Cole Flory, Nick Genco, Lucas Villalba

Dr. Kerestes & Paulo Radatz

ECE 1774/2774 – Advanced Power System Analysis

28 April 2023

Software Documentation

**Milestone 1: Develop Power Flow System YBus Matrix**

Milestone 1 we are given transformer system ratings, winding connection configurations, transmission line conductor data, line lengths, and tower drawings. Using the information provided our software was used to calculate the positive sequence series impedance and shunt admittance matrices. The data given was then used to calculate the positive sequence series impedance and shunt admittance matrices. Using the data obtained and the one-line diagram we developed the positive sequence power flow system YBus Matrix.

**Milestone 2: Produce Power Flow Input Data, Jacobian, and Injection Equations**

Milestone 2 we are given loading and generator setpoint data which is then used to write up power flow input data which includes line data, transformer data, and bus data. The first four steps of the Newton-Raphson power flow program for a flat start for one iteration are then implemented which included calculating: power mismatches, Jacobian matrix, voltage and phase mismatches, and voltage and phase angles for the next iteration.

**Milestone 3: Solve Power System Power Flow**

Milestone 3 we used our Milestone 2 input power flow data to solve the Newton-Raphson algorithm. We calculated the line current, power on both the receiving and sending ends of the line, power losses in each line, and the total power losses for the entire system. The real and reactive power injected to the slack bus and the reactive power injected into the PV bus were also calculated.

**Milestone 4: Var Limits and Different Solvers**

Skipped

**Milestone 5: Sequency Networks**

Milestone 5 we included two neutral conductors to find the positive, negative, and zero sequence impedance for the lines. Both of our generators had equal impedances and were wye connected. We developed a positive, negative, and zero sequence bus impedance matrices for the sub transient reactance of the generators given.

**Mileston 6: Sequence Voltage & Current Calculations, Fault Calculations**

Calculates zero, positive, and negative sequence voltages and currents depending on the type of faults.

**Program Requirements:**

* Numpy
* Pandas
* Cmath

**Class Breakdown:**

System class adds the different elements to the system such as buses, generators, conductors, transformers, transmission lines, resistors, and loads.

Geometry class takes seven arguments including string name and three pairs of transmission line coordinates ax, ay, bx, by, cx, and cy. The constructor calculates the distance between each point and stores them as instance variables dab, dbc, and dca. It then calculates deq (equivalent diameter) by taking the square root of the product of the three instance variables. Four methods are then used to return the values of the instance variables.

Conductor class takes several arguments including the name, outerDiamaeter, geometric mean radius (gmr), and AC resistance (rAC) which is passed to bundle. Additionally, an ampacity (max amount of electric current a conductor can carry under certain conditions) can be specified. Three methods are then used to access the corresponding properties of the conductor object created.

Bundle class takes several arguments including the name, bundleSize, bundleDistance, and a Conductor object to represent the type of conductor used in the bundle. The constructor calculates the distance between the center of the bundle and the conductors dsl and dsc. The calculation performed depends on the size of the bundle. The class also has several methods to access properties of the bundle object and to access the bundle size and distance properties.

TransmissionLineData class takes in four arguments including the name, a Bundle object that contains the number and arrangement of conductors in the line, a Geometry object that contains the geometry of the line, and a Conductor object that contains information about the electrical properties of the conductor material. The class calculates equivalent diameter (deq) of the bundle, capacitance (cPrime), inductance (lPrime), resistance per mile (ohmsPerMile), and impedance per mile (zseriesperMile). The calculations performed depend on the input values of the bundle, the geometry, and the conductor.

TransmissionLine class takes in six arguments including the name, the two buses it connects (bus1 and bus2), a lineData object containing the data for the transmission line, and a bases object containing the base values for the system. The class calculates the admittance matrix calculates the admittance matrix (y) and voltage drop matrix (v) for the transmission line. The calculations are performed using the line data and base values.

TransformerData class takes seven arguments including the name, rated power (s\_rated), primary and secondary voltage (v\_primary and v\_secondary), the impedance of the transformer (zpu\_transformer), the xr\_ratio, and a bases object containing the base values for the system. The class calculates the zpu\_new and zpu\_phase values using the rated power and base values, the txRpu and txXpu values using the phase angle (xr\_ratio) and complex exponential calculations, and the admittance (txYpu) of the transformer using the complex conjucate of (txZpu). Three methods are then used to return the calculated admittance, resistance, and reactance values of the transformer in per-unit.

Transformer class takes in four arguments including name, the two buses it connects (bus1 and bus2), and an instance of the TransformerData class containing the transformer data. The class calculates the admittance matrix of the transformer and saves it to the y attribute as a Pandas DataFrame. The bus indices are the row and column labels, while the actual values are the admittances between the corresponding buses. The DataFrame populates its elements with the per unit admittance values according to the buses they are connected to. Since the matrix is a two-winding device, the matrix is symmetric. The diagonal elements are represented by the sum of all admittances connected to bus n, while the off-diagonal elements are represented by sum of the negative admittance connected from bus n to bus m.

Bus class has several attributes such as name, type, vk, vdf, delta1, pk, qk, and busCount which updates the number of buses created. The class has two methods, getBusTypeCount and calc\_vdf. The first method checks the type of bus and updates the class variables slackCount, type, load\_Count, and vc\_count accordingly. The second method calculates the voltage difference between two buses and measures the voltage drop and stores it in the vdf attribute of the bus.

BaseValues class initializes variables base power (pbase), base voltage (vbase), base impedance (zbase), and base admittance (ybase) using the given pbase and vbase arguments.

Constants class initializes electric constant (E0), meters in a mile (MILE), FREQ for a 60Hz system, and angular frequency (W).

YBusFormation class has several attributes such as name, ymatrix, numBuses, and bus\_order. The class also contains an object called System which contains information about the buses, branches, and generators within the system. The YBus Matrix is initialized with zeros before iterating over the network elements in the fillYbus method, which then adds the admittance values to the appropriate elements within the matrix.

Solution class uses the YbusFormation class and PowerFlow class to calculate the YBus Matrix and perform power flow analysis. The do\_power\_flow method is used to calculate the voltage and current values for the circuit, using Ohm’s law and the circuit equations. The print\_nodal\_voltages method is used to print the voltages at Bus A and Bus B, which are stores in the bus\_voltages list.

Main class creates a power system model including buses, transmission lines, transformers, loads, and a slack bus. Each element of the system is defined with its own characteristics including its geometry, conductor, resistance, and reactance. The power flow for the system is calculated using the information defined within the Solution Class.

PowerFlow class implements an algorithm based on the Newton-Raphson method, which iteratively solves nonlinear equations of power flow. There are several steps involved in solving the algorithm. The first step finds the y array, which contains the starting power values. The second step finds the x array, which contains the flat start voltage. The third step is to find the fx array, which contains the active and reactive power flows for each bus. The fourth step finds the change in power for each bus. The class also calculates the Jacobian matrix which is used to calculate the voltage and power flow across the network. The methods create sub-matrices of the Jacobian matrix based on the partial derivative of the active and reactive power injections with respect to the voltage magnitudes and angles. The sub-matrices are then combined to form the full Jacobian matrix. The class also solves any mismatches in the power flow solution by adjusting the solution vector iteratively until the mismatches converge to zero. The Newton-Raphson algorithm iteratively solves voltage magnitude and phase angles at each bus until convergence is reached. The code checks whether the absolute value of delta y is less than the specified tolerance, if so, the method exists, otherwise it forms the Jacobian matrix and solves for power mismatch variables. Electrical quantities such as voltage drops, line currents, power losses, and reactive power at each bus and transmission line are also calculated. It checks if any current in the transmission line is over the rated values and updates accordingly. The system is originally an array of zeros before iterating over all elements in the dictionary and making the necessary calculations. Depending on the type of bus and transmission line it updates the necessary variables to find the sum of the power losses in the end.

Generator class calculates the YBus Matrix for the generator in each of the three sequence networks. The YBus Matrix represents the admittance between the nodes in the power system and is used in power flow and fault analysis. The class also has methods for getting the name, voltage, and bus of the generator.

Load class keeps track of the load’s name, the bus connected to it, the power rating, and the voltage rating. It also calculates the admittance based on its resistance.

Resistor class keeps track of the resistor’s name, the bus connected to it, and its resistance. It also calculates the admittance matrix.

ZbusFormation class takes the power system and YBus Matrix parameters and forms the ZBus Matrix, which is computed from the inverse of the YBus Matrix. The ZBus Matrix is used ot compute the bus voltages and power flows within the system.

VoltageMatrix class is used to calculate the voltage difference between buses. The class iterates through each element and identifies the buses attached to the element and calculates the voltage difference between those buses. If the buses are the same, the diagonal element of the matrix is set to the voltage magnitude and phase angle of the bus. If they are different, the off-diagonal element is set to the voltage difference between the buses.

SequenceNetwork class constructs the sequence voltage components for the power system. The class initializes the transformation matrix then iterates through each bus in the System object and calculates the complex phase voltage at that bus. It then uses the transformation matrix to convert it to the sequence voltage components. For each bus in the network the complex phase voltage and phase angle are calculated. The resulting sequence voltages represent the zero, positive, and negative sequence components.

Circuit class is represented using a dictionary of buses and their corresponding elements. Each bus has a unique name and order to which they are added to the circuit. The class also has a count of the total components and circuits created.

DisplayResults class is designed to display the results of a power flow analysis performed on a power system. The class takes two inputs System and Solution. System is an object class containing information about the power system such as buses, generators, transformers, and transmission lines. Solution is also an object class containing the solution of the power flow analysis. The class initializes the System and Solution attributes of the class. The class also provides a printed message describing the objective and milestones for the project for which the code was designed. Furthermore, it provides instructions to the user about the elements that can be added to the power system and the order in which they should be added. It also prompts the user to reorder the buses before displaying the results. If the user chooses to do so, the method reorders the Ybus, Zbus, powerflow\_X, and powerflow\_Y data according to the new order of the buses entered by the user. The Ybus, Ybus1, Ybus2, Ybus0, Zbus, Zbus1, Zbus2, Zbus0, powerflow\_X, powerflow\_Y, JacobIndex, and Jacobian data frames are then generated based on the System and Solution inputs. The method then prints the formatted versions of the data frames for the user to analyze the results of the power flow analysis.

**Limitations & Considerations:**

* Y-Bus coded to specific components needing to be connected to specific buses.
* High voltage rating on one transformer must equal high voltage rating on second transformer.

**Verification of Results:**

Diagram, schematic

Description automatically generated

Graphical user interface

Description automatically generated

Scatter chart

Description automatically generated with medium confidence

Graphical user interface, application, table

Description automatically generated