PowerGridSynth

Release 0.1.0

PowerGridSynth Developers

Dec 22, 2025

CONTENTS:

Documentation sources

CHAPTER ONE

TOPOLOGY GENERATOR

class powergrid_synth.**PowerGridGenerator**(*seed: int | None = None*)

Bases: object

Implements Algorithm 4: Generative model for entire power grid graph on k voltage levels.

generate_grid(*degrees_by_level: List[List[int]]*, *diameters_by_level: List[int]*, *transformer_degrees: Dict[Tuple[int, int], Tuple[List[int], List[int]]]*, *keep_lcc: bool = True*) → Graph

Procedure CLCSTARS({d_xi}, {delta_xi}, {t[Xi, Xj]}) -> E

Parameters

- **degrees_by_level** – List of degree sequences, one for each voltage level.
- **diameters_by_level** – List of target diameters, one for each voltage level.
- **transformer_degrees** – Dictionary mapping level pairs (i, j) to a tuple of transformer degree lists.
- **keep_lcc** – If True, returns only the Largest Connected Component of the generated grid, removing isolated islands. Default: True

Returns

A NetworkX graph representing the entire multi-level grid.

class powergrid_synth.**InputConfigurator**(*seed: int | None = None*)

Bases: object

Helper class to generate the detailed input arrays (degrees, transformer connections) required by PowerGridGenerator from high-level parameters.

create_params(*levels: List[Dict[str, Any]]*, *inter_connections: Dict[Tuple[int, int], Dict[str, Any]]*) → Dict[str, Any]

Generates the full parameter set.

Parameters

inter_connections – Dict mapping (i, j) to config. Config can be {'type': 'simple', 'p_i_j': …, 'p_j_i': …} OR {'type': 'k-stars', 'c': 0.174, 'gamma': 4.15}

CHAPTER

TWO

GRID DATA GENERATOR

class powergrid_synth.**BusTypeAllocator**(*graph: Graph*, *entropy_model: int = 0*)

Bases: `object`

Assigns bus types (Generator, Load, Connection) to a power grid topology using an Artificial Immune System (AIS) optimization algorithm to match target topological entropy properties.

Ported and adapted from 'sg_bus_type.m' (SynGrid).

TYPE_CONN = 3

TYPE_GEN = 1

TYPE_LOAD = 2

allocate(*max_iter: int = 100*, *population_size: int = 20*) → Dict[int, str]

Main execution method. Runs the AIS optimization.

Returns

Dictionary mapping node_id -> 'Gen', 'Load', or 'Conn'

class powergrid_synth.**CapacityAllocator**(*graph: Graph*, *ref_sys_id: int = 1*)

Bases: `object`

Assigns generation capacities (PgMax) to generator buses in the grid. Ported and adapted from 'sg_gen_capacity.m' (SynGrid).

allocate(*tab_2d: ndarray | None = None*) → Dict[int, float]

Main execution method.

Parameters

tab_2d – Optional 14x14 probability matrix. If None, uses default based on ref_sys_id.

Returns

Dictionary mapping Generator Node ID -> Capacity (PgMax)

class powergrid_synth.**LoadAllocator**(*graph: Graph*, *ref_sys_id: int = 1*)

Bases: `object`

Assigns active power loads (PL) to load buses in the grid. Ported and adapted from 'sg_load.m' (SynGrid).

allocate(*loading_level: str = 'H'*) → Dict[int, float]

Allocates loads to buses.

Parameters

loading_level – 'D' (Default Formula), 'L' (Light), 'M' (Medium), 'H' (Heavy).

Returns

Dictionary mapping Load Node ID -> Active Power (MW)

class powergrid_synth.**TransmissionLineAllocator**(*graph: Graph*, *ref_sys_id: int = 1*)

Bases: object

Allocates impedance (X, R) and Capacity Limits to transmission lines. Based on 'sg_flow_lim.m' from SynGrid.

Steps: 1. Initialize random impedances (Zpr) based on LogNormal distribution. 2. Run iterative DCPF (Swapping Logic):

- Calculate flows.
- Assign lower Impedance (Z) to lines with higher Flow.
- Perform random swaps to introduce variance.

3. (Optional) Topology Refinement: - Add low-impedance lines to bridge large phase angle differences. - Remove weak (high-impedance) lines to maintain grid density.
4. Allocate Capacity: - Use 'Tab_2D_FlBeta' to assign Capacity Factors (Beta). - Capacity = Flow / Beta.

allocate(*refine_topology: bool = False*) → Dict[Tuple[int, int], float]

Main execution method.

class powergrid_synth.**GenerationDispatcher**(*graph: Graph*, *ref_sys_id: int = 1*)

Bases: object

Allocates active power setpoints (Pg) to generators. Ported and adapted from 'sg_gen_dispatch.m' (SynGrid).

dispatch() → Dict[int, float]

Utilities

class powergrid_synth.**GridVisualizer**

Bases: object

Visualization module for synthetic power grids. Allows plotting the grid with different layouts including Yifan Hu, Kamada-Kawai, and Voltage Layered.

plot_bus_types(*graph: Graph*, *layout: str = 'kamada_kawai'*, *title: str = 'Bus Type Visualization'*, *show_impedance: bool = False*, *figsize: Tuple[int, int] = (12, 10)*)

Visualizes the grid coloring nodes by their Bus Type (Static). Option to show impedance on edges.

plot_grid(*graph: Graph*, *layout: str = 'kamada_kawai'*, *title: str = 'Grid'*, *show_labels: bool = False*, *show_bus_types: bool = False*, *show_impedance: bool = False*, *figsize: Tuple[int, int] = (12, 10)*)

Static plot function for grid topology. Options allow overlaying bus types or impedance features.

plot_impedance(*grid: Graph*, *layout: str = 'kamada_kawai'*, *title: str = 'Transmission Line Impedance'*, *figsize: Tuple[int, int] = (12, 10)*)

Plots the grid with edges colored by their impedance magnitude (Z). Blue = Low Impedance (Strong), Red = High Impedance (Weak).

plot_interactive(*graph: Graph*, *title: str = 'Interactive Grid'*, *figsize: Tuple[int, int] = (14, 10)*)

Opens an interactive window for the full grid.

plot_interactive_bus_types(*graph: Graph*, *title: str = 'Interactive Bus Type Visualization'*, *figsize: Tuple[int, int] = (14, 10)*)

Opens an interactive window for Bus Type Visualization with layout selection.

plot_interactive_voltage_level(*graph: Graph*, *level: int*, *title: str | None = None*, *figsize: Tuple[int, int] = (12, 10)*)

Opens an interactive window for a specific voltage level.

plot_load_gen_bubbles(*grid: Graph*, *layout: str = 'kamada_kawai'*, *title: str = 'Generation vs Load'*, *show_impedance: bool = False*, *figsize: Tuple[int, int] = (12, 10)*)

Bubble plot showing generation and load magnitudes. Generators are blue squares, Loads are red circles. Size is proportional to capacity/load. Optionally plots impedance on edges.

class powergrid_synth.**GridExporter**(*graph: Graph*, *base_mva: float = 100.0*, *base_kv: float = 230.0*)

Bases: `object`

Exports the generated synthetic grid to standard file formats.

1. MATPOWER (.m): - Standard bridge for Pypowsybl (pypowsybl.network.load) - Supported by Pandapower (pandapower.converter.from_mpc)
2. Pandapower CSVs (Folder): - Converts PU to Physical units (Ohms, kA) based on V_base=230kV. - Column names match pandapower.create_* parameters.

export_to_matpower(*filepath: str*)

Saves the grid to a MATPOWER (.m) file. This is the preferred format for bridging with Pypowsybl.

export_to_pandapower_csv(*folder_path: str*)

Exports CSVs with Physical Units (Ohm, kA) for Pandapower.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX