EE454 – Power System Analysis – Project

Due Date: Wednesday December 1, 2021, at 11:59 pm on Canvas

(Late projects will **not** be accepted)

Specification:

Design, write, test, and document a computer program in Python that solves the power flow problem using the Newton-Raphson method and exports results to a file.

You may work on this project individually or in teams of two maximum. If you choose to work as a team, your report must include a section describing the contribution of each team member.

Testing:

1. Use the 12-bus network described below to test your program. An input data file with the network information will be provided for each test case.
2. Run your power flow program to study how the test system behaves
    a. Under the base case conditions
    b. Under the contingency conditions described below.

Report:

Your report should not exceed 7 pages (excluding appendices) and should follow the good practices that you are taught in EE393. In particular, pay attention to grammar, spelling, conciseness, and use of active voice.

It must include the following:

- A concise description of the design of your program, including a flowchart or other schematic representation.
- A description of the tests that you performed to verify the correctness of your program (see section on software development below)
    - If your code fails to work by the deadline, include a discussion of your approach to fixing and validating components of the code.
- The results that you obtained when running your program for the conditions given below. This includes:
    - The voltage magnitude (p.u.) and angle (degrees) at each bus of the test system

    - The active (MW) and reactive power (MVAr) produced by each generator and each synchronous condenser
    - The active (MW), reactive (MVAr) and apparent (MVA) power flowing in each line
    - Any violation of normal operating limits on voltages and flows
- A brief discussion of these results i.e., do they make sense based on the physics of flows and voltages in power systems as discussed in the lectures.
- For the test system under the base case conditions, provide a convergence record i.e., for each iteration of the Newton-Raphson method, provide the following information in a table:
    - Largest active power mismatch and bus where this mismatch occurs
    - Largest reactive power mismatch and bus where this mismatch occurs
- If applicable, a section describing the contributions of each team member.
- Your documented code should be included as an appendix to your report (and does not count against the seven-page limit). It should also be provided as .py files (see below).

Deliverables:

- Report as described above, in .pdf form.
- Python code in a .py file or files, and all the data files that your program needs i.e., all the files needed to check that your program actually runs for the base case conditions. Your code should export the following results to .csv or .xlsx files upon execution:
    - Network admittance matrix
    - Convergence history of bus results
    - Final bus results (P, Q, V, θ) and voltage limit check
    - Line flows and MVA limit check

Grading:

- Demonstration of a working power flow program: 40%
- Discussion of the results of your program for the base case and contingency conditions: 20%
- Description of the structure of your program: 10%
- Quality of the code (structure, clarity, documentation): 15%
- Quality and presentation of the report: 15%

Software development:

- The use of Python is recommended. If you are not familiar with Python, you may use another language. However, the amount of help that you may be able to get from the TA could be limited if you choose to use another language.
- It is very important to write code that is clean and easy to read. In particular, this means that you should:
    - Organize your program in a top-down fashion with a main program that calls various functions. Each function can also call other functions
    - Each of these functions should do one thing and be short (a useful rule of thumb is to limit each function to a maximum 15 to 20 lines of code. If you need more than that, consider dividing it into several functions)
    - At the top level, a program of this type is typically structured as follows:
        - Data input: the program reads the input file
        - Processing
        - Data output: the program displays and saves the results
- Your code must be documented. This means that you must include the following in your code:
    - Definition of each variable and parameter
    - Definition of the purpose of each function e.g., "Build the Jacobian"
    - List of the inputs of each function i.e., the variables and parameters that it uses
    - List of the outputs of each function i.e., the variables that it calculates
    - As an example of a good documentation technique that includes each of the above items, please refer to the documentation on creating Sphinx docstrings
- Consider using a code formatter, such as black (https://pypi.org/project/black/)
- Avoid "hard coding" data and parameters in your program, for example:
    - Use a function to read from a file each type of input data
    - Use parameters rather than numbers inside your code.
- Test each part of your software separately before putting them together. For example, check that the functions you use to read the input data work properly before integrating them with the functions that perform the power flow computations

Data

Your program should be tested on the 12-bus test system whose diagram is shown in Figure 1. All the data is given with S_{base}=100 MVA. Use bus 1 as the slack bus.

Do NOT retype this data. It is provided in the file "system_basecase.xlsx".

To test for convergence, use a maximum mismatch of 0.1 MW, 0.1 MVAr.

BUS DATA

1. Load at each bus in the system:

Bus #	2	3	4	5	6	7	8	9	10	11	12
P MW	23.7	84.2	57.8	7.6	13.5	29.5	9	4.3	5.2	13.5	14.9
Q MVAr	15.3	19	-3.9	1.6	8.5	13.6	5.8	2.1	1.6	5.8	5

2. Active power produced by each generator in the system:

Bus #	1	2	3	10	12
P MW	-	42	23	33	27

3. Reference voltages at the PV busses:

Bus #	1	2	3	10	12
$V^{reference}$ p.u.	1.050	1.045	1.010	1.060	1.040

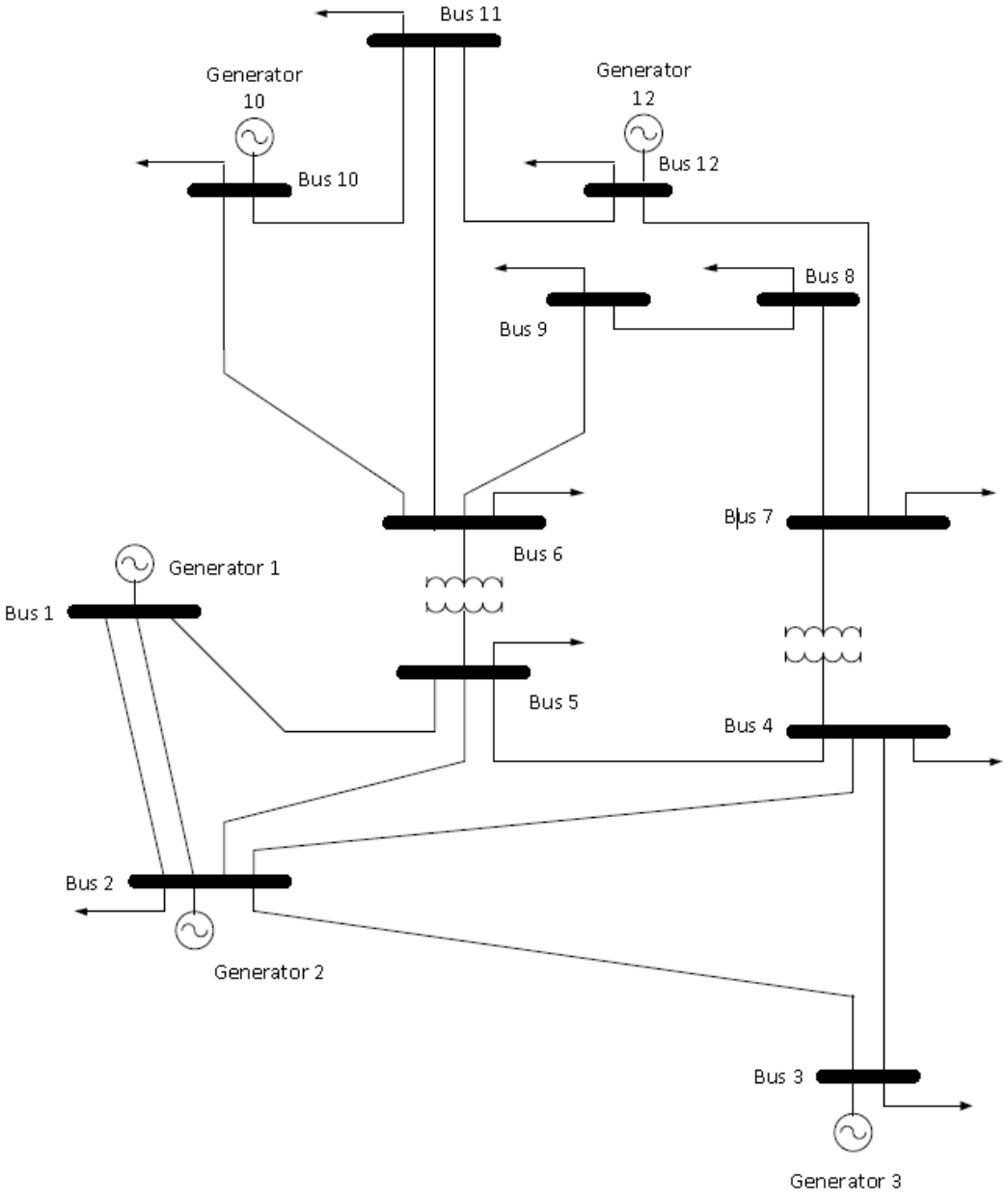

Figure 1: One-line diagram of the test system

BRANCH DATA

Line	R^{total}, p.u.	X^{total}, p.u.	B^{total}, p.u.	F^{max}, MVA
1-2	0.01938	0.05917	0.0528	95
1-5	0.05403	0.22304	0.0492	100
2-3	0.04699	0.19797	0.0438	-
2-4	0.05811	0.17632	0.0340	-
2-5	0.05695	0.17388	0.0346	-
3-4	0.6701	0.17103	0.0128	-
4-5	0.01335	0.04211	0	-
4-7	0	0.55618	0	-
5-6	0	0.25202	0	-
6-9	0.09498	0.19890	0	-
6-10	0.12291	0.25581	0	-
6-11	0.06615	0.13027	0	-
7-8	0.03181	0.08450	0	-
7-12	0.12711	0.27038	0	-
8-9	0.08205	0.19207	0	-
10-11	0.22092	0.19988	0	-
11-12	0.17093	0.34802	0	-

All branches are transmission lines, except branches 4-7 and 5-6, which are transformers. Note that the parameters above are total impedance of branches i.e., if a line is double circuit, then values in the table are the impedance and susceptance of the parallel combination of the circuits. The last column shows the maximum transmission capacity of some lines. For parallel lines, it is the total transmission capacity of the parallel lines.

Test Cases:

1. *Base case:* Solve the power flow program for the conditions described by the data given above. Your program should report on whether the flow in any line or the voltage at any bus exceeds normal operating limits. Voltages should be between 0.95 and 1.05. Line flow limits (in MVA) are given for some lines in the table above.
2. *Contingency case 1:* Solve the power flow for the case where one circuit of line 1-2 is taken out of service. Your program should report on whether the flow in any line or the voltage at any bus exceeds normal operating limits. In your report compare voltages and power flows with the base case.
3. *Contingency case 2:* Solve the power flow for the case where line 10-11 is taken out of service. Your program should report line flow and bus voltage violations. In your report compare voltages and power flows with the base case.

Some Advice

This is advice, not requirements!

Stubs

Use stubs to test parts of the program when the other parts are not working yet. For example, use a hard-coded function to populate matrices with the problem data, so you can work on processing while the input routine is being written. Similarly, use a hard-coded function to populate the matrices of output data that the processing will produce. This will allow you to work on the output before the development of the processing module is complete. Obviously, the data structures in the stub files will need to match the data structures used in the actual program.

Excel

Use Excel files or .csv files for input and output. Many Python libraries provide functions to easily do this (e.g., `openpyxl`).

Initialize your variables and matrices!

Python:

```
N = 2
Matrix = numpy.zeros((N,N))
```

Note how matrix size is parameterized rather than hard coded.

Processing Steps

1. Calculate the mismatches
2. Test for convergence
3. Exit if sufficiently small
4. Build the Jacobian
5. Invert the Jacobian
6. Calculate corrections
7. Update variables
8. Go back to step 1

On exit:

1. Calculate explicit variables
2. Calculate line flows

Descriptive Variable Names

Use descriptive variable names.

Good: `J, jacobian, Vset, Vcalc`

Bad: `matrix, m, v1, v2`

Debugging and Version Control

Use an IDE (e.g., PyCharm, Spyder, Jupyter notebooks, etc.) to test each piece of code as it is written, in isolation. Create input data, run the code with it, and observe the output data. You should have an idea of what the output data should look like, given the input, since you wrote the code. When the output data does not do what you expect, use the debugger to add break points, watch variables, and step through your code.

Whenever a piece of code works, either in isolation or in integration, COMMIT the working code to a local (or, ideally, a remote) repository (e.g., GitHub). This is an important coding habit that will help you effectively manage your code, track down bugs, and save work in the event of a computer crash.

Debugging and version control tools may seem time consuming to set up, but they will save you time finding and fixing errors in the long run. Talk to the TA for support on getting this set up.

Python

A Python Tutorial

The "official" Python tutorial is available at:
https://docs.python.org/3/tutorial/index.html

If you have a question about any specific thing in Python, do a web search for "python <thing>" and you will find lots of help, including but not limited to the official documentation.

Additional user guides and tutorials relevant to this project:

- NumPy quick-start tutorial
  https://docs.scipy.org/doc/numpy/user/quickstart.html#
- More NumPy Tutorials http://cs231n.github.io/python-numpy-tutorial/
- PyCharm IDE: https://www.jetbrains.com/pycharm/download/#section=windows
- GitHub (version control): https://github.com/

Downloading Python

https://www.python.org/

Consider downloading the current release of Anaconda. It comes with many useful scientific packages preinstalled.

You should be able to complete your project using the following packages/libraries:

- `numpy, scipy, math`
- `pandas, openpyxl, xlsxwriter, csv` (for reading and writing data files)

You must consult the TA before using another package.

Python Forbidden Functions

Python is too full of useful stuff. To maximize learning, the use of functions that perform optimization or compute derivatives is forbidden. Examples include:

`scipy.optimize.fsolve, scipy.optimize.newton,`
`scipy.optimize.approx_fprime, numpy.gradient,` *etc.*

The use of Symbolic variables is also forbidden.