Power System Equipment Classes

Comprehensive Documentation for Milestone 1

Table of Contents

1. Bus Class

Purpose

The `Bus` class represents a node (connection point) in an electrical power system where equipment such as generators, loads, transformers, and transmission lines connect. Each bus is automatically assigned a unique index for identification in network matrices.

Attributes

name *(str)*

The name identifier of the bus.

bus_index *(int)* Read-Only

Unique integer index automatically assigned to each bus instance using a class-level counter.

nominal_kv *(float, optional)*

The nominal voltage of the bus in kilovolts.

voltage *(float)* Property

Returns the nominal voltage value. This is a read-only property intended to be set by solver classes via the private `_set_voltage()` method.

Key Features

Automatic Indexing: Each Bus instance receives a unique index via the class-level `_bus_index` counter, which increments with each instantiation.

Read-Only Voltage Property: Users can read voltage but cannot directly set it. Only solver classes should use the private `_set_voltage()` method.

Index Counter Reset: The class method `reset_index_counter()` allows resetting the counter to 1, useful for testing scenarios.

Example Usage

```
# Create buses bus1 = Bus("Bus-1", 20.0) bus2 = Bus("Bus-2",
230.0) # Access attributes print(bus1.name) # "Bus-1"
print(bus1.bus_index) # 1 print(bus2.bus_index) # 2 # Solver
sets voltage (internal use) bus1._set_voltage(240.0)
print(bus1.voltage) # 240.0
```

Test Summary

7 Tests Included:

- ✓ Basic creation with automatic indexing (first bus gets index 1)
- ✓ Sequential indexing (second bus gets index 2)
- ✓ Voltage setting via internal method (_set_voltage)
- ✓ Property access and unique indexing validation
- ✓ Read-only voltage property enforcement (raises AttributeError)
- ✓ Index counter reset functionality
- ✓ Complete workflow demonstration with multiple buses

2. Transformer Class

Purpose

The `Transformer` class models a transformer as a series impedance element connecting two buses in a power system network. It automatically computes series admittance parameters for network matrix construction.

Attributes

name *(str)*

Transformer identifier.

bus1 *(str)*

Connected bus name on side 1.

bus2 *(str)*

Connected bus name on side 2.

r *(float)*

Series resistance in per-unit or ohms (consistent with system base).

x *(float)*

Series reactance in per-unit or ohms (consistent with system base).

g *(float)* Computed Read-Only

Series conductance (siemens).

```
g = r / (r² + x²)
```

b *(float)* Computed Read-Only

Series susceptance (siemens).

```
b = -x / (r² + x²)
```

Key Features

Computed Admittance: The conductance `g` and susceptance `b` are automatically calculated from resistance and reactance.

Dynamic Recalculation: When `r` or `x` are updated via setters, `g` and `b` are automatically recomputed.

Validation: Ensures `r` and `x` are non-negative and cannot both be zero.

Example Usage

```
# Create transformer t1 = Transformer( name="T1", bus1="Bus-1", bus2="Bus-2", r=0.01, x=0.10 ) # Access computed admittance print(t1.g) # Conductance print(t1.b) # Susceptance # Update impedance (admittance automatically recalculates) t1.r = 0.02 print(t1.g) # New conductance value
```

Test Summary

5 Tests Included:

- ✓ Basic admittance computation from r=0.02, x=0.04
- ✓ Dynamic updates when r changes (admittance recalculates)
- ✓ Dynamic updates when x changes (admittance recalculates)
- ✓ Zero impedance rejection (r=0 and x=0 raises ValueError)
- ✓ Negative value rejection (negative r or x raises ValueError)

3. TransmissionLine Class

Purpose

The `TransmissionLine` class represents a transmission line connecting two buses using lumped parameter modeling. It computes series conductance dynamically based on resistance and reactance.

Attributes

name *(str)*

Identifier for the transmission line.

bus1_name *(str)*

From-bus name (sending end).

bus2_name *(str)*

To-bus name (receiving end).

r *(float)*

Series resistance in ohms.

x *(float)*

Series reactance in ohms.

g *(float)* Computed Read-Only

Series conductance in siemens.

```
g = r / (r² + x²)
```

Key Features

Property-Based Access: All attributes use properties with validation on setters.

Computed Conductance: The `g` property dynamically computes conductance based on current `r` and `x` values.

String Trimming: Name properties automatically strip whitespace.

Type Coercion: Numeric setters convert values to float.

Example Usage

```
# Create transmission line line1 = TransmissionLine(
name="Line-1", bus1_name="Bus-1", bus2_name="Bus-2", r=0.02,
x=0.25 ) # Access computed conductance print(line1.g) # 0.02 /
(0.02² + 0.25²) # Update reactance line1.x = 0.30
print(line1.g) # Automatically recalculated
```

Test Summary

6 Tests Included:

✓ Basic g computation (g=0.5 when r=1.0, x=1.0)

✓ g updates when r changes (r=2.0 updates g to 0.4)

✓ g updates when x changes (x=3.0 updates g to 2.0/13.0)

- ✓ Empty name rejection (raises ValueError)
- ✓ Negative r rejection (raises ValueError)
- ✓ Zero r and x rejection (both zero raises ValueError)

4. Load Class

Purpose

The `Load` class represents a constant real and reactive power load connected to a bus. It models power consumption and automatically computes apparent power using the power triangle relationship.

Attributes

`name` *(str)*

Identifier for the load.

`bus1_name` *(str)*

Bus name where the load is connected.

`mw` *(float)*

Real power (active power) in megawatts.

mvar *(float)*

Reactive power in megavolt-amperes reactive.

mva *(float)* Computed Read-Only

Apparent power in megavolt-amperes.

```
MVA = √(MW² + MVAr²)
```

Key Features

Power Triangle Computation: The `mva` property implements the standard power triangle relationship.

Flexible Power Values: No restrictions on sign; allows modeling of loads, generators, or reactive compensation.

Dynamic MVA Calculation: The `mva` value automatically updates when `mw` or `mvar` change.

Example Usage

```
# Create load load1 = Load( name="Load-1", bus1_name="Bus-2", mw=50.0, mvar=30.0 ) # Access computed MVA print(load1.mva) # √(50² + 30²) = 58.31 MVA # Update power (MVA automatically recalculates) load1.mw = 60.0 print(load1.mva) # √(60² + 30²) = 67.08 MVA
```

Test Summary

6 Tests Included:

- ✓ Basic mva computation (3-4-5 triangle: mva=5.0)
- ✓ mva updates when mw changes (mw=6.0 updates mva correctly)
- ✓ mva updates when mvar changes (mvar=8.0 updates mva correctly)
- ✓ Empty name rejection (raises ValueError)
- ✓ Empty bus1_name rejection (raises ValueError)
- ✓ Zero load handling (mva=0.0 when mw=0, mvar=0)

5. Generator Class

Purpose

The `Generator` class represents a generator connected to a bus with specified operating setpoints. It models the active power output and optional voltage magnitude control.

Attributes

`name` *(str)*

Name of the generator (non-empty string).

`bus_name` *(str)*

Name of the connected bus (non-empty string).

`mw_setpoint` *(float)*

Active power setpoint in megawatts. Must be a finite number.

`v_setpoint` *(float | None)*

Voltage magnitude setpoint in per-unit. Optional parameter that must be positive when provided.

Key Features

Optional Voltage Control: The `v_setpoint` parameter is optional, allowing generators to be modeled with or without voltage regulation.

Strict Validation: Enforces finite values for `mw_setpoint` and positive values for `v_setpoint` when provided.

Type Safety: Includes a custom `_as_float()` method that rejects boolean values and ensures proper numeric types.

Flexible String Representation: The `__str__()` method adapts output based on whether voltage setpoint is defined.

Example Usage

```
# Create generator with voltage control gen1 = Generator(
name="G1", bus_name="Bus-1", mw_setpoint=100.0, v_setpoint=1.04
) # Create generator without voltage control gen2 = Generator(
name="G2", bus_name="Bus-3", mw_setpoint=50.0 ) # Set voltage
control later gen2.v_setpoint = 1.02 # Access setpoints
print(gen1.mw_setpoint) # 100.0 print(gen1.v_setpoint) # 1.04
```

Test Summary

10 Tests Included:

- ✓ Basic creation with all parameters (name, bus, MW, voltage)
- ✓ Optional v_setpoint (can be None, can be set later)
- ✓ __repr__ contains all field values
- ✓ __str__ produces human-readable format with units
- ✓ Empty name rejection (raises ValueError)
- ✓ Empty bus_name rejection (raises ValueError)
- ✓ Invalid mw_setpoint type (string raises TypeError)
- ✓ Infinite mw_setpoint rejection (raises ValueError)
- ✓ Negative v_setpoint rejection (raises ValueError)
- ✓ Setter validation enforcement (all setters enforce rules)

Milestone 1: Power System Equipment Classes

Documentation generated for Python implementation

These classes provide the structural foundation for constructing network matrices and performing power system simulations.