rgb.labels** **import** labels\_to\_rgb **>>>** labels = np.array([[0, 0, 1], **...**  [1, 2, 2]]) **>>>** rgb\_image = labels\_to\_rgb(labels) **>>>** print(rgb\_image.shape) (2, 3, 3)  Пример 2: Использование существующего RGB-изображения для вычисления цветов:  **>>> from** **hsip.rgb.labels** **import** labels\_to\_rgb **>>>** labels = np.array([[0, 0, 1], **...**  [1, 2, 2]]) **>>>** RGB\_image = np.random.randint(0, 255, size=(2, 3, 3), dtype=np.uint8) **>>>** rgb\_image = labels\_to\_rgb(labels, RGB\_image=RGB\_image) **>>>** print(rgb\_image.shape) (2, 3, 3) |

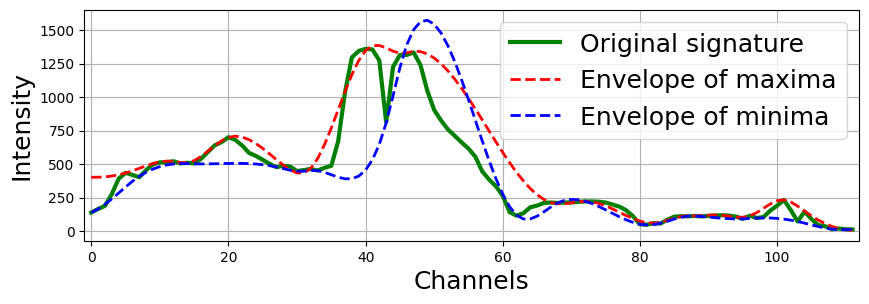
В некоторых случаях, когда требуется выделить различные области, сегменты или графики, важно использовать контрастный набор цветов, который позволяет ясно различать эти элементы и визуально выделить их.

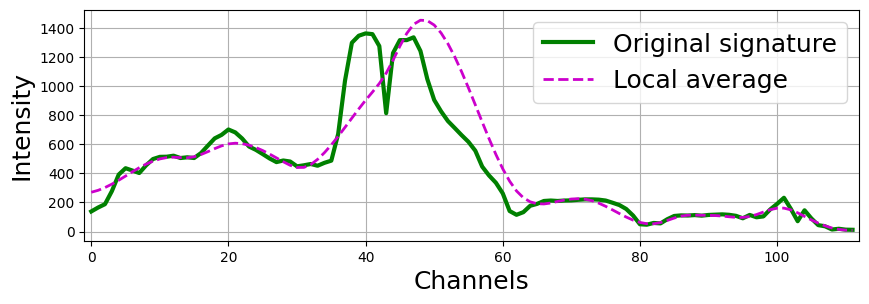
### Класс с предопределенным набором цветов:

|  |
| --- |
| *class* **hsip.rgb.colors.Color** |
| Базовые классы: object  Утилитный класс для управления и доступа к предустановленному набору цветов.  Этот класс предоставляет коллекцию значений цветов в формате RGB в диапазоне [0, 1] и методы для получения одного или нескольких цветов по индексу или срезу. Он также поддерживает модульную индексацию для циклического обращения к набору цветов. Атрибуты colors : np.ndarray  Двумерный массив формы (n\_colors, 3), содержащий значения цветов RGB в диапазоне [0, 1]. Методы \_\_get\_color\_\_(index)  Получает один цвет или подмножество цветов по индексу или срезу.  \_\_getitem\_\_(index)  Псевдоним для метода \_\_get\_color\_\_, позволяет использовать квадратные скобки для индексации.  \_\_len\_\_()  Возвращает общее количество цветов в палитре.  \_\_iter\_\_()  Возвращает итератор по цветам. Параметры Нет Примеры Создание экземпляра класса Color и получение цветов:  **>>> from** **hsip.rgb.colors** **import** Color **>>>** color\_palette = Color()  Получение одного цвета с использованием индекса:  **>>>** color\_palette[0] array([0.90196078, 0.09803922, 0.29411765])  Получение нескольких цветов с использованием среза:  **>>>** color\_palette[1:4] array([[0.23529412, 0.70588235, 0.29411765],  [0. , 0.50980392, 0.78431373],  [0.96078431, 0.50980392, 0.18823529]])  Модульная индексация:  **>>>** color\_palette[20] array([0.23529412, 0.70588235, 0.29411765]) # Индекс 20 соответствует индексу 1 (20 % len(colors))  Итерация по цветам:  **>>> for** color **in** color\_palette: **...**  print(color) array([0.90196078, 0.09803922, 0.29411765]) array([0.23529412, 0.70588235, 0.29411765]) ... |

## Разложение на эмпирические моды (EMD)

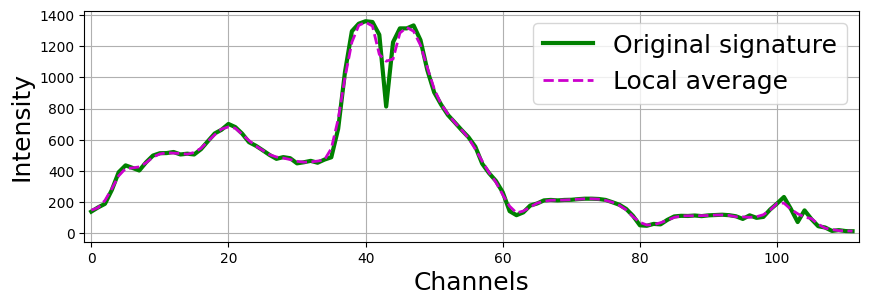
Разложение на эмпирические моды — это метод обработки сигналов, используемый для разложения сигнала на набор собственных мод (IMFs), которые представляют собой осцилляторные компоненты. Метод итеративно выявляет локальные экстремумы, строит верхние и нижние огибающие с использованием сплайн-интерполяции и вычисляет локальное среднее для извлечения IMFs. Существенным недостатком EMD является точность определения локального среднего и огибающих, особенно вблизи границ данных, где выход за границы может привести к искажениям. Эти пограничные проблемы и зависимость от сплайн-интерполяции могут вызвать неточности в извлеченных модах.





Алгоритм разложения SWEMD, адаптированный для анализа гиперспектральных изображений, включает несколько обновлений по сравнению с классическим подходом:

1. Точное вычисление локального среднего с использованием скользящего среднего окна, а не арифметического среднего значений максимума и минимума огибающих сигнала.
2. Введение правила для вычисления локального среднего, когда окно выходит за пределы диапазона спектральных каналов.
3. Гибкость в адаптации размера окна к меньшим масштабам, чем расстояние между соседними нулевыми пересечениями в текущей эмпирической моде (EM) или начальная ширина окна.
4. Подавление шума в эмпирических модах путем увеличения начального размера окна и его многократного применения.



### Methods for decomposition into EMD

|  |
| --- |
| **hsip.swemd.swemd.SWEMD**(*data: ndarray*, *number\_of\_modes: int = 4*, *windows\_size: list = [3]*, *verbose: bool = True*) |
| Рассчитывает и возвращает IMF и окна для каждого каждого сигнала, указанного в data. Параметры x : np.ndarray  Массив размерности 3 (height \* width \* bands), размерности 2 (n\_samples \* bands), или просто один образец.  number\_of\_modes : int, по умолчанию=4  Количество IMF, которые необходимо вычислить для входного сигнала.  windows\_size : list или tuple of int, по умолчанию=3  Размер окон для каждой моды, начиная с первой. Если передан список, то каждый элемент указывает размер окна для соответствующего IMF. Если список переданных размеров меньше, чем указано в number\_of\_modes, то последующие размеры скользящих окон будут вычислены автоматически. Если в списке указан элемент со значением -1, то размер этого окна будет также вычислен автоматически. Если передано целое число, это будет размер скользящего окна только для первого IMF. Возвращает IMFs : list  Эмпирические моды для каждого образца.  err\_windows\_size : list  Размеры окон для каждого уровня эмпирических мод. Примеры Вычисление EMD: длины окон равны трем для 1-ой и 2-ой моды, 5 для 3-ий моды и автоматически для остальных.  **>>> import** **numpy** **as** **np** **>>> from** **hsip.swemd.swemd** **import** SWEMD **>>>** data = np.random.rand(1000, 1000, 100) \* 10 *# Пример спектральных данных* **>>>** IMFs, windows = SWEMD(data, number\_of\_modes=8, windows\_size=[3, 3, 5]) **>>>** print(IMFs.shape, windows.shape) (8, 1000, 1000, 100), (8, 1000, 1000) |

|  |
| --- |
| **hsip.swemd.swemd.SWEMD\_signal**(*iSample: ndarray*, *number\_of\_modes: int = 4*, *windows\_size: list = [3]*) |
| Возвращает IMF для одномерного образца. Параметры x : np.ndarray  Одномерный сигнал.  number\_of\_modes : int, по умолчанию=4  Количество IMF, которые необходимо вычислить для входного сигнала.  windows\_size : list или tuple of int, по умолчанию=3  Размер окон для каждой моды, начиная с первой. Если передан список, то каждый элемент указывает размер окна для соответствующего IMF. Если список переданных размеров меньше, чем указано в number\_of\_modes, то последующие размеры скользящих окон будут вычислены автоматически. Если в списке указан элемент со значением -1, то размер этого окна будет также вычислен автоматически. Если передано целое число, это будет размер скользящего окна только для первого IMF. Возвращает IMFs : list  Эмпирические моды для каждого образца.  err\_windows\_size : list  Размеры окон для каждого уровня эмпирических мод. Примеры Вычисление EMD: длины окон равны трем для 1-ой и 2-ой моды, 5 для 3-ий моды и автоматически для остальных.  **>>> import** **numpy** **as** **np** **>>> from** **hsip.swemd.swemd** **import** SWEMD\_signal **>>>** data = np.random.rand(100) \* 10 *# Пример спектральных данных* **>>>** IMFs, windows = SWEMD(data, number\_of\_modes=8, windows\_size=[3, 3, 5]) **>>>** print(IMFs.shape, windows.shape) (100), (8) |