Bryce Aniszewski
Brian Janas
Ganpath Karl

Dr. Kerestes, Jack Carnovale,
Advanced Power Systems Analysis, University of Pittsburgh

Work In Progress until mid-April 2025

Project Documentation for APSA Project 2

Overview of Project

Design a full-scale power systems simulator using Python 3 and PowerWorld.

Our circuit:

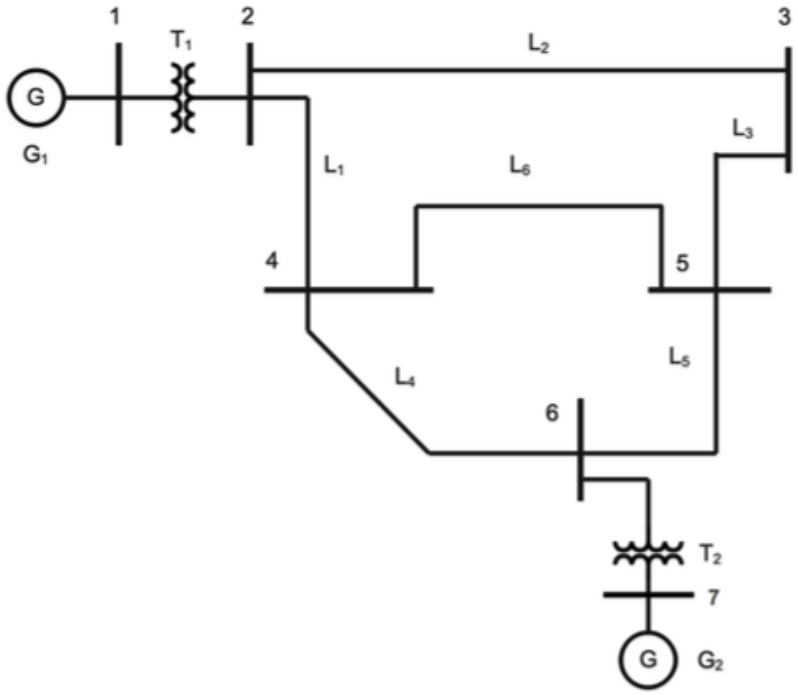

Figure 1: Seven Bus Power System

Class List and explanations

Add diagram of circuit

Class Definition: The Conductor class mode ls the physical and electrical characteristics of a conductor used in transmission lines.

Constructor (__init__ method): Initializes the Conductor object with the given parameters:

name: Name of the conductor.
diam: Diameter of the conductor (in inches).
GMR: Geometric Mean Radius (in feet).
resistance: Resistance of the conductor (in ohms per mile).
ampacity: Ampacity of the conductor (in amperes).
Attributes: The constructor initializes the attributes name, diam, GMR, resistance, and ampacity with the provided values.
String Representation (__str__ method): Returns a string representation of the Conductor object, which is useful for debugging and logging.

Conductor Class
The Conductor class models the physical and electrical characteristics of a conductor used in transmission lines.

```
class Conductor:
    """
    The Conductor class models the physical and electrical characteristics of a conductor used in transmission lines.
    """

    def __init__(self, name, diam, GMR, resistance, ampacity):
        """
        Initialize the Conductor object with the given parameters.

        :param name: Name of the conductor
        :param diam: Diameter of the conductor (in inches)
        :param GMR: Geometric Mean Radius (in feet)
```

```
        :param resistance: Resistance of the conductor (in ohms per mile)
        :param ampacity: Ampacity of the conductor (in amperes)
        """
        self.name = name  # Name of the conductor
        self.diam = diam  # Diameter of the conductor
        self.GMR = GMR  # Geometric Mean Radius of the conductor
        self.resistance = resistance  # Electrical resistance of the conductor
        self.ampacity = ampacity  # Maximum current carrying capacity of the conductor

    def __str__(self):
        """
        Return a string representation of the Conductor object.
        """
        return f"Conductor(name={self.name}, diam={self.diam}, GMR={self.GMR},
resistance={self.resistance}, ampacity={self.ampacity})"

# Example usage:
# conductor1 = Conductor("Partridge", 0.642, 0.0217, 0.385, 460)
# print(conductor1)
```

Explanation:

Class Definition: The Conductor class models the physical and electrical characteristics of a conductor used in transmission lines.

Constructor (__init__ method): Initializes the Conductor object with the given parameters:

name: Name of the conductor.

diam: Diameter of the conductor (in inches).

GMR: Geometric Mean Radius (in feet).

resistance: Resistance of the conductor (in ohms per mile).

ampacity: Ampacity of the conductor (in amperes).

Attributes: The constructor initializes the attributes name, diam, GMR, resistance, and ampacity with the provided values.

String Representation (__str__ method): Returns a string representation of the Conductor object, which is useful for debugging and logging.

Transmission Line Class

The TransmissionLine class models a transmission line connecting two buses in a power system. This class uses the Conductor and Geometry subclasses to determine its electrical characteristics.

```
class TransmissionLine:
    """
    The TransmissionLine class models a transmission line connecting two buses in a power
system.
    This class uses the Conductor and Geometry subclasses to determine its electrical
characteristics.
    """
```

```
    def __init__(self, name, bus1, bus2, bundle, geometry, length):
        """
        Initialize the TransmissionLine object with the given parameters.

        :param name: Name of the transmission line
        :param bus1: The first bus connected by the transmission line
        :param bus2: The second bus connected by the transmission line
        :param bundle: The bundle of conductors used in the transmission line
        :param geometry: The physical arrangement of conductors in the transmission line
        :param length: Length of the transmission line (in miles)
        """
        self.name = name  # Name of the transmission line
        self.bus1 = bus1  # The first bus connected by the transmission line
        self.bus2 = bus2  # The second bus connected by the transmission line
        self.bundle = bundle  # The bundle of conductors used in the transmission line
        self.geometry = geometry  # The physical arrangement of conductors in the
transmission line
        self.length = length  # Length of the transmission line

    def calc_base_values(self):
        """
        Calculate base impedance and admittance values for the transmission line.
        """
        # Placeholder for base impedance calculation (zbase)
        self.zbase = None  # Replace with actual calculation

        # Placeholder for base admittance calculation (ybase)
        self.ybase = None  # Replace with actual calculation

    def calc_admittances(self):
        """
        Calculate series impedance, shunt admittance, and series admittance for the
transmission line.
        """
        # Placeholder for series impedance calculation (zseries)
        self.zseries = None  # Replace with actual calculation

        # Placeholder for shunt admittance calculation (yshunt)
        self.yshunt = None  # Replace with actual calculation

        # Placeholder for series admittance calculation (yseries)
        self.yseries = None  # Replace with actual calculation
```

```
    def calc_yprim(self):
        """
        Calculate and populate the admittance matrix (yprim) for the transmission line.
        """
        # Placeholder for admittance matrix calculation (yprim)
        self.yprim = None  # Replace with actual calculation

    def __str__(self):
        """
        Return a string representation of the TransmissionLine object.
        """
        return f"TransmissionLine(name={self.name}, bus1={self.bus1}, bus2={self.bus2},
length={self.length})"

# Example usage:
# Assuming Bus, Bundle, and Geometry classes are defined elsewhere
# bus1 = Bus("Bus 1", 20)
# bus2 = Bus("Bus 2", 230)
# bundle1 = Bundle("Bundle 1", 2, 1.5, conductor1)
# geometry1 = Geometry("Geometry 1", 0, 0, 18.5, 0, 37, 0)
# line1 = TransmissionLine("Line 1", bus1, bus2, bundle1, geometry1, 10)
# print(line1)
```

Explanation:
Class Definition: The TransmissionLine class models a transmission line connecting two buses in a power system. It uses the Conductor and Geometry subclasses to determine its electrical characteristics.
Constructor (__init__ method): Initializes the TransmissionLine object with the given parameters:
name: Name of the transmission line.
bus1: The first bus connected by the transmission line.
bus2: The second bus connected by the transmission line.
bundle: The bundle of conductors used in the transmission line.
geometry: The physical arrangement of conductors in the transmission line.
length: Length of the transmission line (in miles).
Attributes: The constructor initializes the attributes name, bus1, bus2, bundle, geometry, and length with the provided values.
Base Values Calculation (calc_base_values method): Placeholder method to calculate base impedance and admittance values for the transmission line.
Admittances Calculation (calc_admittances method): Placeholder method to calculate series impedance, shunt admittance, and series admittance for the transmission line.
Admittance Matrix Calculation (calc_yprim method): Placeholder method to calculate and populate the admittance matrix for the transmission line.

String Representation (__str__ method): Returns a string representation of the TransmissionLine object, which is useful for debugging and logging.

Bus Class

The Bus class models a bus in a power system. Each bus has a name and a nominal voltage level.

```
class Bus:
    """
    The Bus class models a bus in a power system.
    Each bus has a name and a nominal voltage level.
    """

    # Class variable to keep count of bus instances
    bus_count = 0

    def __init__(self, name, base_kv):
        """
        Initialize the Bus object with the given parameters.

        :param name: Name of the bus
        :param base_kv: Nominal voltage level of the bus (in kV)
        """
        self.name = name  # Name of the bus
        self.base_kv = base_kv  # Nominal voltage level of the bus
        self.index = Bus.bus_count  # Unique index for the bus

        # Increment the bus count for each new instance
        Bus.bus_count += 1

    def __str__(self):
        """
        Return a string representation of the Bus object.
        """
        return f"Bus(name={self.name}, base_kv={self.base_kv}, index={self.index})"

# Example usage:
# bus1 = Bus("Bus 1", 20)
# bus2 = Bus("Bus 2", 230)
# print(bus1)
# print(bus2)
```