pysewer

Release 0.1.13

UFZ

Oct 20, 2023

CONTENTS:

CHAPTER

ONE

ABOUT

Pysewer provides a framework to automatically generate cost-efficient sewer network layouts using minimal data requirements.

It is build around an algorithm for generation of viable sewer-network layouts. The approximated sewer network is represented by sources (households/buildings), potential pathways, and one or multiple sinks. The algorithm approximates the directed Steiner tree (the Steiner arborescence) between all sources and the sink by using an repeated shortest path heuristic (RSPH).

1.1 Key features

- Automatic generation of gravity-flow prioritised sewer network layouts
- Minimal data requirements
- Sewer network visualisation
- Export of the sewer network for preliminary planning and analysis

1.2 Who is it for?

Pysewer is for anyone who wants to generate a sewer network layout with minimal data requirements. It is designed for sanitary engineers, critical infrastructure planners, researchers who need to generate sewer network layouts for preliminary planning and analysis.

1.3 Development team

Pysewer is developed by the working group on Water-Sensitive Infrastructure Planning (WASP) at the Helmholtz Centre for Environmental Research (UFZ) in Leipzig, Germany.

SPDX-FileCopyrightText: 2023 Helmholtz Centre for Environmental Research (UFZ) # SPDX-License-Identifier: GPL-3.0-only

CHAPTER

TWO

INSTALLATION

Currently the installation is easiest managed via Anaconda. Anaconda 3 can be downloaded here

2.1 Create a new environment

First, we want to create a new environment in Anaconda. Therefore, we open Anaconda prompt and create a new Python 3.10 Environment and name it pysewer by running the following command:

```
conda create -n pysewer python=3.10
```

We can then activate the environment by running:

```
conda activate pysewer
```

For installing GDAL, rasterio and fiona :

```
conda install -c conda-forge gdal rasterio fiona
```

2.2 Install pysewer via pip

You can either get pysewer and install it using git and pip with:

```
git clone https://git.ufz.de/despot/pysewer_dev.git
cd pysewer
pip install .
# for the development version
python -m pip install -e .
```

CHAPTER

THREE

PYSEWER MODULES

3.1 Default settings

class pysewer.config.settings.**Config**(*preprocessing: pysewer.config.settings.Preporocessing*, *optimization: pysewer.config.settings.Optimization*, *plotting: pysewer.config.settings.Plotting*, *export: pysewer.config.settings.Export*)

Bases: object

export: Export

optimization: Optimization

plotting: Plotting

preprocessing: Preporocessing

class pysewer.config.settings.**Export**(*file_format: str*)

Bases: object

file_format: str

class pysewer.config.settings.**Optimization**(*inhabitants_dwelling: int*, *daily_wastewater_person: float*, *peak_factor: float*, *min_slope: float*, *tmax: float*, *tmin: float*, *inflow_trench_depth: float*, *min_trench_depth: float*, *diameters: List[float]*, *roughness: float*, *pressurized_diameter: float*)

Bases: object

daily_wastewater_person: float

diameters: List[float]

inflow_trench_depth: float

inhabitants_dwelling: int

min_slope: float

min_trench_depth: float

peak_factor: float

pressurized_diameter: float

roughness: float

tmax: float

tmin: float

class pysewer.config.settings.**Plotting**(*plot_connection_graph: bool*, *plot_junction_graph: bool*, *plot_sink: bool*, *plot_sewer: bool*, *hillshade: bool*, *colormap: str*, *sewer_graph: networkx.classes.graph.Graph = None*, *info_table: dict | None = None*)

Bases: object

colormap: str

hillshade: bool

info_table: dict | None = None

plot_connection_graph: bool

plot_junction_graph: bool

plot_sewer: bool

plot_sink: bool

sewer_graph: Graph = None

class pysewer.config.settings.**Preporocessing**(*dem_file_path: str | None*, *roads_input_data: str | geopandas.geodataframe.GeoDataFrame | NoneType*, *buildings_input_data: str | geopandas.geodataframe.GeoDataFrame | NoneType*, *dx: int*, *pump_penalty: int*, *max_connection_length: int*, *clustering: str*, *connect_buildings: bool*, *add_private_sewer: bool*, *field_get_sinks: str*, *field_get_buildings: str*, *value_get_sinks: str*, *value_get_buildings: str*)

Bases: object

add_private_sewer: bool

buildings_input_data: str | GeoDataFrame | None

clustering: str

connect_buildings: bool

dem_file_path: str | None

dx: int

field_get_buildings: str

field_get_sinks: str

max_connection_length: int

pump_penalty: int

roads_input_data: str | GeoDataFrame | None

value_get_buildings: str

value_get_sinks: str

pysewer.config.settings.**config_to_dataframe**(*config: Config*) → DataFrame

pysewer.config.settings.**deep_merge**(*source*, *destination*)

pysewer.config.settings.**flatten_config**(*config_dict*, *parent_key=''*, *sep='_'*)

pysewer.config.settings.**load_config**(*custom_path: str | None = None*, *custom_setting_dict: dict | None = None*) → Config

Load the default settings and override them with custom settings if needed.

pysewer.config.settings.**load_settings**(*file_path: str | None = None*)

pysewer.config.settings.**override_setting_to_config**(*custom_path: str | None = None*, *custom_setting_dict: dict | None = None*)

pysewer.config.settings.**override_settings**(*custom_path: str | None = None*, *custom_setting_dict: dict | None = None*)

Override the settings with a custom settings file or a custom dictionary.

pysewer.config.settings.**view_default_settings**()

3.2 Export

Since the profile and trench_depth_profile are list of tuples, they cannot be exported to a shapefile or GeoPackage directly using the native GeoPandas function (to_file). This module contains functions to convert the list of tuples to JSON strings, which can then be exported to a shapefile or GeoPackage. The option saving the data as a parquet file is added.

pysewer.export.**export_sewer_network**(*gdf: GeoDataFrame*, *filepath: str*, *file_format: str = 'gpkg'*)

Export a sewer network GeoDataFrame to a file.

Parameters

- **gdf** (*gpd.GeoDataFrame*) – A GeoDataFrame containing the sewer network.
- **filepath** (*str*) – The path to the file to which the sewer network should be exported.
- **file_format** (*str*) – The file format to which the sewer network should be exported. Default is 'gpkg' (GeoPackage). Currently supported formats are 'gpkg' (GeoPackage), 'shp' (ESRI Shapefile) and Geoparquet 'parquet'.

Raises

ValueError – If the file format is not supported.

Return type

None

pysewer.export.**generate_schema**(*gdf: GeoDataFrame*)

Generate a schema based on the GeoDataFrame.

Parameters

gdf (*gpd.GeoDataFrame*) – The GeoDataFrame to generate the schema from.

Returns
The schema dictionary.

Return type
dict

pysewer.export.**is_list_of_tuples**(*column*)

Check if a pandas Series contains lists of tuples.

pysewer.export.**map_dtype_to_fiona**(*dtype*)

Map pandas data type to Fiona/OGR data type.

pysewer.export.**tuple_list_to_json**(*tuple_list*)

Serialize a list of tuples to a JSON string.

pysewer.export.**write_gdf_to_gpkg**(*gdf: GeoDataFrame*, *filepath: str*)

Write a GeoDataFrame to a GeoPackage (GPKG) file using Fiona, converting lists of tuples to JSON strings.

Parameters

- **gdf** (*gpd.GeoDataFrame*) – The GeoDataFrame to be written to the GPKG file.
- **filepath** (*str*) – The file path to the GPKG file.

Return type
None

Notes

This function converts any columns with list of tuples to JSON strings before writing to the GPKG file.

pysewer.export.**write_gdf_to_shp**(*gdf: GeoDataFrame*, *filepath: str*)

Write a GeoDataFrame to an ESRI Shapefile (SHP) file using Fiona, converting lists of tuples to JSON strings.

Parameters

- **gdf** (*gpd.GeoDataFrame*) – The GeoDataFrame to be written to the SHP file.
- **filepath** (*str*) – The file path to save the SHP file.

Return type
None

Notes

This function converts any columns in the GeoDataFrame that contain lists of tuples to JSON strings before writing to the SHP file.

3.3 Helper

pysewer.helper.**ckdnearest**(*gdfA: GeoDataFrame*, *gdfB: GeoDataFrame*, *gdfB_cols=['closest_edge']*) → GeoDataFrame

Returns a GeoDataFrame containing the closest geometry and attributes from gdfB to each geometry in gdfA.

pysewer.helper.**get_closest_edge**(*G: Graph*, *point: Point*) → tuple

Returns the closest edge to a given point in a networkx graph.

3.3.1 Parameters:

G
[networkx.Graph] The graph to search for the closest edge.

point
[shapely.geometry.Point] The point to search for the closest edge.

3.3.2 Returns:

closest_edge
[tuple] A tuple representing the closest edge to the given point.

pysewer.helper.**get_closest_edge_multiple**(*G: Graph*, *list_of_points: list*) → list

Returns a list of the closest edges in a networkx graph to a list of points.

Parameters

- **G** (*networkx.Graph*) – A networkx graph object.
- **list_of_points** (*list*) – A list of shapely Point objects.

Returns
A list of shapely LineString objects representing the closest edges to the input points.

Return type
list

pysewer.helper.**get_edge_gdf**(*G: Graph*, *field: str | None = None*, *value: any | None = None*, *detailed: bool = False*) → GeoDataFrame

Returns a GeoDataFrame of edges in a networkx graph that match a given field and value.

Parameters

- **G** (*networkx.Graph*) – The graph to extract edges from.
- **field** (*str, optional*) – The edge attribute to filter on.
- **value** (*any, optional*) – The value to filter the edge attribute on.
- **detailed** (*bool, optional*) – If True, returns a detailed GeoDataFrame with all edge attributes. If False, returns a simplified GeoDataFrame with only the edge geometry.

Returns
A GeoDataFrame of edges that match the given field and value.

Return type
gpd.GeoDataFrame

pysewer.helper.**get_edge_keys**(*G*, *field=None*, *value=None*)

Returns a list of edge keys (tuples of nodes) for the given graph *G* that have an edge attribute *field* with value *value*.

3.3.3 Parameters:

G
[networkx.Graph] The graph to search for edges.

field
[str, optional] The name of the edge attribute to filter on.

value
[any, optional] The value of the edge attribute to filter on.

3.3.4 Returns:

list of tuples
A list of edge keys (tuples of nodes) that have an edge attribute *field* with value *value*.

pysewer.helper.**get_mean_slope**(*G: MultiDiGraph*, *upstream: int*, *downstream: int*, *us_td: float*, *ds_td: float*) → float

Calculates the mean slope of a path between two nodes in a graph.

3.3.5 Parameters:

G
[networkx.MultiDiGraph] A directed graph object.

upstream
[int] The upstream node ID.

downstream
[int] The downstream node ID.

us_td
[float] The upstream node topographic elevation.

ds_td
[float] The downstream node topographic elevation.

3.3.6 Returns:

float
The mean slope of the path between the upstream and downstream nodes.

pysewer.helper.**get_node_gdf**(*G: Graph*, *field=None*, *value=None*) → GeoDataFrame

Returns a GeoDataFrame of nodes in the graph that match the specified field and value.

Parameters

- **G** (*nx.Graph*) – The input graph.
- **field** (*str, optional*) – The node attribute to filter on.
- **value** (*str, optional*) – The value to filter on.

Returns
gdf – A GeoDataFrame of nodes that match the specified field and value.

Return type
gpd.GeoDataFrame

Raises
ValueError – If no geometry column is found in the GeoDataFrame.

Notes

This function filters the nodes in the input graph based on the specified field and value, and returns a GeoDataFrame containing the filtered nodes and their attributes. The GeoDataFrame is created using the coordinates of the filtered nodes and their attributes.

pysewer.helper.**get_node_keys**(*G: Graph*, *field: str | None = None*, *value: str | None = None*)

Returns a list of keys for nodes in the graph *G* that have a specified *field* with a specified *value*.

3.3.7 Parameters:

G
[networkx.Graph] The graph to search for nodes.

field
[str, optional] The field to search for in the node data dictionary. If None, all nodes are returned.

value
[str] The value to search for in the specified *field*. If None, all nodes with the specified *field* are returned.

3.3.8 Returns:

list
A list of keys for nodes in the graph *G* that have a specified *field* with a specified *value*.

pysewer.helper.**get_path_distance**(*detailed_path: List[tuple]*) → float

Calculates the total distance of a path given a list of detailed path coordinates.

Parameters
detailed_path (*List[tuple]*) – A list of tuples representing the detailed path coordinates.

Returns
The total distance of the path.

Return type
float

Examples

```
>>> get_path_distance([(0, 0), (3, 4), (7, 1)])
9.848857801796104
```

pysewer.helper.**get_path_gdf**(*G*, *upstream*, *downstream*)

Returns a GeoDataFrame containing the geometry of the shortest path between two nodes in a graph.

3.3.9 Parameters:

G
[networkx.Graph] The graph to find the shortest path in.

upstream
[int] The starting node of the path.

downstream
[int] The ending node of the path.

3.3.10 Returns:

gpd.GeoDataFrame
A GeoDataFrame containing the geometry of the shortest path between the upstream and downstream nodes.

pysewer.helper.**get_sewer_info**(*G*)

Returns a dictionary with information about the sewer network.

Parameters
G (*networkx.Graph*) – The sewer network graph.

Returns
A dictionary containing the following information: - Total Buildings: Total number of buildings in the network. - Pressurized Sewer Length [m]: Total length of pressurized sewers in meters. - Gravity Sewer Length [m]: Total length of gravity sewers in meters. - Lifting Stations: Total number of lifting stations in the network. - Pumping Stations: Total number of pumping stations in the network (excluding those located in buildings). - Private Pumps: Total number of pumps located in buildings.

Return type
dict

pysewer.helper.**get_upstream_nodes**(*G: DiGraph*, *start_node*, *field: str*, *value: str*) → List

Returns a list of all upstream nodes in a directed graph *G* that have a node attribute *field* with value *value*, starting from *start_node* and traversing the graph in reverse order using a breadth-first search algorithm.

Parameters

- **G** (*networkx.DiGraph*) – The directed graph to traverse.
- **start_node** (*hashable*) – The node to start the traversal from.
- **field** (*str*) – The name of the node attribute to filter by.
- **value** (*Any*) – The value of the node attribute to filter by.

Returns
A list of all upstream nodes in *G* that have a node attribute *field* with value *value*.

Return type
List

pysewer.helper.**remove_third_dimension**(*geom*)

remove the third dimension of a shapely geometry

3.4 Optimization

pysewer.optimization.**calculate_hydraulic_parameters**(*G*, *sinks: list*, *pressurized_diameter: float = 0.2*, *diameters: List[float] = [0.1, 0.15, 0.2, 0.25, 0.3, 0.4]*, *roughness: float = 0.013*, *include_private_sewer: bool = True*)

Calculates hydraulic parameters for a sewer network graph.

Parameters

- **G** (*networkx.Graph*) – The sewer network graph.
- **sinks** (*list*) – A list of sink nodes in the graph.
- **pressurized_diameter** (*float*) – The diameter of pressurized pipes in the network.
- **diameters** (*list*) – A list of available pipe diameters.
- **roughness** (*float*) – The roughness coefficient of the pipes.
- **include_private_sewer** (*bool, optional*) – Whether to include private sewer connections in the graph, by default True.

Returns
The sewer network graph with updated hydraulic parameters.

Return type
networkx.Graph

Notes

This function places pumps/lifting stations on linear sections between road junctions. Three cases are possible: 1. Terrain does not allow for gravity flow to the downstream node (this check uses the "needs_pump" attribute from the preprocessing to reduce computational load) -> place pump 2. Terrain does not require pump but lowest inflow trench depth is too low for gravitational flow -> place lifting station 3. Gravity flow is possible within given constraints

pysewer.optimization.**estimate_peakflow**(*G: Graph*, *inhabitants_dwelling: int = 3*, *daily_wastewater_person: float = 0.2*, *peak_factor: float = 2.3*)

Estimate the peakflow in m^3/s for a node n in Graph G.

Parameters

- **G** (*networkx.Graph*) – The graph to estimate peakflow for.
- **inhabitants_dwelling** (*int*) – The number of inhabitants per dwelling.
- **daily_wastewater_person** (*float*) – The daily wastewater generated per person in m^3.
- **peak_factor** (*float, optional*) – The peak factor to use in the calculation, by default 2.3.

Returns
The graph with updated node attributes for peak flow, average daily flow, and upstream pe.

Return type
networkx.Graph

pysewer.optimization.**get_downstream_junction**(*G: Graph*, *node: int*)

Returns the next downstream junction from the specified node in G.

Parameters

- **G** (*networkx.Graph*) – The graph to search for the downstream junction.
- **node** (*int*) – The node to start the search from.

Returns
The downstream junction from the specified node in G.

Return type
int

Notes

The downstream junction is defined as the next junction in the graph that has a degree greater than 2 or an out-degree of 0.

pysewer.optimization.**get_junction_front**(*G: Graph*, *junctions*)

Returns a list of junctions or terminals which have as many entries for inflow trench depths as they have incoming edges.

Parameters

- **G** (*networkx.DiGraph*) – A directed graph representing the sewer network.
- **junctions** (*list*) – A list of junctions or terminals in the sewer network.

Returns
A list of junctions or terminals which have as many entries for inflow trench depths as they have incoming edges.

Return type
list

pysewer.optimization.**get_max_upstream_diameter**(*G*, *edge: tuple*)

Returns the maximum diameter of all upstream edges of the given edge in the directed graph G.

Parameters

- **G** (*networkx.DiGraph*) – The directed graph.
- **edge** (*tuple*) – The edge for which to find the maximum upstream diameter.

Returns
The maximum diameter of all upstream edges of the given edge.

Return type
float

pysewer.optimization.**mannings_equation**(*pipe_diameter: float*, *roughness: float*, *slope: float*) → float

Calculates the volume flow rate of a pipe using Manning's equation.

Parameters

- **pipe_diameter** (*float*) – Diameter of the pipe in meters.
- **roughness** (*float*) – Roughness coefficient of the pipe.
- **slope** (*float*) – Slope of the pipe in units of elevation drop per unit length.

Returns
Volume flow rate of the pipe in cubic meters per second.

Return type
float

Raises
ValueError – If the slope is greater than 0.

Notes

Manning's equation is used to calculate the volume flow rate of a pipe based on its diameter, roughness coefficient, and slope.

pysewer.optimization.**needs_pump**(*profile*, *min_slope: float = -0.01*, *tmax: float = 8*, *tmin: float = 0.25*, *inflow_trench_depth: float = 0*)

Traces a profile to determine if gravitational flow can be achieved within slope and trench depth constraints.

Parameters

- **profile** (*list of tuples*) – A list of (x, y) tuples representing the profile to be traced.
- **min_slope** (*float, optional*) – The minimum slope required for gravitational flow. Default is -0.01.
- **tmax** (*float, optional*) – The maximum trench depth allowed. Default is 8.
- **tmin** (*float, optional*) – The minimum trench depth allowed. Default is 0.25.
- **inflow_trench_depth** (*float, optional*) – The trench depth at the inflow point. If not specified, it is set to tmin.

Returns
A tuple containing: - A boolean indicating whether a pump is needed. - The height difference between the outflow and the trench depth at the outflow point. - A list of (x, trench_depth) tuples representing the trench depth at each point along the profile.

Return type
tuple

pysewer.optimization.**place_lifting_station**(*G*, *node*)

Places a lifting station at the specified node in the graph.

Parameters

- **G** (*networkx.Graph*) – The graph to add the lifting station to.
- **node** (*int*) – The node to add the lifting station to.

Returns
The graph with the added lifting station.

Return type
networkx.Graph

pysewer.optimization.**place_pump**(*G*, *node*)

Places a pump at the specified node in the graph G and sets downstream edges "pressurized" attribute.

Parameters

- **G** (*networkx.Graph*) – The graph in which the pump is to be placed.
- **node** (*hashable*) – The node at which the pump is to be placed.

Returns
The graph with the pump placed at the specified node and downstream edges "pressurized" attribute set.

Return type
networkx.Graph

pysewer.optimization.**reverse_bfs**(*G*, *sink: str*, *include_private_sewer: bool = True*)

Returns an iterator over edges in a sequential fashion, starting at the terminals (i.e. buildings) and returning all upstream edges of a junction before moving downstream

Parameters

- **G** (*networkx.DiGraph*) – The graph to traverse
- **sink** (*str*) – The node to start the traversal from
- **include_private_sewer** (*bool, optional*) – Whether to include private sewer nodes in the traversal, by default True

Yields
tuple – A tuple representing an edge in the graph, in the form (source, target)

pysewer.optimization.**select_diameter**(*target_flow: float*, *diameters: List[float]*, *roughness: float*, *slope: float*)

Returns the minimum pipe diameter.

Parameters

- **target_flow** (*float*) – The target flow rate in cubic meters per second.
- **diameters** (*list*) – A list of possible pipe diameters in meters.
- **roughness** (*float*) – The pipe roughness coefficient in meters.
- **slope** (*float*) – The pipe slope in meters per meter.

Returns
The minimum pipe diameter required to achieve the target flow rate.

Return type
float

Raises
ValueError – If the maximum diameter is insufficient to reach the target flow rate.

pysewer.optimization.**set_diameter**(*G: Graph*, *edge: tuple*, *diameter: float*)

Set the diameter of an edge in a graph.

3.4.1 Parameters:

G
[networkx.Graph] The graph to modify.

edge
[tuple] The edge to modify.

diameter
[float] The diameter to set.

3.4.2 Returns:

networkx.Graph
The modified graph.

3.5 Visualization

pysewer.plotting.**plot_model_domain**(*modelDomain*, *plot_connection_graph: bool = False*, *plot_junction_graph: bool = False*, *plot_sink: bool = False*, *plot_sewer: bool = False*, *sewer_graph: Graph | None = None*, *info_table: dict | None = None*, *hs_alt=30*, *hs_az=0*, *hillshade: bool = False*, *fig_size: tuple = (20, 20)*)

Plots the sewer network model domain.

Parameters

- **modelDomain** (*pysewer.ModelDomain*) – The model domain to plot.
- **plot_connection_graph** (*bool, optional*) – Whether to plot the connection graph, by default False.
- **plot_junction_graph** (*bool, optional*) – Whether to plot the junction graph, by default False.
- **plot_sink** (*bool, optional*) – Whether to plot the sink, by default True.
- **plot_sewer** (*bool, optional*) – Whether to plot the sewer, by default False.
- **sewer_graph** (*networkx.Graph, optional*) – The sewer graph to plot, by default None.
- **info_table** (*dict, optional*) – The information table to plot, by default None.
- **hs_alt** (*int, optional*) – The altitude of the hillshade, by default 30.
- **hs_az** (*int, optional*) – The azimuth of the hillshade, by default 0.
- **hillshade** (*bool, optional*) – Whether to plot the hillshade, by default False.

Returns
fig, ax – The figure and axes of the plot.

Return type
matplotlib.figure.Figure, matplotlib.axes.Axes

pysewer.plotting.**plot_sewer_attributes**(*modelDomain*, *sewer_graph*, *attribute*, *colormap='jet'*, *title='Sewer Network Plot'*, *hillshade=False*, *fig_size=(20, 20)*)

Plots the sewer network with the specified attribute.

Parameters

- **modelDomain** (*object*) – The model domain object.
- **sewer_graph** (*object*) – The sewer graph object.
- **attribute** (*str*) – The attribute to plot.
- **colormap** (*str, optional*) – The colormap to use for the plot. Default is “jet”.
- **title** (*str, optional*) – The title of the plot. Default is “Sewer Network Plot”.
- **hillshade** (*bool, optional*) – Whether to include a hillshade in the plot. Default is False.

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure object.
- **ax** (*matplotlib.axes.Axes*) – The axes object.

3.6 Preprocessing

class pysewer.preprocessing.**Buildings**(*input_data: str | GeoDataFrame*, *roads_obj:* Roads)

Bases: `object`

A class to preprocess building data.

Parameters

- **input_data** (*str or geopandas.GeoDataFrame*) – Path to a shapefile or a GeoDataFrame containing the input data.
- **roads_obj** (*pysewer.Roads*) – A Roads object containing the road network data.

gdf

The input building data.

Type
geopandas.GeoDataFrame

roads_obj

The road network data.

Type
pysewer.Roads

get_gdf():

Returns building data as geopandas dataframe.

get_crs():

Returns the Coordinate System of the DEM Object.

cluster_centers(max_connection_length):

Returns a list of cluster centers for all buildings with greater than max_connection_length distance to the nearest street.

cluster_centers(*max_connection_length: float*)

Returns a list of cluster centers for all buildings with greater than max_connection_length distance to the nearest street.

Parameters
max_connection_length (*float*) – The maximum distance between a building and the nearest street for it to be included in the cluster centers list.

Returns
A GeoDataFrame containing the cluster centers and their distances to the nearest street, sorted by distance.

Return type
geopandas.GeoDataFrame

get_crs()

Returns the Coordinate System of the DEM Object.

Returns
The Coordinate Reference System (CRS) of the building data.

Return type
dict

get_gdf()

Returns building data as geopandas dataframe.

Returns
The building data.

Return type
geopandas.GeoDataFrame

class pysewer.preprocessing.**DEM**(*file_path: str | None = None*)

Bases: object

A class for handling digital elevation model (DEM) data.

file_path

The file path to the DEM raster file.

Type
str, optional

raster

The rasterio dataset reader object for the DEM raster file.

Type
rasterio.DatasetReader, optional

no_dem

A flag indicating whether or not a DEM has been loaded.

Type
bool

get_elevation(*point: shapely.geometry.Point*) → int:

Returns elevation data in meters for a given point rounded to two decimals.

get_profile(*line: shapely.geometry.LineString*, *dx: int = 10*) → List[Tuple[float, int]]:

Extracts elevation data from a digital elevation model (DEM) along a given path.

reproject_dem(*crs: CRS*) → None:

Reprojects the DEM raster to the given CRS.

Properties()

----

get_crs : CRS

Returns the coordinate system of the DEM object.

file_path: str | None = None

property get_crs: CRS | None

Returns the coordinate system of the DEM Object

get_elevation(*point: Point*)

Returns elevation data in meters for a given point rounded to two decimals.

Parameters
point (*shapely.geometry.Point*) – The point for which to retrieve elevation data.

Returns
The elevation in meters.

Return type
int

Raises
ValueError – If the query point is out of bounds or if there is no elevation data for the given coordinates.

get_profile(*line: LineString*, *dx: int = 10*)

Extracts elevation data from a digital elevation model (DEM) along a given path.

Parameters

- **line** (*shapely.geometry.LineString*) – The path along which to extract elevation data.
- **dx** (*float, optional*) – The sampling resolution in meters. Default is 10.

Returns
A list of (x, elevation) tuples representing the x-coordinate and elevation data of the profile. The x-coordinate values start at 0 and are spaced at intervals of dx meters.

Return type
List of Tuples

no_dem: bool = True

raster: DatasetReader = None

reproject_dem(*crs: CRS*)

Reprojects the DEM raster to the given CRS.

Parameters
crs (*CRS*) – The target CRS to reproject the raster to.

Raises
ValueError – If no DEM is loaded, cannot reproject DEM.

class pysewer.preprocessing.**ModelDomain**(*dem: str*, *roads: str*, *buildings: str*, *clustering: str = 'centers'*, *pump_penalty: int = 1000*, *connect_buildings: bool = True*)

Bases: object

Class for preprocessing input data for the sewer network.

Parameters

- **dem** (*str*) – Path to the digital elevation model file.
- **roads** (*str*) – Path to the roads shapefile.
- **buildings** (*str*) – Path to the buildings shapefile.
- **clustering** (*str, optional*) – Clustering method for connecting buildings to the sewer network. Default is “centers”.

- **pump_penalty** (*int, optional*) – Penalty for adding a pump to the sewer network. Default is 1000.
- **connect_buildings** (*bool, optional*) – Whether to connect buildings to the sewer network. Default is True.

roads

Roads object.

Type
Roads

buildings

Buildings object.

Type
Buildings

dem

DEM object.

Type
DEM

connection_graph

Graph representing the road network.

Type
nx.Graph

pump_penalty

Penalty for adding a pump to the sewer network.

Type
int

add_node(*point*, *node_type*, *closest_edge=None*, *node_attributes=None*)

Adds a node to the connection graph.

Parameters

- **point** (*shapely.geometry.Point*) – The point to add as a node.
- **node_type** (*str*) – The type of node to add.
- **closest_edge** (*shapely.geometry.LineString, optional*) – The closest edge to the point. Defaults to None.
- **node_attributes** (*dict, optional*) – Additional attributes to add to the node. Defaults to None.

Return type
None

Notes

This method adds a node to the connection graph. If *closest_edge* is not provided, it finds the closest edge to the point and uses that as the *closest_edge*. It then disconnects edges from the node and adds the node to the connection graph. If there are any cluster centers, it connects the node to the nearest cluster center. If there are no cluster centers, it connects the node to the road network.

add_sink(*sink_location: tuple*)

Adds a sink node to the graph at the specified location.

Parameters
sink_location (*tuple*) – A tuple containing the x and y coordinates of the sink location.

Return type
None

connect_buildings(*max_connection_length: int = 30*, *clustering: str = 'centers'*)

Connects the buildings in the network by adding nodes to the graph. :param max_connection_length: The maximum distance between a building and the nearest street for it to be included in the cluster

centers list. The default is 30.

Parameters
clustering (*str, optional*) – The method used to cluster the buildings. Can be "centers" or "none". The default is "centers".

Return type
None

Notes

This method adds nodes to the graph to connect the buildings in the network. It first gets the building points and then clusters them based on the specified method. If clustering is set to "centers", it gets the cluster centers and finds the closest edges to them. It then adds nodes to the graph for each cluster center, with the closest edge as an attribute. If clustering is set to "none", it simply adds nodes to the graph for each building. In both cases, it finds the closest edges to the buildings and adds nodes to the graph for each building, with the closest edge as an attribute.

Examples

```
>>> network = Network()
>>> network.connect_buildings(max_connection_length=50, clustering="centers")
```

connect_subgraphs()

Identifies unconnect street subnetworks and connects them based on shortest distance

connect_to_roadnetwork(*G: Graph*, *new_node*, *conn_point*, *closest_edge*, *add_private_sewer: bool = True*)

Connects a new node to the road network by inserting a connection point on the closest edge and adjusting edges.

Args:
G (networkx.Graph): The road network graph. new_node (NodeTpye): The new node to be connected to the road network. conn_point (pysewer.Point): The point where the new node will be connected to the road network. closest_edge (pysewer.Edge): The closest edge to the new node. add_private_sewer

(bool, optional): Whether to add a private sewer between the new node and the connection point. Defaults to True.

Returns:
bool: True if the connection was successful, False otherwise.

`create_unsimplified_graph`(*roads_gdf: GeoDataFrame*) → Graph

Create an unsimplified graph from a GeoDataFrame of roads. :param roads_gdf: A GeoDataFrame containing road data. :type roads_gdf: gpd.GeoDataFrame

Returns
An unsimplified graph containing nodes and edges from the GeoDataFrame.

Return type
nx.Graph

`generate_connection_graph`() → MultiDiGraph

Generates a connection graph from the given connection data and returns it. This method simplifies the connection graph, removes any self loops, sets trench depth node attributes to 0, calculates the geometry, distance, profile, needs_pump, weight, and elevation attributes for each edge and node in the connection graph.

3.6.1 Returns:

nx.MultiDiGraph
A directed graph representing the connections between the different points in the network.

`get_buildings`()

Returns a list of node keys for all buildings in the connection graph.

`get_sinks`()

Returns a list of node keys for all wastewater treatment plants (wwtp) in the connection graph.

`reset_sinks`()

Resets the sinks in the connection graph by setting their node_type attribute to an empty string. :returns: This method does not return anything. :rtype: None

`set_pump_penalty`(*pp*)

Set the pump penalty for the current instance of the ModelDomain class. :param pp: The pump penalty to set. :type pp: float

Return type
None

`set_sink_lowest`(*candidate_nodes: list | None = None*)

Sets the sink node to the lowest point in the graph.

Parameters
`candidate_nodes` (*list, optional*) – A list of candidate nodes to consider for the sink node. If None, all nodes except buildings are considered.

Return type
None

Notes

This method sets the sink node to the lowest point in the graph. If candidate_nodes is not None, only the nodes in candidate_nodes that are not buildings are considered.

class pysewer.preprocessing.**Roads**(*input_data: str | GeoDataFrame*)

Bases: object

A class to represent road data from either a shapefile or a geopandas dataframe. Attributes: ————- gdf : geopandas.GeoDataFrame

A geopandas dataframe containing road data.

merged_roads
[shapely.geometry.MultiLineString] A shapely MultiLineString object representing the merged road data.

3.6.2 Methods:

__init__(self, input_data: str or geopandas.GeoDataFrame) -> None
Initializes a Roads object with road data from either a shapefile or a geopandas dataframe.

get_crs()

Gets the coordinate reference system (CRS) of the Roads object.

Returns
The coordinate system of the Roads object.

Return type
dict

get_gdf()

Returns the road data as a geopandas dataframe. :returns: The road data as a geopandas dataframe. :rtype: geopandas.GeoDataFrame

get_merged_roads()

Merge the road network as a shapely MultiLineString.

Returns
merged road network as a shapely MultiLineString

Return type
shapely MultiLineString

get_nearest_point(*point*)

Returns the nearest location to the input point on the street network. :param point: Point to find nearest location to. :type point: shapely.geometry.Point

Returns
Nearest location to the input point on the street network.

Return type
shapely.geometry.Point

3.7 Routing

pysewer.routing.**find_rsph_path**(*connection_graph: Graph*, *subgraph_nodes: List[int | str | Hashable]*, *terminals: List[int | str | Hashable]*, *all_paths: dict*, *all_lengths: dict*) → List[int | str | Hashable]

Identifies the closest terminal node to the growing tree and returns the connection path.

Parameters

- **connection_graph** (*nx.Graph*) – The graph representing the entire network.
- **subgraph_nodes** (*List[NodeType]*) – The nodes in the subgraph being grown.
- **terminals** (*List[NodeType]*) – The terminal nodes in the network.
- **all_paths** (*dict*) – A dictionary containing all the shortest paths between terminal nodes.
- **all_lengths** (*dict*) – A dictionary containing the lengths of all the shortest paths between terminal nodes.

Returns
The shortest path between the closest terminal node and the growing subgraph.

Return type
List[NodeType]

Raises
ValueError – If there is no recorded path from the closest terminal node to the growing subgraph in the all_paths dictionary.

pysewer.routing.**rsph_tree**(*connection_graph: Graph*, *sinks: List[int | str | Hashable]*, *from_type: str = 'building'*, *to_type: str = 'wwtp'*, *skip_nodes: List[int | str | Hashable] = []*) → DiGraph

Returns the directed routed steiner tree that connects all terminal nodes to the sink using the repeated shortest path heuristic.

Parameters

- **connection_graph** (*nx.Graph*) – The graph representing the sewer network.
- **sinks** (*List[NodeType]*) – The nodes representing the sinks in the sewer network.
- **from_type** (*str, optional*) – The type of the source nodes, by default “building”.
- **to_type** (*str, optional*) – The type of the sink nodes, by default “wwtp”.
- **skip_nodes** (*List[NodeType], optional*) – The nodes to skip in the routing, by default [].

Returns
The directed routed steiner tree that connects all terminal nodes to the sink using the repeated shortest path heuristic.

Return type
nx.DiGraph

3.8 Simplify Graph

pysewer.simplify.**get_essential_nodes**(*G*)

Returns a list of essential nodes to build the simplified graph. This includes all junctions (degree > 2), buildings, and connection points.

Parameters
G (*networkx.Graph*) – NetworkX street graph with connected buildings.

Returns
List of node keys.

Return type
List

Examples

```
>>> G = nx.Graph()
>>> G.add_node(1, road_network=True)
>>> G.add_node(2, road_network=True)
>>> G.add_node(3, road_network=False)
>>> G.add_node(4, connection_node=True)
>>> G.add_node(5, connection_node=False)
>>> G.add_edge(1, 2)
>>> G.add_edge(2, 3)
>>> G.add_edge(2, 4)
>>> G.add_edge(4, 5)
>>> get_essential_nodes(G)
[1, 2, 3, 5]
```

pysewer.simplify.**simplify_graph**(*G: MultiDiGraph*, *strict: bool = True*, *remove_rings: bool = True*) → MultiDiGraph

Simplify a graph's topology by removing interstitial nodes. Simplify graph topology by removing all nodes that are not intersections or dead-ends. Create an edge directly between the end points that encapsulate them, but retain the geometry of the original edges, saved as an attribute in new edge. Some of the resulting consolidated edges may comprise multiple OSM ways, and if so, their multiple attribute values are stored as a list. :param G: input graph :type G: networkx.MultiDiGraph :param strict: if False, allow nodes to be end points even if they fail all other

rules but have incident edges with different OSM IDs. Lets you keep nodes at elbow two-way intersections, but sometimes individual blocks have multiple OSM IDs within them too.

Parameters
remove_rings (*bool*) – if True, remove isolated self-contained rings that have no endpoints

Returns
G – topologically simplified graph

Return type
networkx.MultiDiGraph

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

G

H

I

L

M

N

O

P

R

S

T

V

W