Jack Carnovale & Aiden Landis

Power Flow Simulator: Documentation

This document serves to show how to use Solution, PowerSystem, and other classes in order to conduct power systems studies. It will go into all methods and variables accessible by a user of each class.

Solution:

Class to hold all solution related operations and data. Solution options include 3 different PowerFlow Algorithms, as well as fault analysis calculations.

Variables:

- sys
  - PowerSystem variable that holds all system details including present states of and parameters associated with bus voltage, angle, transformers, lines, loads, generators
- tolerance
  - Maximum per unit error threshold on system power for solving Power Flow
- maxIterations
  - Maximum allowed iterations for a Power Flow solutions before divergence assumed

Methods:

- setTolerance(self, tolerance)
  - Sets the maximum per unit tolerance of power mismatch for a solution
  - Ex: soln1.setTolerance(0.001) sets the tolerance for solving to 0.001 per unit of the system power base
- setMaxIterations(self,maxIter)
  - Sets the maximum allowed iterations for a power flow solution
  - maxIter: New value for maxIterations
  - Ex: soln1.setMaxIterations(50) sets the maximum allowed number of iterations to 50 iterations
- solveNewtonRaphson(self)
  - Solves and sets the PowerSystem bus angles and voltages using the Newton-Raphson technique
  - Displays all voltages, angles, power, and currents associated with the system using Pandas formatted output. See example output below

```
Newton Raphson Solver:
Newton Raphson Converged in 3 iterations!
System Parameters:
     Voltage (kV)  Angle (deg)
Bus
1           1.000        0.000
2           0.935       -4.447
3           0.917       -5.636
4           0.927       -4.755
5           0.923       -4.909
6           0.938       -3.886
7           1.000        2.219


Bus Powers:
      Real (MW)  Reactive (Mvar)
Bus
1    116.003955        88.802750
2     -0.000143         0.000090
3   -109.999819       -50.000005
4    -99.999884       -70.000002
5    -99.999883       -64.999989
6     -0.000172         0.000098
7    199.999974       112.553811
```

```
Line Currents and Angles:
      Current (A)  Angle (deg)  From Bus  To Bus
Line
1         177.684      -46.637       4.0     2.0
2         199.358      -32.490       3.0     2.0
3         126.926      -19.292       5.0     3.0
4         167.833      -25.871       6.0     4.0
5         413.089      -29.138       6.0     5.0
6          22.937      -42.921       5.0     4.0

Transmission Line Ampacity:
      % Ampacity
Line
1      19.313499
2      21.669382
3      13.796291
4      18.242717
5      44.901015
6       2.493182

Transmission Line Power Losses:
      Losses (kW)
Line
1      182.326407
2      573.800203
3      186.072571
4      325.339425
5      985.462191
6       10.634175
```

```
Transformer Power Losses:
             Losses (kW)
Transformer
1            1444.110932
2            2296.280909

Total Line Losses: 2263.635 kW
Total Transformer Losses: 3740.392 kW
Total System Losses: 6004.027 kW

Process finished with exit code 0
```

- solveFastDecoupled(self)
    - Finds the fast decoupled Newton Raphson solution to the power system
    - Outputs the same information as above
- solveDCPowerFlow(self)
    - Sets the system to a flat start and finds the DC Power Flow solution to the system
    - Outputs the following information

```
DC Power Flow Solver:
System Parameters:
     Voltage (kV)  Angle (deg)  Power (MW)
Bus
1             1.0        0.000  113.646008
2             1.0       -4.437    0.356088
3             1.0       -5.721 -110.436485
4             1.0       -4.818 -100.702315
5             1.0       -4.994 -100.442153
6             1.0       -3.880    1.430234
7             1.0        2.157  200.507374

Line Currents and Angles:
      Current (A)  Angle (deg)  From Bus  To Bus
Line
1         125.421       11.250       4.0     2.0
2         169.164       10.798       3.0     2.0
3         119.820       10.520       5.0     3.0
4         154.411       11.528       6.0     4.0
5         366.674       11.440       6.0     5.0
6          16.530       10.972       5.0     4.0
```

- checkMvarLims(self, yGuess)
    - Function used by Newton Raphson and fast decoupled power flow solutions to make sure that the Mvar limits on generators are not exceeded
    - yGuess: the calculated real and reactive power values for a power flow solution

    - Returns true and sets the generator bus to PQ if the Mvar limits were exceeded on a generator
- getJacobian(self)
    - Finds and returns the jacobian of the power system in its present state of bus voltages and angles
    - Returns the jacobian in a numpy matrix
- getJ1(self)
    - Finds and returns the first quadrant of the jacobian for Fast Decoupled analysis
    - Returns the quadrant in a numpy matrix
- getJ4(self)
    - Finds and returns the fourth quadrant of the jacobian for Fast Decoupled analysis
    - Returns the quadrant in a numpy matrix
- updateBuses(self, deltaX,usedBuses)
    - Based on the change in voltages and angles calculated in the power flow iteration, this function updates the bus voltages and angles
    - deltaX: numpy vector of changes to be made to bus angles and voltages
    - usedBuses: numpy matrix which corresponds to the buses that need to be updated in the first column. The second column is whether that update is to the angle (0) or the voltage (1) of the bus. This is a product of reducing the jacobian in the Newton Raphson
- print_fault_v_i(self, index1, Vabc, faultcurrentsabc)
    - For a fault at bus at index1, prints all bus voltages and the current at the faulted bus
    - Index1: Bus index in the power system bus_order list
    - Vabc: NumBuses x 6 numpy matrix that holds all bus voltages (per unit) and angles (degrees) for all phases
    - Iabc: Numpy matrix of all magnitudes and phases for the current on every phase at the faulted bus
- solveSymmetricalFault(self, index1, Vf)
    - Finds the voltages and currents from symmetrical fault at bus at index1, given a prefault voltage of Vf per unit.
    - Prints output using print_fault_v_i()
    - Index1: Bus index for fault
    - Vf: Prefault voltage in per unit
- solveLineToLineFault(self, index1, Vf, Zf)
    - Finds the voltages and currents from a line to line fault at bus at index1, given a prefault voltage of Vf per unit and a fault impedance of Zf ohms.
    - Prints output using print_fault_v_i()
    - Index1: Bus index for fault
    - Vf: Prefault voltage in per unit
    - Zf: Complex fault impedance in ohms
- solveLineToGroundFault(self, index1, Vf, Zf)

        - Finds the voltages and currents from a line to ground fault at bus at index1, given a prefault voltage of Vf per unit and a fault impedance of Zf ohms.
        - Prints output using print_fault_v_i()
        - Index1: Bus index for fault
        - Vf: Prefault voltage in per unit
        - Zf: Complex fault impedance in ohms
    - solveDoubleLineToGroundFault(self, index1, Vf, Zf)
        - Finds the voltages and currents from a double line to ground fault at bus at index1, given a prefault voltage of Vf per unit and a fault impedance of Zf ohms.
        - Prints output using print_fault_v_i()
        - Index1: Bus index for fault
        - Vf: Prefault voltage in per unit
        - Zf: Complex fault impedance in ohms
    - get012toabc(self, k012)
        - Transforms a quantity in the positive, negative, zero domain into the A, B, C phase domain
        - K012: numpy matrix with X rows and 3 columns, corresponding to zero, positive, and negative sequence values respectively
        - Returns an X by 6 numpy matrix with magnitudes and phases for each A, B, and C quantity
    - getMaxError(self,deltaY)
        - Finds the maximum error in power between the calculated and expected power in the system
        - deltaY: power mismatch (error) vector
    - adjustAngleRange(angle)
        - Adjusts an angle to the range -180 to 180 degrees before it is used, based on mathematical standard
        - angle: angle to be adjusted
    - printMatrix(M)
        - Global class function to print out a 2D matrix. Used for debugging and verification

PowerSystem:
Class to hold all information and operations required for creating an electrical power system. Allows for adding transmission lines, transformers, generators, loads and buses to the system.

Variables:

- name
    - System name
- bus_order

    - List containing the all string bus names in the order that they were created in the system
    - Basis for how indexing is done on actual Ybus matrix
- buses
    - Dictionary for all buses in the system that allows for bus access by name
- connection_matrix
    - Numpy adjacency matrix for all connections in the system. When a component is added it has a weight (1 for lines and 2 for transformers), and at index (bus1 Index, bus2 index) a negative weight is added. At index (bus2 index, bus1 index) a positive weight is added. This shows directionality of the component
- yBusM
    - Numpy matrix YBus admittance values. It holds all zeros until the solve() function is called
- zBus0
    - System's zero sequence zBus matrix
- zBus1
    - System's positive sequence zBus matrix
- zBus2
    - System's negative sequence zBus matrix
- lines
    - List of all lines in the system
- transformers
    - List of all transformers in the system
- generators
    - List of all generators in the system
- loads
    - List of all loads in the system
- vBase
    - Voltage base at the slack bus that all voltage bases are determined by
    - Units of KV
- slackBus
    - Bus name of the system slack bus
- SBase
    - Power base for the system
    - Units of MVA
    - Default 100 MVA

Methods:

- __init__(self, name)
    - Initializes and Creates a YBus Variable with the name of system, and it establishes all YBus variables above
    - All lists and dictionaries are created empty
    - The default SBase is 100 MVA and the default vBase is 20 kV
- setFlatStart(self)

    - Sets all bus voltages to 1 per unit and all bus angles to 0 degrees
- add_bus(self, bus, type='PQ',voltage=1.0, angle=0.0)
    - Adds a bus to the system, updating bus_order list and buses dictionary
    - Buses are only added to lists and dictionaries if they do not already exist in the system
    - By default, first bus added is set as system slack bus with a voltage base of 20 kV
    - Adds a bus with options for bus type, voltage setpoint, and angle setpoint (default PQ bus at 1.0 per unit and 0 degrees)
    - Ex: system1.add_bus('A') adds bus A to the system if it does not already exist in the system
- add_Line(self, name, len, bus1, bus2, code:LineCode, geo:LineGeometry=LineGeometry())
    - Creates a line object using all variables sent in and adds to line list
    - Establishes the connection in the conneciton_matrix with -1 at index (bus1, bus2) and a 1 at (bus2, bus1)
    - Adds buses 1 and 2 to the system using add_bus()
    - Default Line Geometry will be used if not specified. See Line Geometry Documentation for more information
    - Ex: system1.add_Line('L1',10,'2','4',partridge,geo1) adds a 10 mile line to the system connecting bus 2 and 4. The line is defined by conductor information in the partridge variable and geometric information in the geo1 variable
- add_Transformer(self, name, bus1, bus2, v_1, v_2, s_rated, zt, x_r_ratio, bus1Conn, bus2Conn, groundingZ1, groundingZ2)
    - Creates a transformer object using all variables sent in and adds to transformer list
    - Establishes the connection in the conneciton_matrix with -2 at index (bus1, bus2) and a 2 at (bus2, bus1)
    - Adds buses 1 and 2 to the system using add_bus()
    - Ex: system1.add_Transformer('T1','1','2',20,230,125,0.085,10,'D','YG',-1,1) adds a 125 MVA 20/230 kV delta-grounded wye transformer to the system that connects buses '1' and '2'. It has a per unit impedance of 8.5% and an X/R ratio of 10. It has a grounding impedance of 1 ohm on the wye side
- add_Generator(self, name, bus, voltage, voltagePU, P_setpoint, Q_setpoint, Q_min, Q_max, Xg1, Xg2, Xg0, grounded, groundingZ)
    - Creates a generator object using all variables sent in
    - Adds a generator to the list at the specified bus
    - Ex: system1.add_Generator('G2', '7', 20,1, 200,0,-10000,10000,0.1,0.01,0.2,0,-1) adds a 200 MW, 20 kV (1 per unit), ungrounded generator at bus 7 with -10,000 to 10,000 Mvar as its Mvar limits

- add_Load(self, name, bus, P, Q)
    - Creates a load object using all the variables sent in
    - Adds a load at the specified bus
    - Ex: system1.add_Load('Z4', '4', 100, 70) adds a 100 MW, 70 Mvar load at bus 4
- set_SBase(self, S)
    - Sets the system apparent power base
    - Ex yBus1.set_SBase(250) sets the power base of the system to 250 MVA
- set_pu_Voltage(self, bus, v, angle=0)
    - Sets the per unit voltage at a bus
- set_Slack(self, slackBus, voltage):
    - Sets the name of the slack bus and the voltage base at that bus. This voltage base determines all other voltage bases
    - Ex: set_Slack('1',20) sets bus '1' as the slack bus with a voltage base of 20 kV
- findZBase(self,bus1)
    - Finds and returns the impedance base at a given bus
    - Gets a scalar multiplier from getMultiplier() to multiply by the voltage base at the slack bus. This new voltage base is used to calculate impedance base
    - This function is mainly intended for use in the solve() function
    - Ex yBus1.findZBase('1') would return 1.6 ohm for the system above in the documentation where bus 1 is the slack bus at 20 kV and the system power base is 250 MVA
- getVMultiplier(self, indexGoal)
    - Returns a scalar multiplier that is the scaling required to get from the voltage base of the system to the voltage base at the bus defined by indexGoal
    - Uses a depth first path finding search through the connnection_matrix to find the path from the slack bus to the desired bus, picking up multipliers based on going across transformers
    - Note that this function is intended to only be used by the findZBase() function
    - Ex: if vBase at the slack bus is 20 kV and bus 'A' at index 5 is 200 kV then yBus1.getMultiplier(5) returns 10
- make_YBus(self)
    - Solves the YBus matrix using the YBus matrix algorithm described in class
    - Function cycles through all lines and transformers, per unitizing the impedances and admittances and adding each value to appropriate spots in the YBus matrix according to the YBus matrix standard algorithm
    - Function calls on findZBase() to per unitize throughout the algorithm
    - Ex: yBus1.solve() configures the YBus matrix for the system stored in yBus1

- get_ZBus012(self)
    - Solves for the positive, negative, and zero sequence YBuses
    - Inverts YBuses into ZBus matrices
- print_YBus(self)
    - Prints the YBus matrix
- print_ZBuses(self)
    - Prints the ZBus matrices
- print_Results
    - Prints results from the solved system using Pandas tables
    - Prints the solved bus voltages, angles, real and reactive power, ampacity, line power loss, total power loss, and transformer power loss
- getCurrentAngles(self, lineCurrents)
    - Adjusts the current angles to fall between -90 and 90
- getCurrentMag(self, lineCurrents)
    - Gets the magnitude of the current in each line
- getCurrentDirections(self)
    - Finds which bus the current flows from and which bus it flows to
- getLineLosses(self)
    - Calculates the losses in each line and returns it as a vector with indexing aligning with the lines list
- printDCPwr(self)
    - Prints the solution to the DC power flow
- currentPowerGuess(self)
    - Cycles through each bus and calculates P and Q for using power injection equations
    - Puts P and Q values into proper place in yGuess matrix and returns final yGuess matrix
- getGivenPower(self)
    - Returns the Power Injection defined by all known loads and generators in a numpy matrix
- getXfmrLosses(self)
    - Calculates the losses in the transformers
- getLineCurrents(self)
    - Finds the current and current angle in each line
- getAmpacityPercent(self, lineCurrents)
    - Calculates the percent ampacity of the line
- get_Bus_Voltages(self)
    - Gets the voltage at each bus
- get_Volt_Angles(self)
    - Gets the angles of all the voltages at each bus
- calculatedPower(self)
    - Cycles through each bus and calculates the real and imaginary power
- updateBuses(self, deltaX, usedBuses)

    - Cycles through each bus and updates the values of voltages and angles based on the solved for deltaX
    - Only updates buses that not slack buses. Only updates angle for PV buses
- hasVisited(visited:list(),ind)
    - Public function for the YBus.py file
    - Returns true if the index number ind is in the list of all visited indices, false otherwise
    - Checks if visited contains index ind
    - Used by the DFS in getMultiplier()
    - Ex: YBus.hasVisited(visited,1) returns 0 if the visited list doesn't include index 1
- printMatrix(M)
    - Prints out all rows and columns of a matrix

Line:

Class to hold all information and operations related to transmission lines including impedance and admittance information.

Variables:

- name
    - Name of the line
- len
    - Length of the line in miles
- bus1
    - String name of the first bus the line is connected to
- bus2
    - String name of the second bus the line is connected to
- code
    - LineCode defined for this conductor. This variable contains information from the ACSR table needed for calculating line impedance and admittance parameters
- geo
    - LineGeometry defined for this transmission line. This variable contains information about conductor bundling and spacing calculating line impedance and admittance parameters
- ampacity
    - Maximum amount of current able to be carried by the line
- Z
    - Ohmic impedance of the line
- shuntY
    - Capacitive shunt admittance in Siemens for the line

Methods:

- __init__(self, name, len, bus1, bus2, code, geo)
    - Initializes a line with given name, length, bus connections, line code and geometry
    - Uses values sent in to calculate ampacity, impedance, and shunt admittance. Note that impedance and admittance is calculated using the findZ() findShuntyY() functions
    - Ex:L1= Line('L1',10,'2','4',partridge,geo1) creates a 10 mile line connecting buses 2 and 4 with geometry geo1 and conductor profile defined by the variable partridge. See LineGeometry and LineCode classes for more information
- per_unit_Z(self, Zbase)
    - Returns the per unit impedance of the line for a given impedance base
    - Ex: L1.per_unit_Z(10) returns 0.1 if the Z of the line is 1 Ohm
- findZ(len,code,geo)
    - Public method for the Line.py file that is not specific to an instance of the Line class
    - Returns the actual ohmic complex impedance of a line with a given length, line code (conductor information), and geometry (bundling and spacing)
    - Calculates both resistive and inductive components assuming transposed line
    - Ex: Line.findZ(10,partridge, geo1) returns the complex impedance in Ohms of the 10 mile long partridge conductor with given geometry
- findShuntY(len,code,geo)
    - Public method for the Line.py file that is not specific to an instance of the Line class
    - Returns the actual complex shunt admittance in Siemens for a line with a given length , line code (conductor information), and geometry (bundling and spacing)
    - Calculates both capacitive components assuming transposed line
    - Ex: Line.findShuntY(10,partridge, geo1) returns the complex admittance in Siemens of the 10 mile long partridge conductor with given geometry

Transformer:
Class to hold information and operations related to transmission system electrical transformers.

Variables:

- name
    - Name of the transformer
- bus1
    - Bus connection 1
- bus2
    - Bus connection 2
- v1

    - Voltage rating of bus 1 side of the transformer
- v2
    - Voltage rating of bus 2 side of the transformer
- s_rated
    - Transformer power rating
- zt
    - Per unit impedance of the transformer for nameplate ratings
- x_r_ratio
    - Ratio of reactance to resistance X/R of the transformer impedance
- bus1Conn
    - Type of connection on bus 1 (delta or wye)
- bus2Conn
    - Type of connection on bus 2 (delta or wye)
- groundingZ1
    - Grounding impedance at bus 1
- groundingZ2
    - Grounding impedance at bus 2

Methods:

- __init__(self, name, bus1, bus2, v1, v2, s_rated, zt, x_r_ratio, bus1Conn, bus2Conn, groundingZ1, groundingZ2):
    - Initializes and creates a transformer variable with a name, bus connections, voltage ratings, power ratings, and initial values for impedance, reactance to resistance ratio, bus connection types, as well as grounding impedances
    - Ex: T1=Transformer('T1','1','2',20,230,125,0.085,10) creates a 20/230 kV transformer T1 with bus 1 being the 20 kV side and bus 2 being the 230 kV side. Its power rating is 125 MVA. It also has a per unit impedance of 8.5% based on nameplate power and voltage ratings with an X/R ratio of 10
- per_unit_impedance(self, z_base)
    - Returns the per unit impedance of the transformer put onto another impedance base
    - This function recalculates complex per unit impedance by using the original impedance based defined by bus 1, v1, and s_rated
    - Ex: T1.per_unit_Z(1) finds the new per unit impedance of the transformer for a base of 1 Ohm by changing from the old per unit base defined by 20 kV on bus 1 and 125 MVA power

Generator:

Class to hold information and handle operations for generators in a power system

Variables:

- Name
  - Generator name
- Bus
  - Name of the bus the generator is connected to
- VoltagePU
  - Voltage set point at the generator in per unit
- Voltage
  - Voltage of the generator
- P_setpoint
  - Real Power Setpoint of the generator
- Q_setpoint
  - Reactive Power Setpoint of the generator
- Q_min
  - Reactive power lower limit
- Q_max
  - Reactive power upper limit
- vControl
  - Boolean for whether the generator is in voltage controlled or reactive power controlled mode. True if voltage controlled
- ZBaseOld
- Xg1
  - Positive sequence generator impedance
- Xg2
  - Negative sequence generator impedance
- Xg0
  - Zero sequence generator impedance
- Grounded
  - 1 for grounded 0 for ungrounded
- GroundingZ
  - Grounding impedance in ohms

Methods:

- __init__(self, name, bus, voltage, voltagePU, P_setpoint, Q_setpoint, Q_min, Q_max, Xg1, Xg2, Xg0, grounded, groundingZ):
  - Initialization function for a generator with default values for Q setpoint, Qmin, and Qmax
- setPower(self, P)
  - Set the starting real power of the generator
- set_Reactive(self,Q)
  - Set the starting reactive power of the generator

Load:

Variables:

- Name
  - Name of the load
- Bus

    - Name of the bus the load is connected to
- P
    - Real power draw of the load
- Q
    - Reactive power draw of the load

Methods:

- __init__(self, name, bus, P, Q):
    - Initialization function for a Load variable

Bus:
Class to hold information and handle operations for transmission system buses.

Variables:

- numBuses
    - Class variable for total number of buses in existence
- name
    - Name of the bus
- Index
    - Keeps track of the index of the bus in reference to all other buses in the system
- voltage
    - Voltage at that bus
- Angle
    - Voltage at that bus in radians
- Type
    - Voltage bus type (PQ, PV, or S (slack))

Methods:

- __init__(self,name, v=1.0, ang=0, type='PQ')
    - Initializes and creates a Bus variable with a name and gives the bus an index, relating to its order of creation compared to the other buses in existence. Can take in a voltage, angle, and type as well. The angle is sent in as degrees but stored as radians
    - Ex: b1= Bus('1') creates a Bus named bus 1 and adds to the total number of buses in existence
- set_type(self, type):
    - Sets the bus to a PV, PQ, or slack bus
- set_bus_voltage(self,voltage, angle=0)
    - Sets the voltage of the entered bus, and is able to set the angle of the bus
    - The angle of the bus is entered in degrees but stored as radians
    - Ex: b1.set_bus_voltage(10) sets the voltage at Bus b1 to 10 kV

LineCode:

Class to hold information about the conductor used for a transmission line.

Variables:

All variables reference a value from the ACSR conductor table

- name
  - Name of the conductor line code
  - Ex: “Partridge”, “Finch”, “Cardinal”
- dInches
  - Outer diameter of the line in inches
- GMRft
  - Geometric Mean Ratio of the line in feet
- ampacity
  - Current carrying ampacity of the line in amps
- R1perMi
  - Resistance per mile of the conductor at 50 degrees celsius

Methods:

- __init__(self, name, dInches, GMRft, ampacity, R1perMi)
  - Stores all of the above information in class variables
  - Ex: code=LineCode('Partridge',0.642,0.0217,460,0.385) creates a conductor code based on partridge information from the ACSR table

LineGeometry:

Class to hold information about the geometrical layout of a transmission line. This includes bundling and spacing information.

Variables:

- nConductors
  - Number of conductors in the bundle
  - Default 1 conductor
- bSpacingft
  - Spacing between bundles in feet
  - Default 1 foot
- Dab
  - Distance between the conductors for phases A and B
  - Default 1 foot based on ACSR datasheet
- Dbc
  - Distance between the conductors for phases B and C
  - Default 1 foot based on ACSR datasheet
- Dca
  - Distance between the conductors for phases C and A
  - Default 1 foot based on ACSR datasheet

Methods:

- __init__(self, nConductors, bSpacingft, Dab, Dbc, Dbc)
  - Initializes a LineGeometry object with given parameters above. Note the default values in the variable description section

- Ex: geo1=LineGeometry() defines a LineGeometry object with 1 conductor, 1 foot bundle spacing, and 1 foot spacing between all conductors
- Ex: geo2=LineGeometry(3,2,5,5,10) defines a line geometry with 3 conductors per bundle with 2 foot bundle spacing, 5 feet between A and B, 5 feet between B and C, and 10 feet between C and A

CSV Formats for Generalized Classes:

The following information shows the formats for CSV files used in creating a power system for this simulator.