Aiden Landis & Jack Carnovale
**Power Flow Simulator: Milestone 2 Documentation**
This document serves to show how to use PowerSystem, Line, Transformer, and other classes.
It will go into all methods and variables accessible by a user of the class.

**PowerSystem**:
Class to hold all information and operations required for creating a YBus matrix for an electrical
power system. Allows for adding transmission lines, transformers, 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
    - Matrix is initialized with a max size of 100 by 100, but only the indexes 0
      to (length of bus_order - 1) are considered in calculation and processing
  - yBusM
    - Numpy matrix YBus admittance values. It holds all zeros until the solve()
      function is called
    - Matrix is initialized with a max size of 100 by 100, but only the indexes 0
      to (length of bus_order - 1) are considered in calculation and processing
  - lines
    - Dictionary of all lines in the system that allows for line access by name
  - transformers
    - Dictionary of all transformers in the system that allows for line access by
      name
  - vBase
    - Voltage base at the slack bus that all voltage bases are determined by
  - slackBus
    - Bus name of the system slack bus
  - SBase
    - Power base for the system
  - generators

- - Dictionary of all generators in the system that allows for line access by name
- loads
  - Dictionary of all loads in the system that allows for line access by

**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 numpy matrices connection_matrix and yBusM are set to all zero with a size of 100 x 100
  - The default SBase is 100 MVA and the default vBase is 20 kV
  - Ex: yBusM=None
- add_bus(self, bus)
  - 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
  - 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 dictionary
  - 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)
  - Creates a transformer object using all variables sent in and adds to transformer dictionary
  - 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) adds a 125 MVA 20/230 kV 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
- add_Generator(self, name, bus, voltage, P_setpoint, Q_setpoint = 0, Q_min = -10000, Q_max = 10000)

- - 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', 1, 200) adds a 200 MW, 1V transformer at bus 7.
- 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
- solve(self)
  - Solves the Power system using Newton Raphson
  - Gets YBus Matrix
  - Solves for deltaY using yGoal and yGuess
  - Reduces the jacobian by removing the slack bus and PV buses and inverts jacobian
  - Solves for deltaX using the inverted jacobian and deltaY
- 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
- getJacobian(self)
  - Cycles through each bus and builds the jacobian matrix
  - Returns a numpy Jacobian Matrix
- getGivenPower(self)
  - Returns the Power Injection defined by all known loads and generators in a numpy matrix
- 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
- print_YBus(self)
  - Prints the YBus matrix
- print_Bus_Voltages(self)
  - Prints the calculated voltage magnitude and voltage angles of all buses in proper order
- printMatrix(M)
  - Prints out all rows and columns of a matrix
- 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

## 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

### Methods:
- __init__(self, name, bus1, bus2, v1, v2, s_rated, zt, x_r_ratio):

- ○ Initializes and creates a transformer variable with a name, bus connections, voltage ratings, power ratings, and initial values for impedance
- ○ 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
- Voltage
  - ○ Voltage set point at the generator in per unit
- P
  - ○ Real Power Setpoint of the generator
- Q
  - ○ 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

**Methods:**
- __init__(self,name,bus,voltage,P_setpoint,Q_setpoint=0,Q_min=-1000, Q_max=10000):
  - ○ 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