Advanced Power Systems Simulator

Language: Python

Group Members: Luke Barnett, Jacqueline Dillon, Ryan Sieber

Section 1: Object Classes

These Classes are used to represent various elements of power systems and their mathematical/physical properties.

**Bus Class**

This class creates a Bus object. Busses refer to any connection in a power system that has a specific nodal voltage related to its connections to other electrical components in the system

Syntax:

*Setting bus data:*

setbusdata(self, name, type: str, real\_P: float, Q\_or\_V: float):

Example:

MainGrid.setBusData("Bus1", "Slack Bus", 0, 0)

|  |  |  |
| --- | --- | --- |
| **Input Argument Name** | **Description** | **Value in example** |
| Name | String name reference to generator | “Bus1” |
| type | Bus type, can be slack, load, or voltage controlled | “Slack Bus” |
| Real\_P | The bus’s real power | 0 |
| Q or V | Either the bus’s per unit voltage V, or its reactive power Q depending on the bus type | 0 |

**Class Code:** located in repository as Bus.py



**Generator Class**

This class creates a Generator object. Generators are used in power systems to control input voltage levels.

**Syntax:**

*Defining a generator object:*

Generator(name, bus1, nominalpower, x1gen, x2gen, x0gen,

grounding\_type, grounding\_value)

Example:

newGenerator = Generator("G1", "Bus1", 100, 0.12, 0.14, 0.05, "Solid ground", 0)

|  |  |  |
| --- | --- | --- |
| **Input Argument Name** | **Description** | **Value in example** |
| Name | String name reference to generator | “G1” |
| Bus1 | String reference to the node bus connection’s name | “Bus1” |
| Nominal power | Generator’s output power under standard test conditions | 100 MVA |
| X1gen | Generator’s per unit positive sequence impedance | 0.12 |
| X2gen | Generator’s per unit negative sequence impedance | 0.14 |
| X0gen | Generator’s per unit zero sequence impedance | 0.05 |
| Grounding\_type | Generator’s grounding connection type | “Solid Ground” |
| Grounding\_value | Generator’s grounding connection value | 0 Ω |

**Class Code:** located in repository as Generator.py

Graphical user interface

Description automatically generated with low confidence

**Transformer Class**

This class creates a transformer object. Transformers are used in power systems to transfer energy from one circuit to another, often stepping up or down this voltage.

**Syntax:**

*Defining a transformer object:*

Transformer(name, bus1, bus2, apparentpower, v1rated, v2rated, impedance, xrratio,

Sbase, Zt\_connection1, Zt\_grounding1, Zt\_value1, Zt\_connection2, Zt\_grounding2, Zt\_value2)

Example:

newTransformer = Transformer("T1", "Bus1", "Bus2", 125, 20, 230, 0.085, 10, 100,

"Delta", "N/A", 0, "Grounded Wye", "Resistor", 1)

|  |  |  |
| --- | --- | --- |
| **Input Argument Name** | **Description** | **Value in example** |
| Name | String name reference to transformer | “T1” |
| Bus1 | String reference to the first node bus connection’s name | “Bus1” |
| Bus2 | String reference to the second node bus connection’s name | “Bus2” |
| Apparentpowerrating | The transformers’ rated apparent power, in MVA | 125 MVA |
| V1rated | The transformers’ primary side rated voltage, in kV | 20 kV |
| V2rated | The transformers’ secondary side rated voltage, in kV | 230 kV |
| Impedance | The transformers’ impedance, in pu | 0.085 |
| Xrratio | The transfromers’ reactance to resistance ratio | 10 |
| Sbase | The power systems base apparent power, in MVA | 100 MVA |
| Zt\_connection1 | The primary side connection type, “Delta” or “Grounded Wye” | “Delta” |
| Zt\_grounding1 | Grounding connection on primary side | “N/A” |
| Zt\_value1 | Value of transformer’s grounding connection on primary side | 0 Ω |
| Zt\_connection2 | The secondary side connection type, “Delta” or “Grounded Wye” | “Grounded Wye” |
| Zt\_grounding2 | Grounding connection on secondary side | “Resistor” |
| Zt\_value2 | Value of transformer’s grounding connection on secondary side | 1 Ω |

**Other values stored in this object class:**

|  |  |
| --- | --- |
| **Variable reference name** | **Description** |
| self.Rpu | The calculated per unit resistance of the transformer |
| self.Xpu | The calculated per unit reactance of the transformer |
| self.powerloss | Power loss value is stored when store\_power\_loss function is referenced, used elsewhere in the simulator |

**Relevant Equations:**

**Class Code:** located in repository as Transformer.py

Graphical user interface, text, application, email

Description automatically generated

**Transmission Line Class**

This class creates a Transmission Line object. Transmission Lines carry power from one area/component of a circuit to another.

*Defining a Transmission Line object:*

TransmissionLine((self, name: str, bus1: str, bus2: str, lengthmi: float,

axaxis: float, ayaxis: float, bxaxis: float, byaxis: float, cxaxis: float, cyaxis: float,

codeword: str, numberofbundles: int, seperationdistance: float, Vbase: float, Sbase: float):

Example:

newLine = TransmissionLine ("L1", "Bus2", "Bus4", 10, 0, 0, 19.5, 0, 39, 0,

"Partridge", 2, 1.5)

|  |  |  |
| --- | --- | --- |
| **Input Argument Name** | **Description** | **Value in example** |
| Name | String name reference to transformer | “T1” |
| Bus1 | String reference to the first node bus connection’s name | “Bus1” |
| Bus2 | String reference to the second node bus connection’s name | “Bus2” |
| Lengthmi | Length of the line in miles, float value | 10 |
| Axaxis | Phase a’s spatial location, x coordinate | 0 |
| Ayaxis | Phase a’s spatial location, y coordinate | 0 |
| Bxaxis | Phase b’s spatial location, x coordinate | 19.5 |
| Byaxis | Phase b’s spatial location, y coordinate | 0 |
| Cxaxis | Phase c’s spatial location, x coordinate | 39 |
| Cyaxis | Phase c’s spatial location, y coordinate | 0 |
| Codeword | The string referring to the type of material the transmission line is composed of. Corresponds to specific physical characteristics associated with this material. | “Partridge” |
| Numberofbundles | The amount of conductors in the bundle, from 1 to 4 usually | 2 |
| separationdistance | The amount of space between the conductor bundles, in ft | 1.5 ft |

TransmissionLine is the parent to the TransmissionLineBundles class

This class assigns the appropriate Geometric mean radius, conductor radius r, and resistance depending on the codeword. For the sake of this simulator, the only codeword used is “Partridge” and it assigns the following values:

|  |  |
| --- | --- |
| Variable | Value Assigned based on codeword “Partridge” |
| Self.GMR | 0.2604 |
| Self.r | 0.321 |
| Self.resistancepermi | 0.385 |

TransmissionLineBundles calculates the following physical characteristics:

|  |  |  |
| --- | --- | --- |
| **Variable reference in code** | **Description** | **Equation used (for 2 bundles)** |
| Self.DSL | Equivalent inductance, depending on number of bundles |  |
| Self.DSC | Equivalent capacitance, depending on number of bundles |  |
| Self.R | Equivalent Resistance, depending on number of bundles |  |

TransmissionLineBundles Code:

Text

Description automatically generated

**Relevant Equations:**

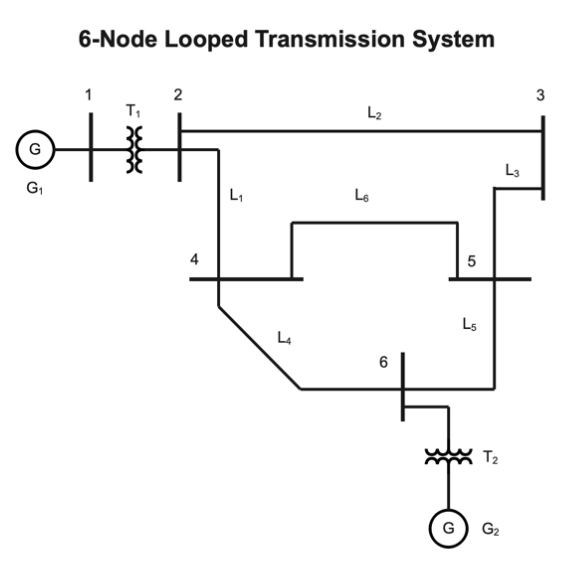
**Class Code:** located in repository as TransmissionLine.pyGraphical user interface, text, application, Word

Description automatically generated

Section 2: Simulation Classes

**Grid Class**

This class physically lays out the power grid that is being analyzed in the simulation. For this specific simulation, the following grid was used



|  |  |
| --- | --- |
| **Variable reference in code** | **Description** |
| self.Ybus | Empty 7x7 admittance matrix that is populated using the below equations in Calculate\_Ybus function |
| self.Rpu | Per unit resistance imported from either the Transformer or Transmission Line class, depending on bus connections |
| self.Xpu | Per unit reactance imported from either the Transformer or Transmission Line class, depending on bus connections |
| self.Bpu | Per unit susceptance imported from the Transmission Line class |

**Relevant Equations:**

Non-diagonal elements:

Diagonal elements:

, where i is the ith transmission line from k to n.

**Class Code:** located in repository as Grid.py

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated Graphical user interface, application, Word

Description automatically generated

**Newton\_Raphson\_Power\_Flow Class**

This class implements the Newton-Raphson Power Flow Solver, which solves the power flow equations for an AC power system using the Newton-Raphson method. This method calculates the Jacobian and power mismatches through multiple iterations until the maximum power mismatch fits the convergence criteria. This class also calculates the maximum reactive power and adds a capacitor bank of the limit is exceeded. Further, this class calculates current in each line and ensures no values exceed the max ampacity. It also calculates the power flowing through each line, power losses in each line, and the total power losses of the system. Finally, this class outputs the real and reactive power injected into the slack bus and the reactive power injected into the PV bus.

|  |  |
| --- | --- |
| **Variable reference in code** | **Description** |
| P\_mismatch | Real power mismatches |
| Q\_mismatch | Reactive power mismatches |
| mismatchPQ | Combined array of Real and Reactive power mismatches |
| J11 | Top left quadrant of Jacobian matrix |
| J12 | Top right quadrant of Jacobian matrix |
| J21 | Bottom left quadrant of Jacobian matrix |
| J22 | Bottom right quadrant of Jacobian matrix |
| self.V\_complex | Array for the complex voltage values |
| self.I\_values | Array of the current values in each line |
| self.P\_inj\_slack | Real power injected into slack bus |
| self.Q\_inj\_slack | Reactive power injected into slack bus |
| self.Q\_inj\_PV | Reactive power injected into PV bus |
| Self.Vbase | Base voltage for the system, given by v2rated of 1st transformer |
| powerloss | Total power loss of the system, calculated using every element within of transformer and transmission line dictionaries |

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Value** |
| self.convergencevalue | Maximum mismatchPQ value at which Newton-Raphson converges | 0.0001 |
| V | Initial voltage array for a flat start | [1 1 1 1 1 1 1] |
| delta | Initial phase angle array for a flat start | [0 0 0 0 0 0 0] |
| self.Q\_k\_limit | The maximum allowable reactive power in the system | 175 MVAR |

**Relevant Equations:**

**Power Mismatches:**

**Jacobian Off Diagonal Elements** **Jacobian Diagonal Elements**

**where i is the ith iteration**

Current Calculations

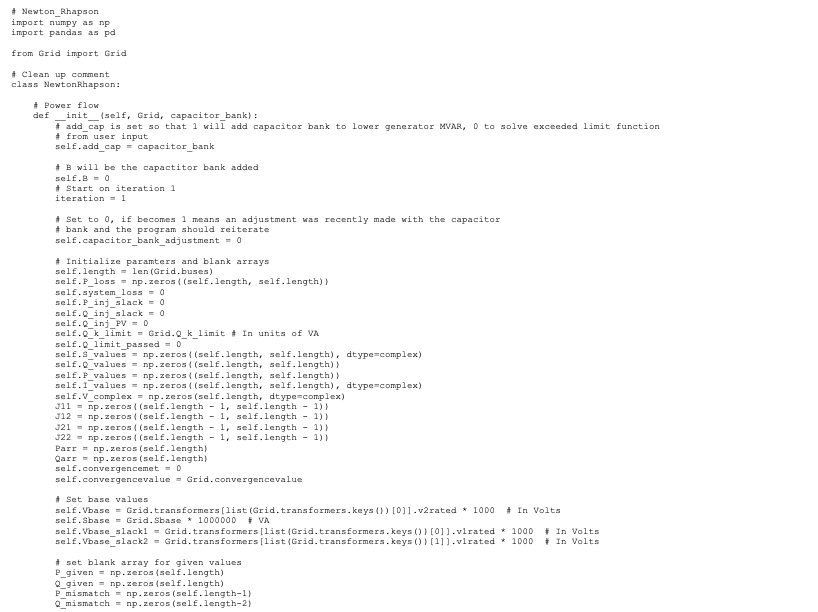
Power Losses

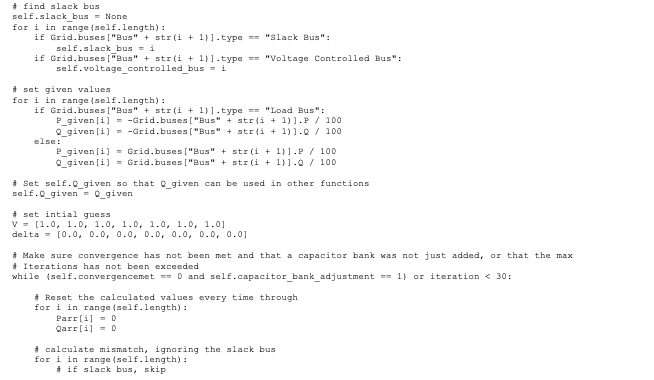
Calculated for each element and summed within code

Power Injections

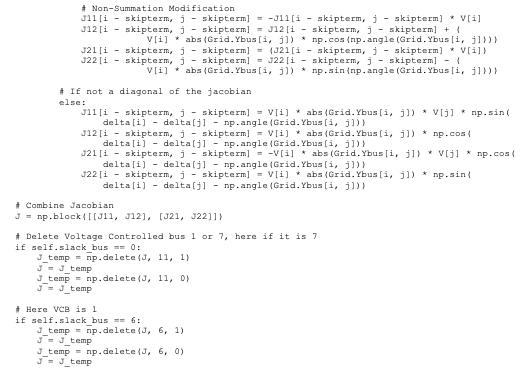
where VC bus is voltage-controlled bus

**Class Code:** located in repository as Newton\_Raphson\_Power\_Flow.py

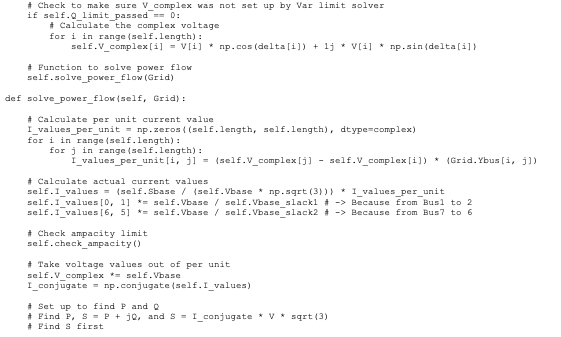


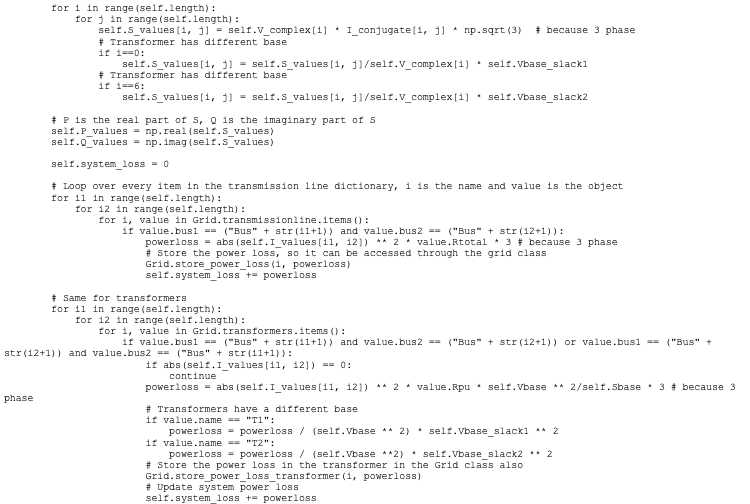






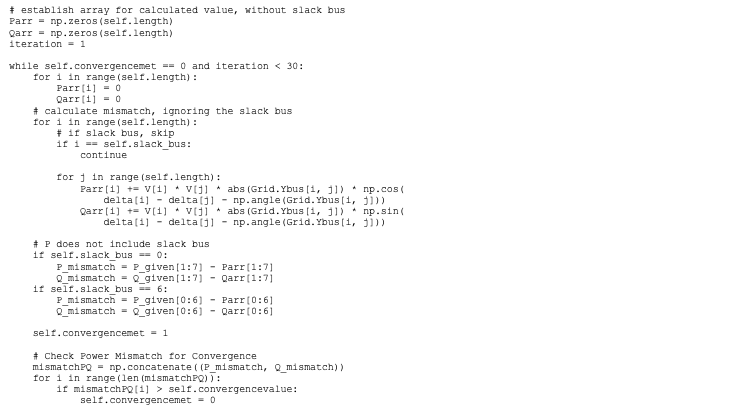


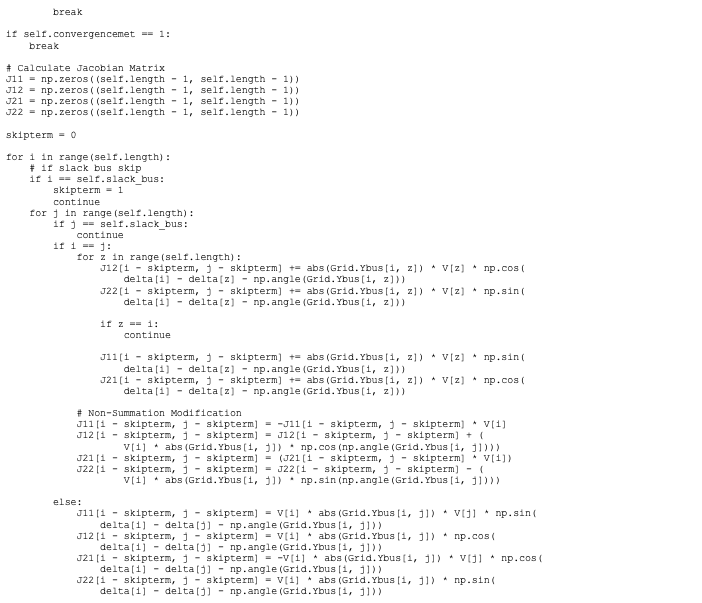


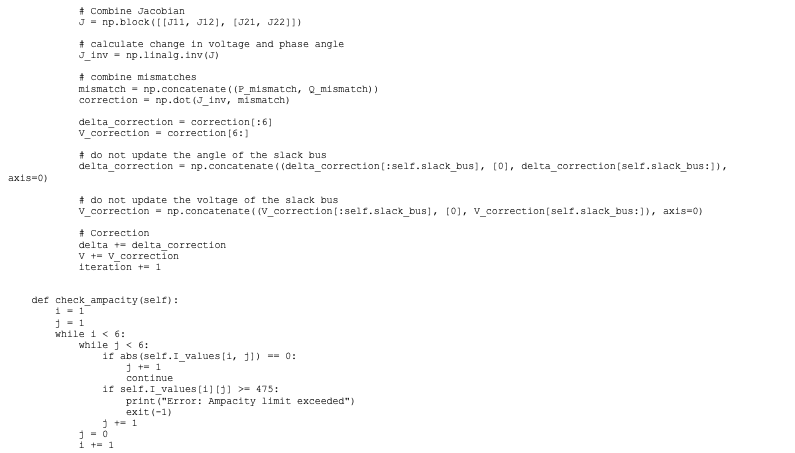










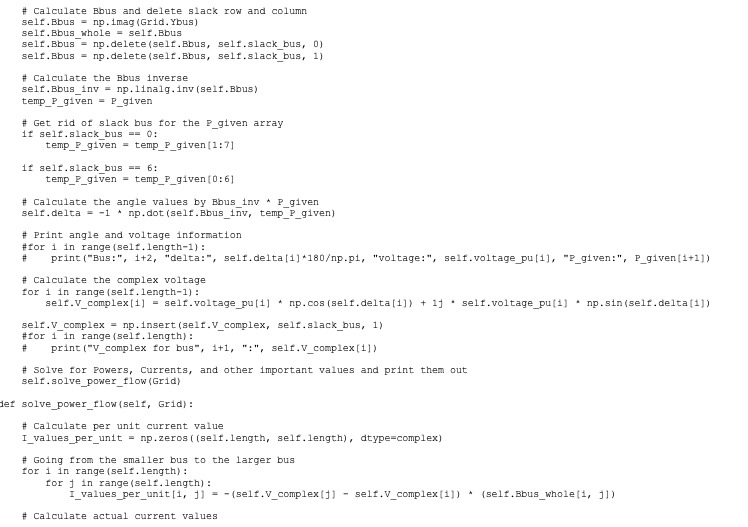


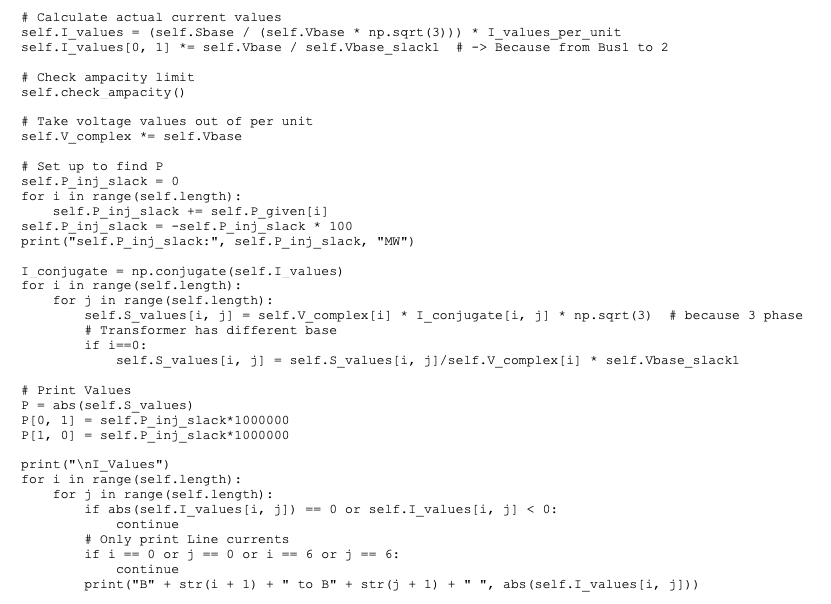
**DC\_Power\_Flow\_Solver Class**

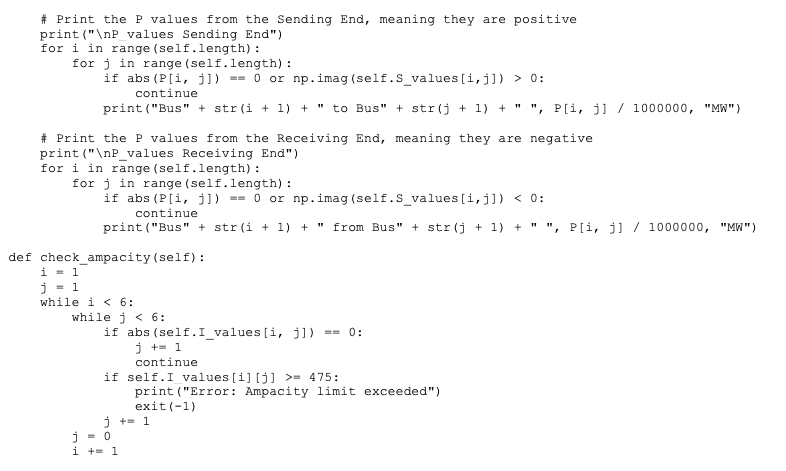
This class solves the power flow equations for a DC power system.

**Class Code:** located in repository as DC\_Power\_Flow\_Solver.py









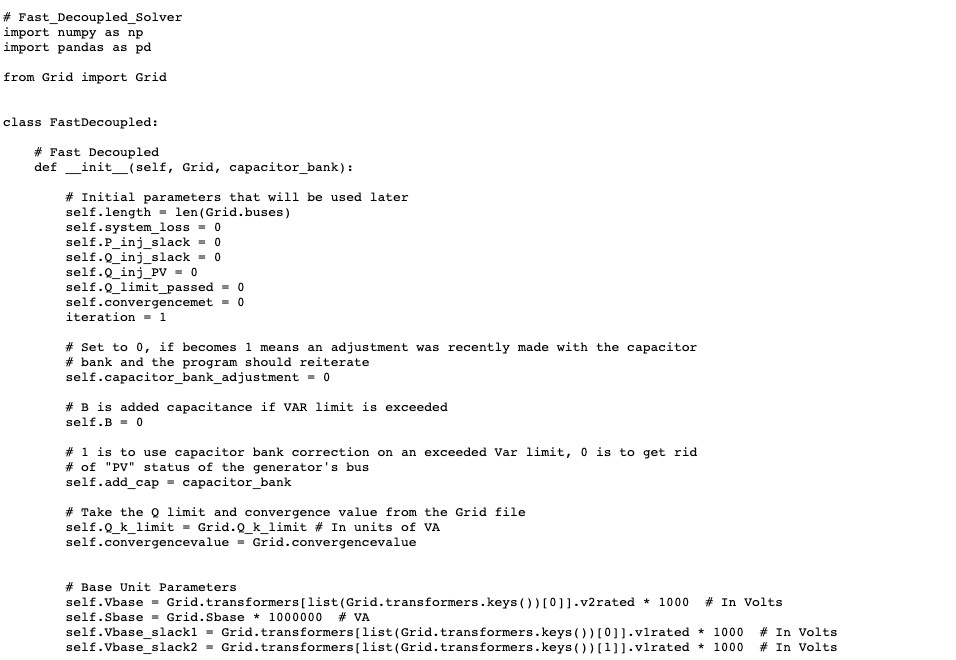
**Fast\_Decoupled\_Solver Class**

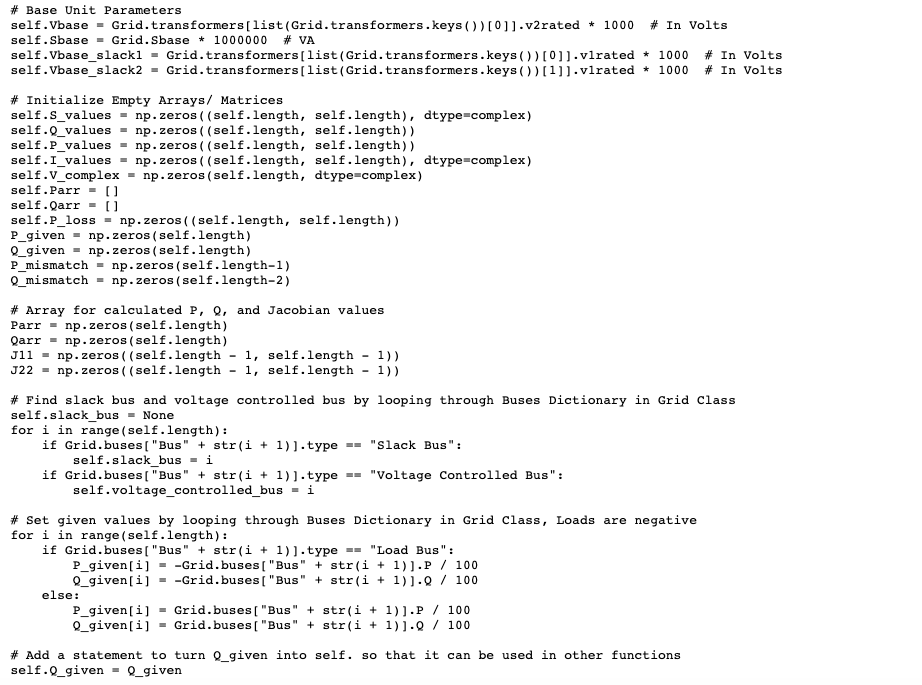
This class solves the power flow equations for an AC power system using the fast decoupled method.

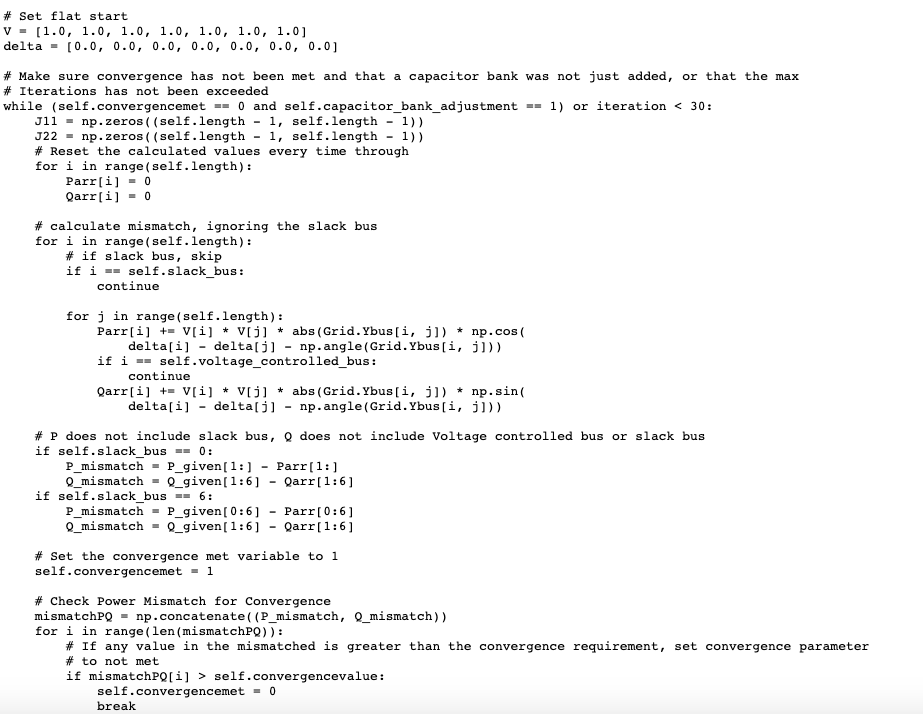
**Relevant Equations:**

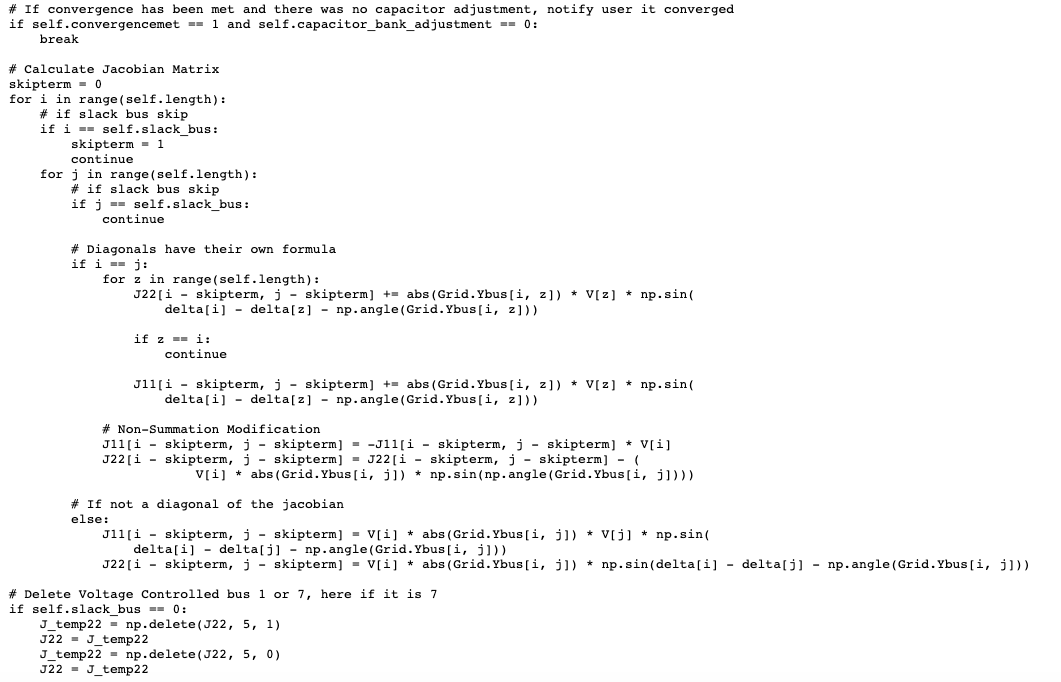
For additional reference, the J indexes refer to the Jacobian matrix output for the Power system in question. Visually:

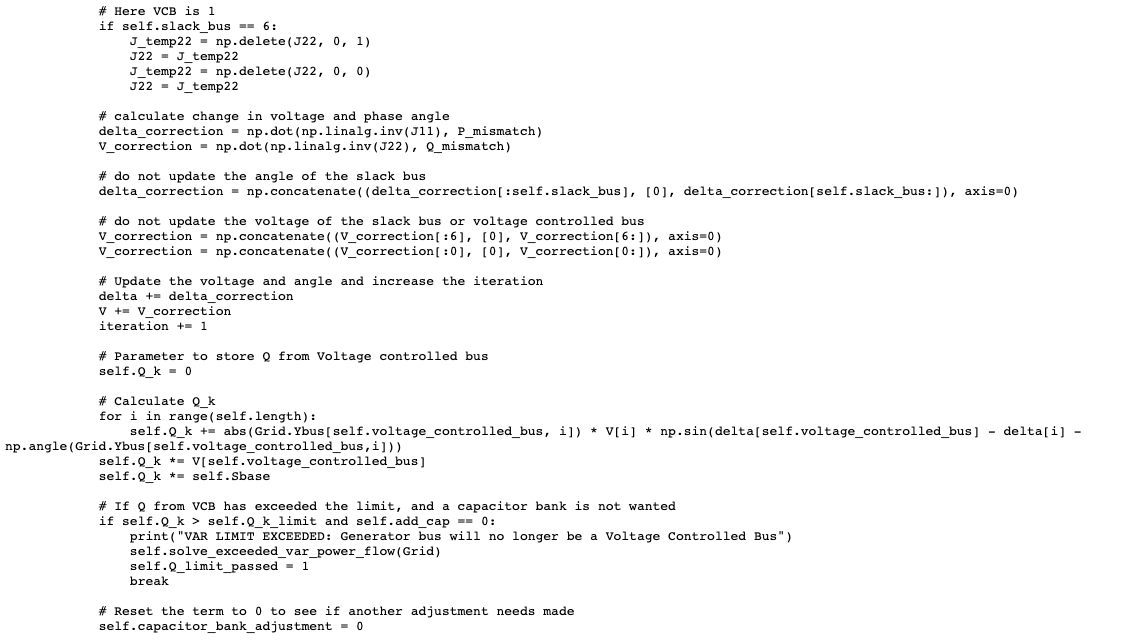
**Class Code:** located in repository as Fast\_Decoupled\_Solver**.**py

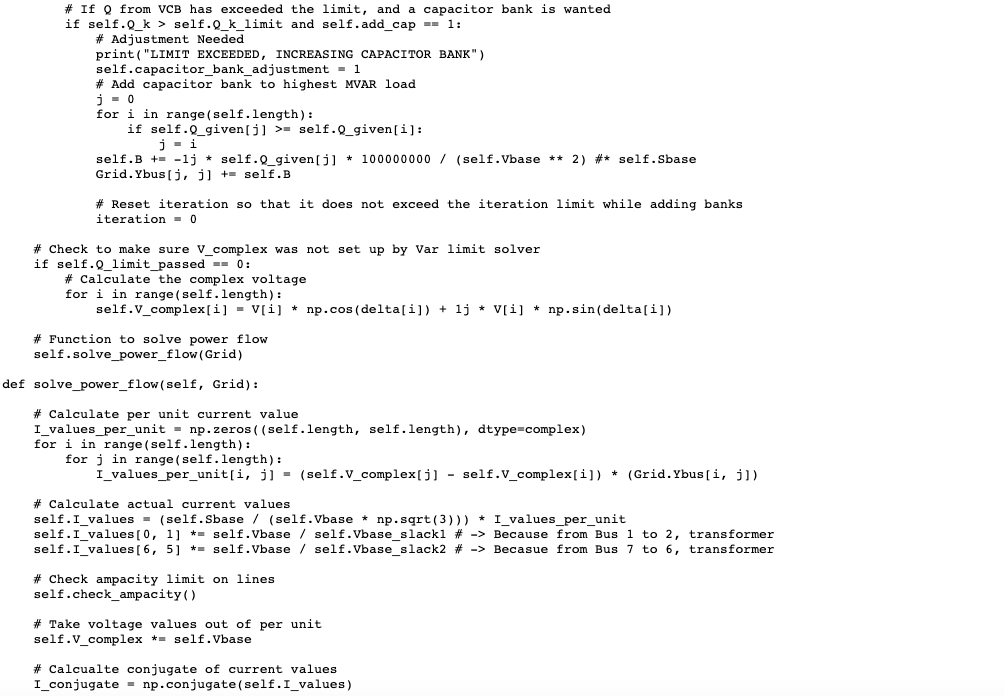


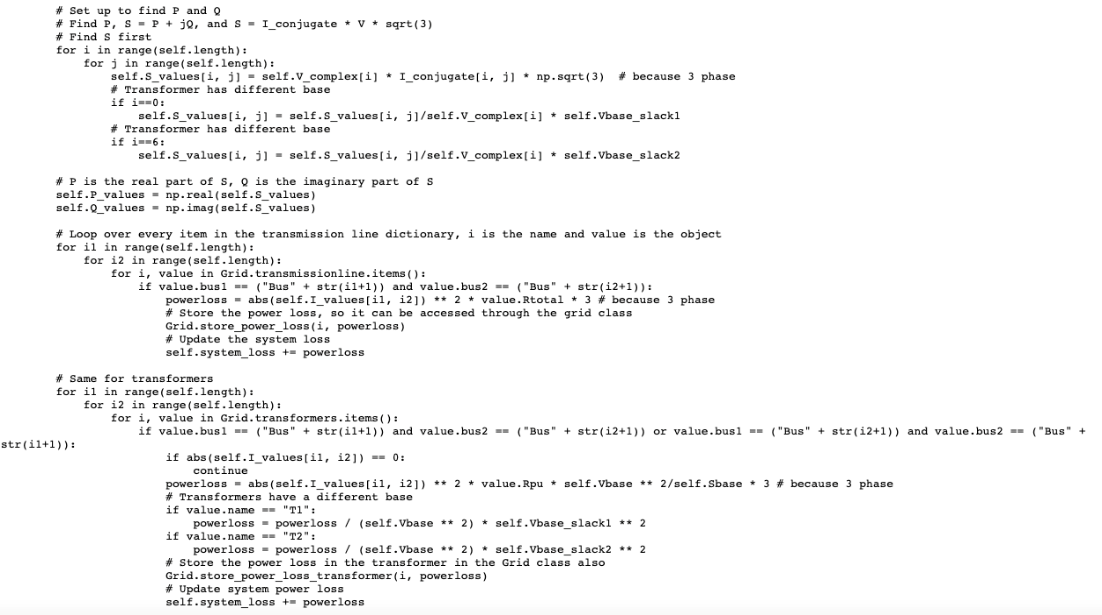


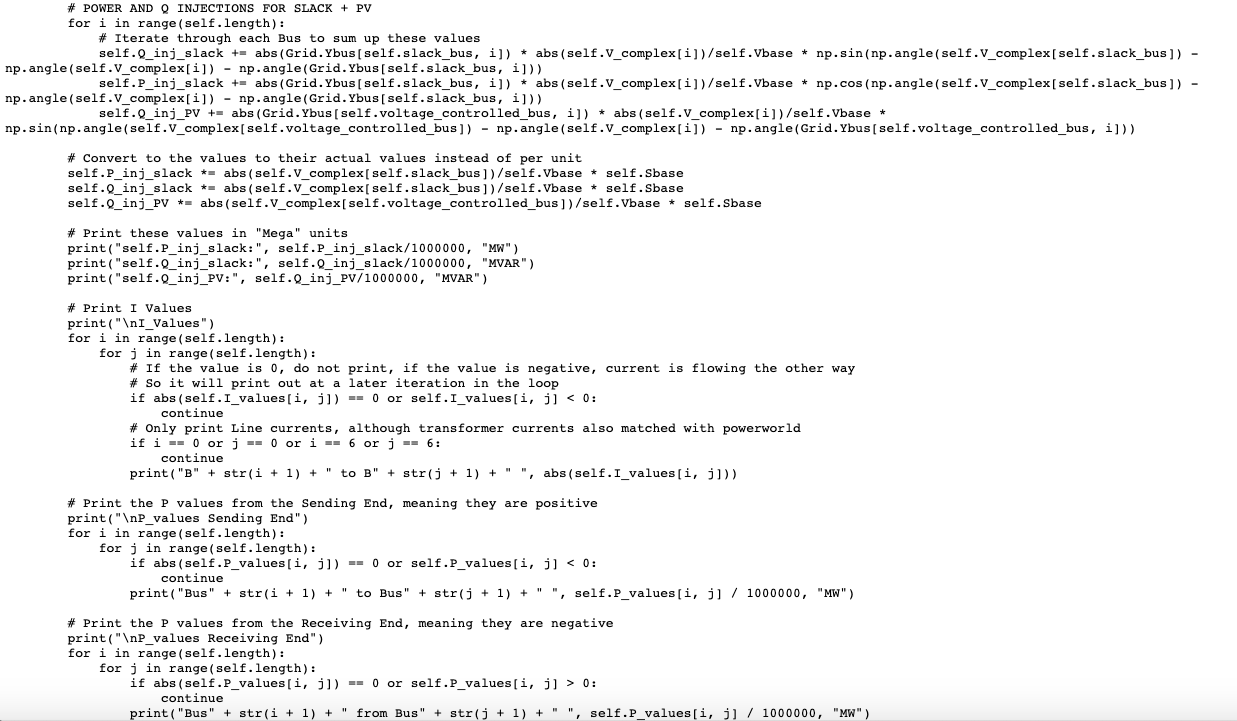


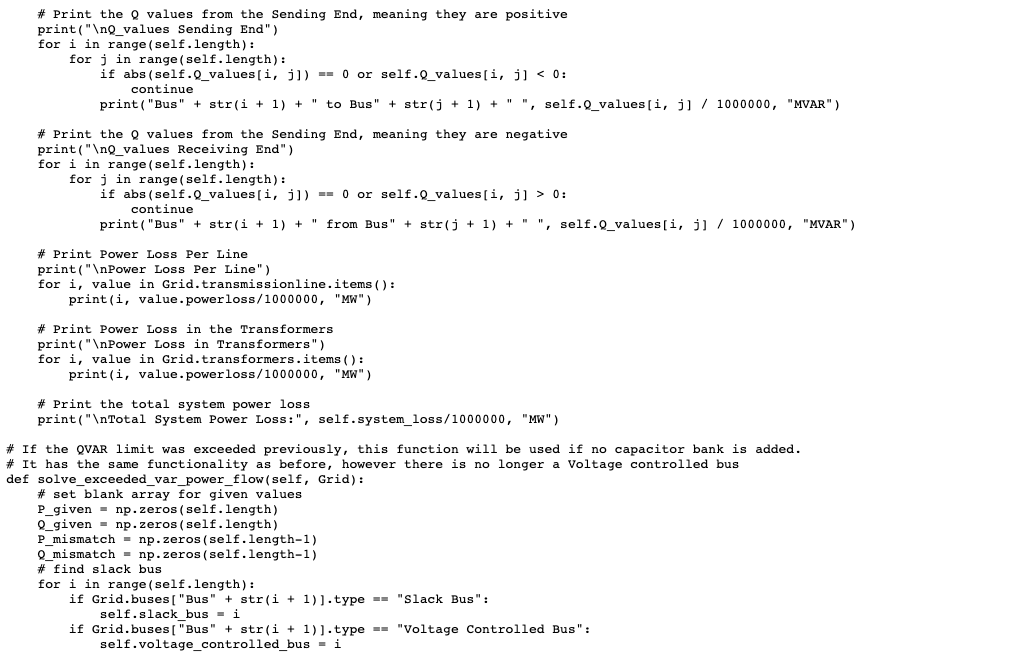


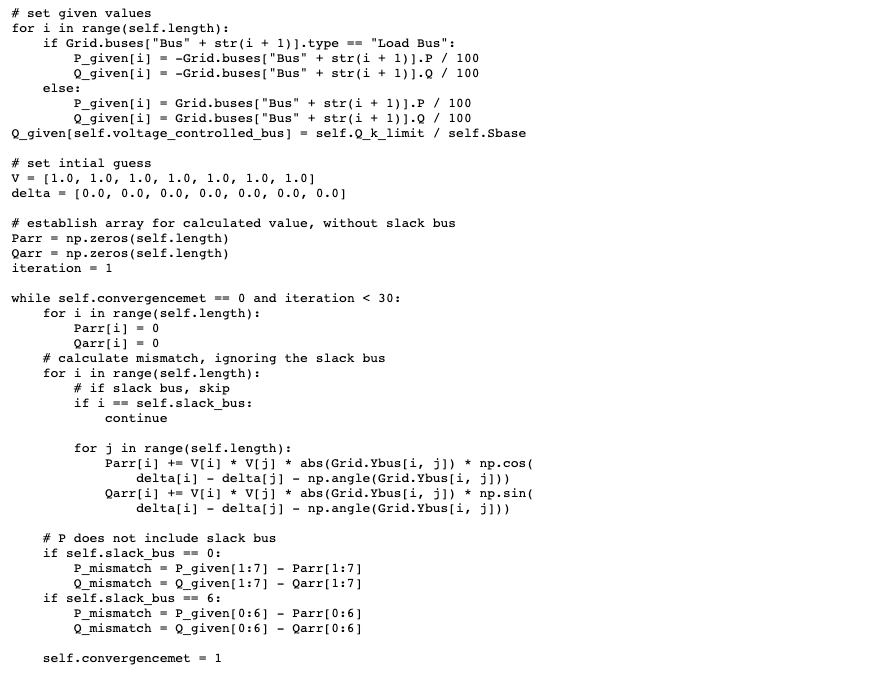


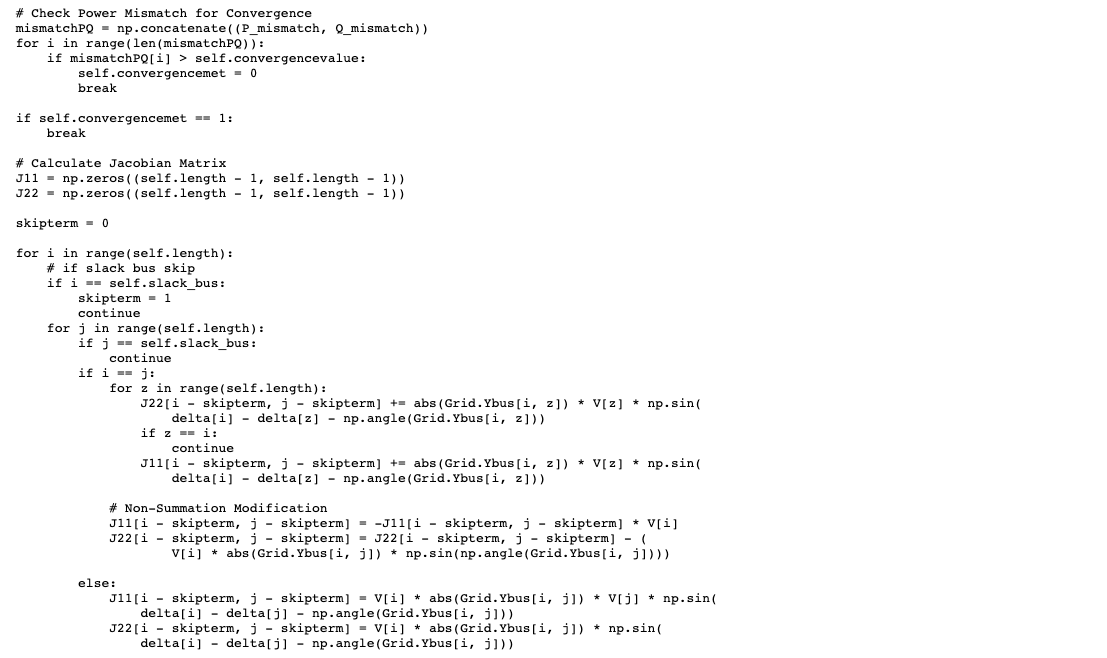


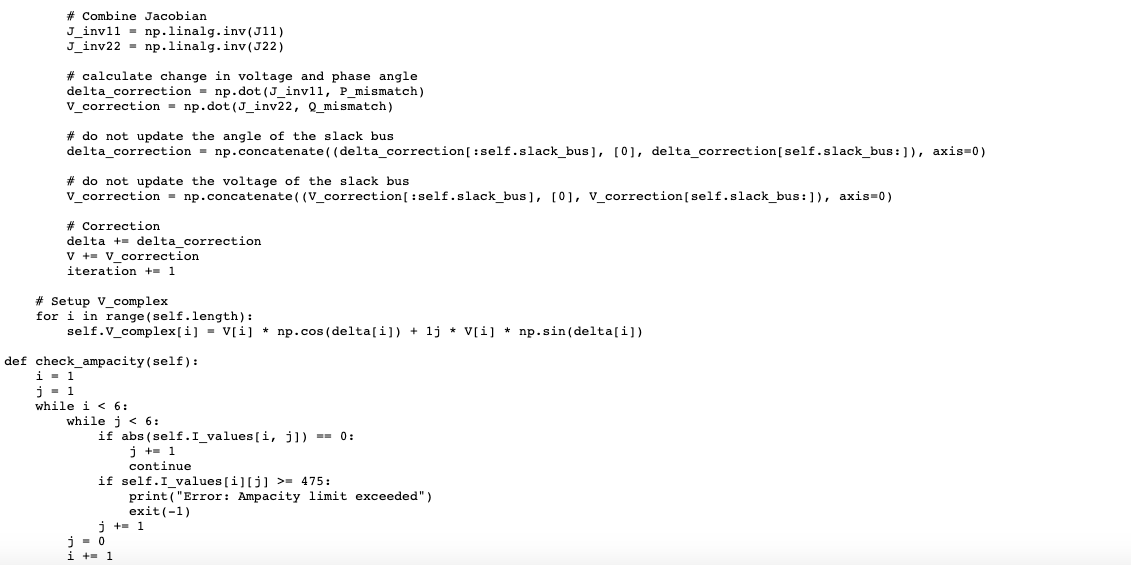












**Sequence\_Networks Class**

This class calculates the positive, negative, and zero sequence impedances in the power system. The positive sequence and negative bus admittance matrices are practically the same as the Ybus calculated in the grid class, but each add their own generator impedance values. The zero sequence matrices depend on both the generator and transformer groundings, and our code allows for user input of any grounding type and still calculates the zero sequence networks.

|  |  |
| --- | --- |
| **Variable reference in code** | **Description** |
| self.Zbus0 | Zero sequence bus impedance matrix |
| self.Zbus1 | Positive sequence bus impedance matrix |
| self.Zbus2 | Negative sequence bus impedance matrix |
| self.Ybus0 | Zero sequence bus admittance matrix |
| self.Ybus1 | Positive sequence bus admittance matrix |
| self.Ybus2 | Negative sequence bus admittance matrix |
| self.x1generators\_newpu1 | The per unit impedance value for generator one in the positive sequence network |
| self.x1generators\_newpu2 | The per unit impedance value for generator two in the positive sequence network |
| self.x2generators\_newpu1 | The per unit impedance value for generator one in the negative sequence network |
| self.x2generators\_newpu2 | The per unit impedance value for generator two in the negative sequence network |
| self.x0generators\_newpu1 | The per unit impedance value for generator one in the zero sequence network |
| self.x0generators\_newpu2 | The per unit impedance value for generator two in the zero sequence network |

|  |  |  |
| --- | --- | --- |
| **Input Argument Name** | **Description** | **Value in example** |
| self.Zg1 | Generator 1 grounding value | 0 |
| self.Zg2 | Generator 2 grounding value | 1 Ω |
| self.Zt1\_value1 | Transformer 1 side 1 grounding value | None |
| self.Zt1\_value2 | Transformer 1 side 2 grounding value | 1 Ω |
| self.Zt2\_value1 | Transformer 2 side 1 grounding value | None |
| self.Zt2\_value2 | Transformer 2 side 2 grounding value | inf |

**Relevant Equations:**

Positive Sequence

Negative Sequence

Zero Sequence Impedances

Lines:

Generators:

Transformers:

Grounded Wye/Grounded Wye

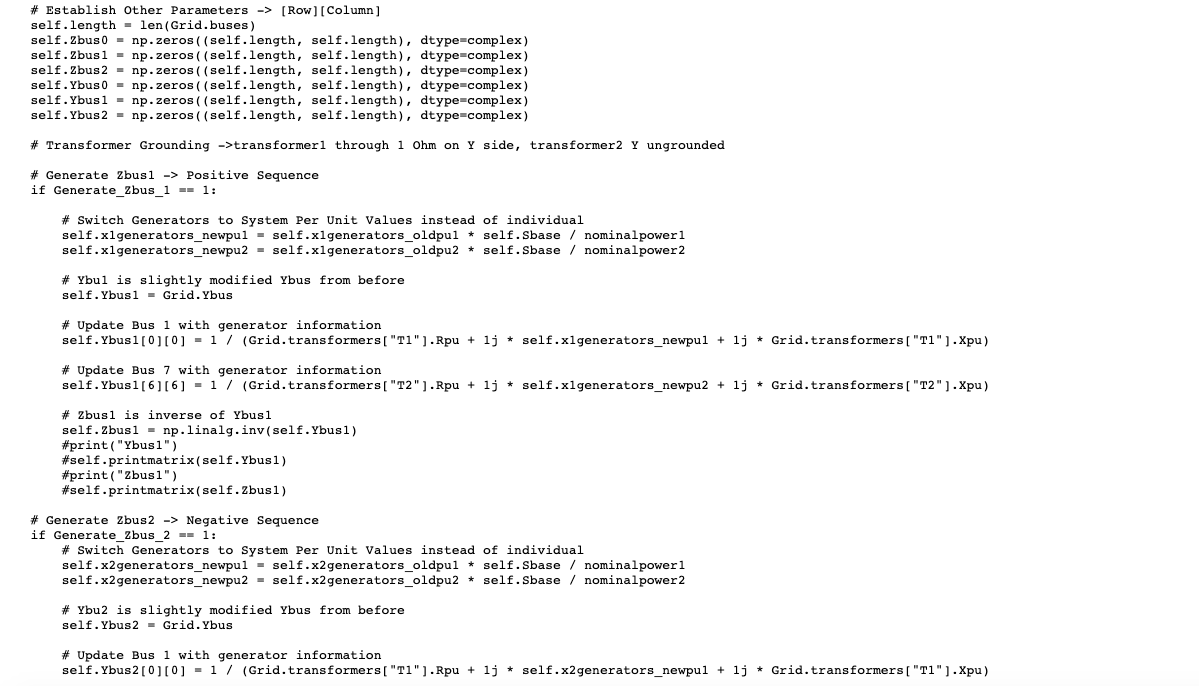
Delta / Grounded Wye

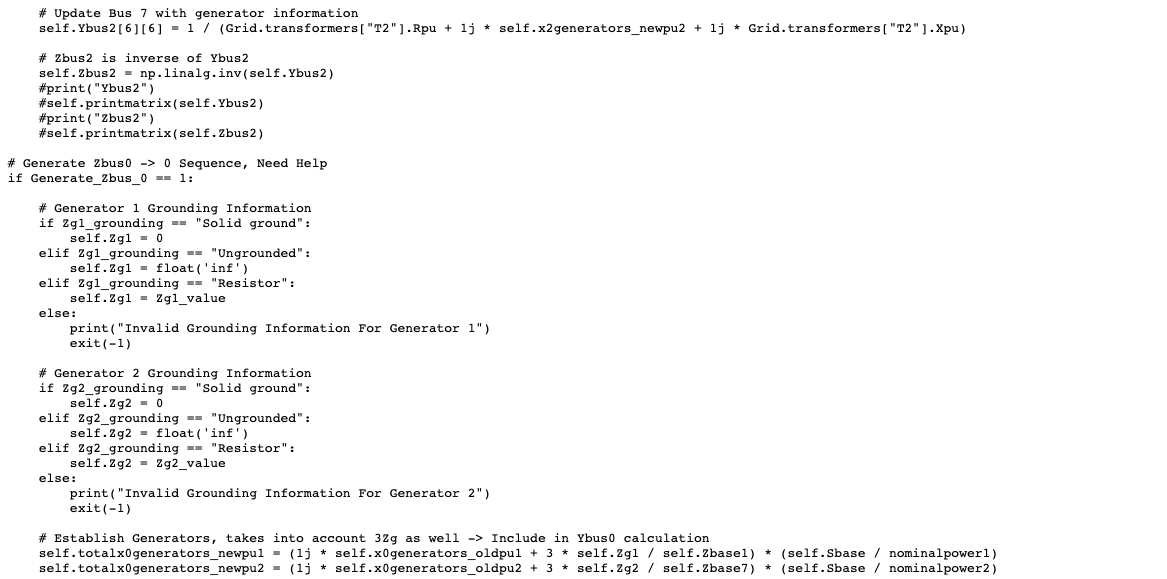
Ungrounded Wye / Grounded Wye

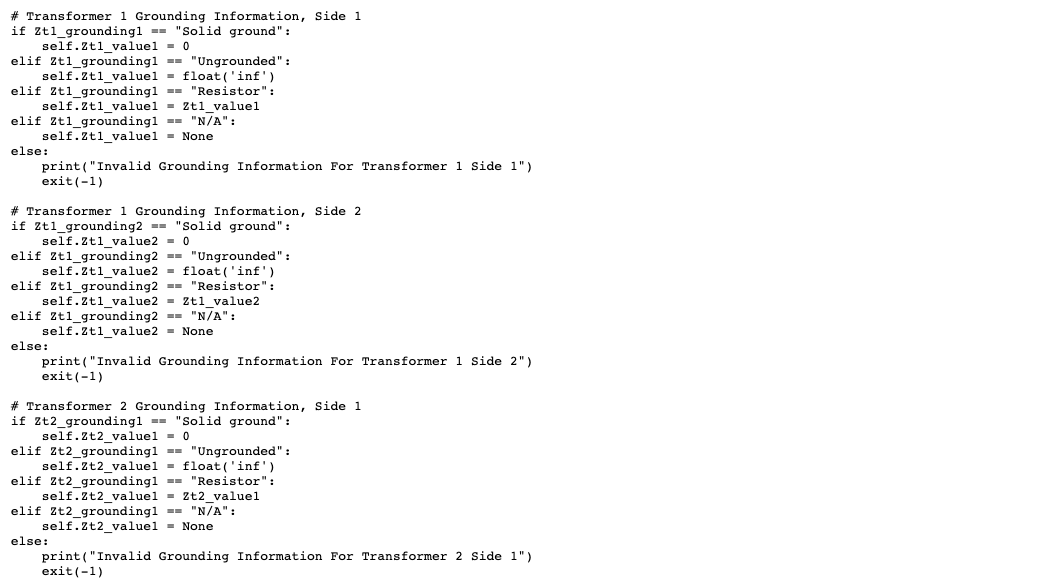
Sum additional impedances connected to each bus, and invert for zero admittance matrices or vice versa.

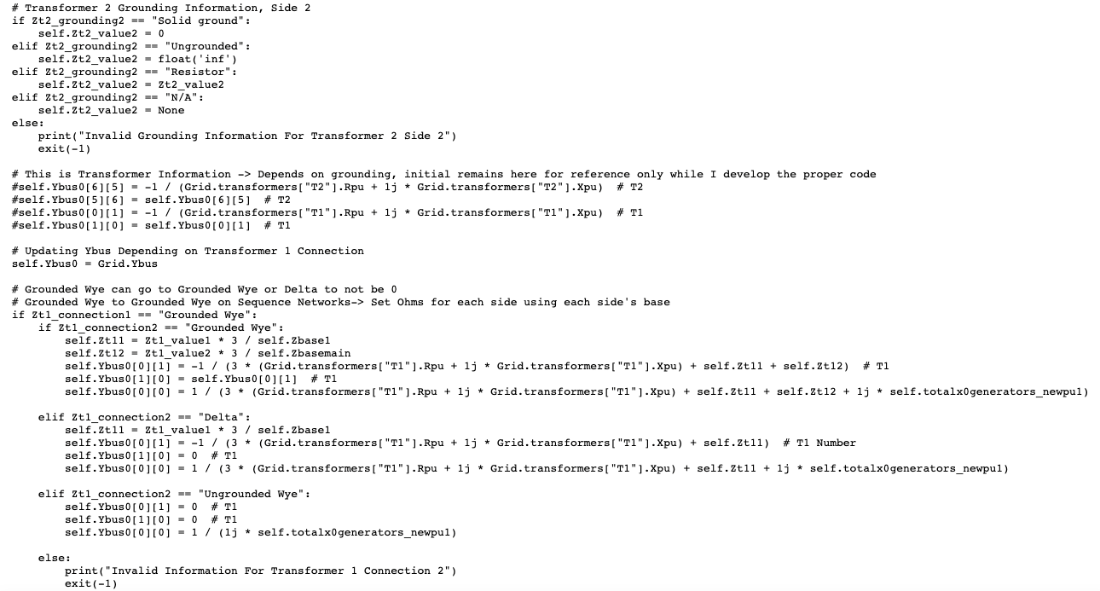
**Class code:** located in repository as Sequence\_Networks.py

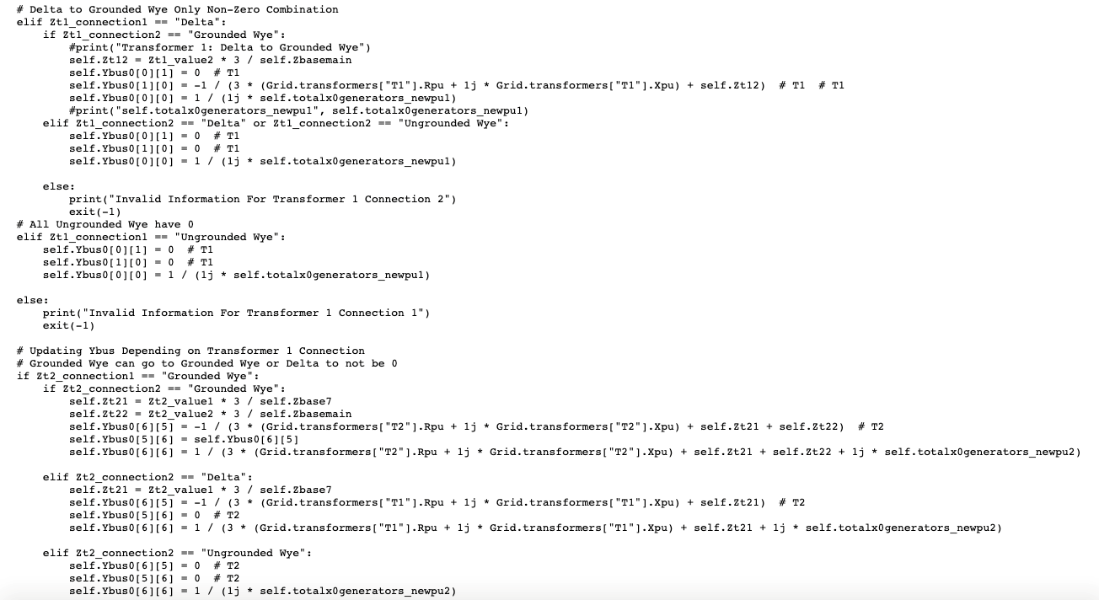




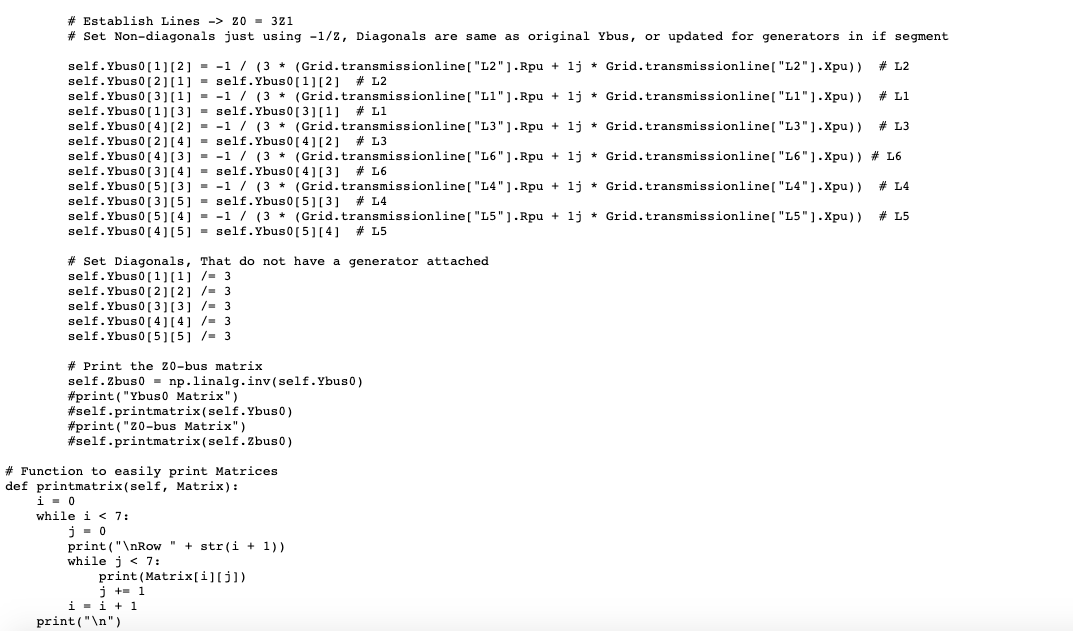












**Fault\_Calculation Class**

This class calculates fault currents and voltages in the power system. When the main function is run, the user is asked if they would like to perform a fault calculation. If they respond with YES, the follow up question is regarding the type of fault analysis they would like performed. The options for the fault analysis include Symmetrical Fault, Single Line to Ground Fault, Line to Line Fault, Double Line to Ground Fault. The user then gets to choose the bus for the fault location and enter a faulting impedance if they so desire.

|  |  |
| --- | --- |
| **Variable reference in code** | **Description** |
| faulttype | Type of fault input by user |
| faultlocation | Bus number input by user where fault is located |
| faulting\_impedance | Optional user input for faulting impedance value |
| self.VF | Pre fault voltage |
| self.Zf | Faulting impedance (zero for bolted fault) |
| self.In\_1 | Positive sequence fault current |
| self.In\_2 | Negative sequence fault current |
| self.In\_0 | Zero sequence fault current |
| self.V\_1 | Positive sequence fault voltage |
| self.V\_2 | Negative sequence fault voltage |
| self.V\_0 | Zero sequence fault voltage |
| self.I\_012 | Array of all sequence fault currents |
| self.I\_abc | Three phase current component vector |
| self.V\_012 | Array of all sequence fault voltages |
| self.V\_abc | Three phase voltage component vector |
| self.Znn\_1 | Positive sequence fault impedance |
| self.Znn\_2 | Negative sequence fault impedance |
| self.Znn\_0 | Zero sequence fault impedance |
| self.A | Symmetrical Component Transform Matrix |

**Relevant Equations:**

Example: Single Line to Ground Fault

Different equations exist for each fault type, but the following generalized equations can be used to find for any fault type.

For any bus k:

**Class Code:** located in repository as Fault\_Calculation.py

