EE454 – Power System Analysis – Project

Due Date: Friday December 7, 2018 at 5:00 pm to Canvas

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

Specification

Design, write, test and document a computer program in Python (recommended) or Matlab that solves the power flow problem using the Newton Raphson method.

Testing

1. Use the data given below to test your program.
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.

(These can be separate runs with separate data files.)

Report

Your report should not exceed 7 pages (excluding appendices) and 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)
- 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 tabular form:
    - Largest active power mismatch and bus where this mismatch occurs
    - Largest reactive power mismatch and bus where this mismatch occurs
- 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 executable .py or .m files (see below).

Deliverables

- Report as described above, in .pdf form.
- Python code in a .py file or files, or Matlab code in a .m 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). The main program of your code should require no additional inputs from a user, i.e. the program should read the input file automatically.

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): 30%

Software development

- Use Python (recommended) or Matlab
- 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 should read the input file)
        - Processing
        - Data output (the program should display and write the output file)
- 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)
- 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
- Test with a small case where you know the solution, such as a three bus case.

Data

Your program should be tested on the 14-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. 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	21.7	94.2	47.8	7.6	11.2	29.5	9	3.5	6.1	13.5	14.9
Q MVAr	12.7	19.0	-3.9	1.6	7.5	16.6	5.8	1.8	1.6	5.8	5.0

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

Bus #	1	2	3	10	12
P MW	-	40	26	28	30

3. Reference voltages at the PV busses:

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

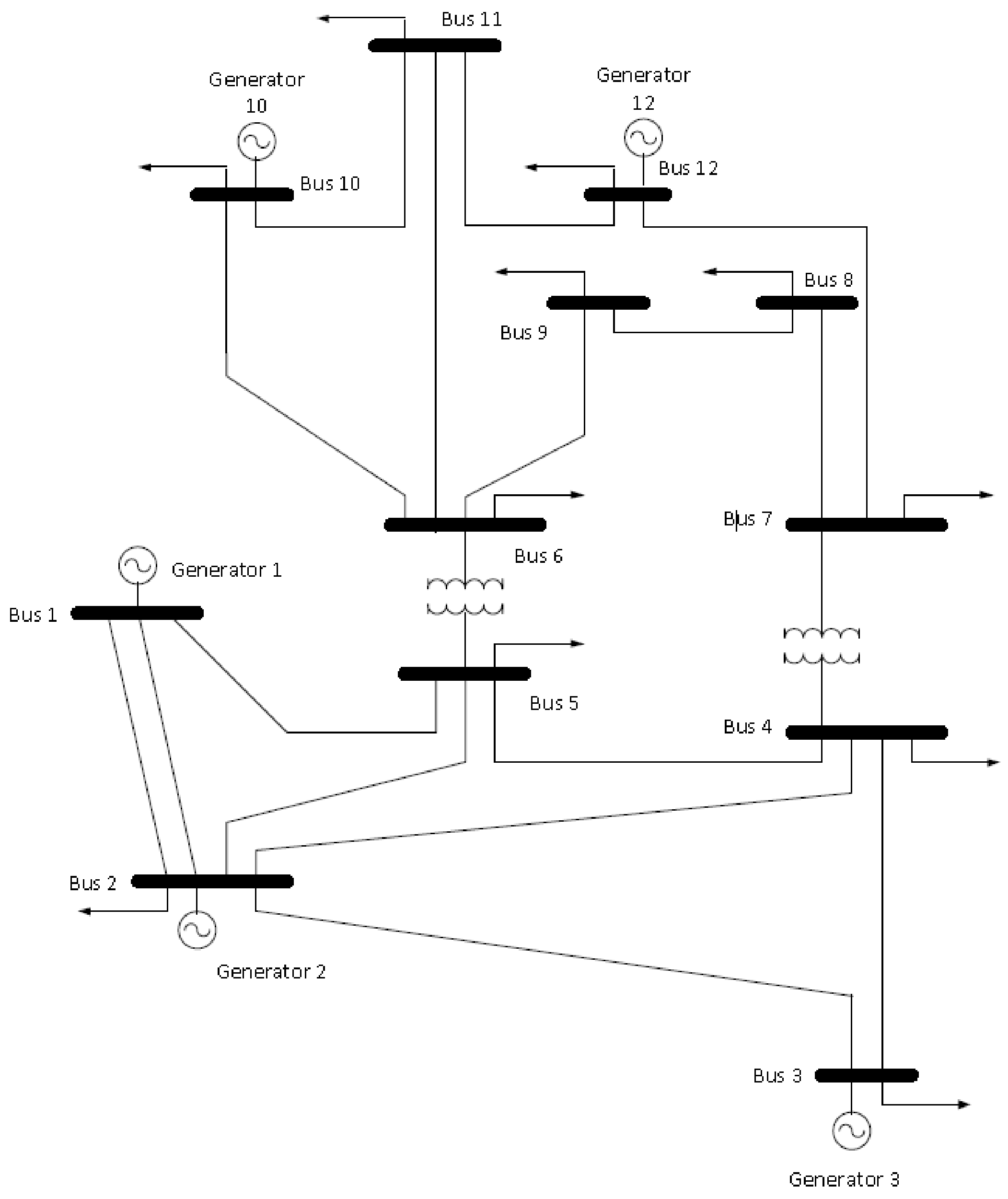

Figure 1: One-line diagram of the test system

LINE 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 4-9 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. Limits on line flows (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 1-5 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.

Python Forbidden Functions

Avoid functions similar to the Matlab forbidden functions below.

In case of doubt, consult me.

Matlab Forbidden Functions

Matlab is too full of useful stuff. To maximize learning, the following functions are forbidden:

`fsolve, jacobian, curl, diff, divergence,` etc.

Symbolic variables are forbidden.

Functions you can use:

`inv, lu, matrix multiplication`

In case of doubt consult me.

(More) 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 the processing will produce to work on the output before the processing 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 for input and output. Both Python and Matlab provide functions to do this. The Python package is `openpyxl`. Matlab functions are s `xlsread` and `xlswrite.`

It's OK to have separate input files for buses and lines. This is much easier than reading data from formatted ASCII files, and writing a formatted output log.

Initialize your variables and matrices!

Python:

```
N = 2
Matrix = [[0.] * N for i in range(N)]
```

Matlab:

```
N = 2;
matrix = 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

Divide et Impera

Divide and Conquer – There's a lot of opportunity for different people to write different parts of the program. Be careful to ensure that everybody on either side of an interface knows the data that has to be passed and what form the data should take.

Descriptive Variable Names

Use descriptive variable names.

Good: `J, jacobian, Vset, Vcalc`

Bad: `matrix, m, v1, v2`

Debugging

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, there's a problem in between. Use the debugger, if you can, or simply have the program print out intermediate results until you find and fix the error.

THEN add the new code to the existing working code, and test that integrated system. Anything wrong is the fault of the new code. (You only added one new piece, right?) Again, the debugger or printing results should be used to find and fix errors.

Whenever a piece of code works, either in isolation or in integration, SAVE the working code in a separate directory. In case something breaks, or gets overwritten, or the disk crashes, or whatever.

Python

Why Python

- It was suggested at a faculty meeting in Spring 2018 that most of you are familiar with Python from your signal processing courses.
- Python syntax is closer to C (as used in real world applications programs when performance is an issue) than Matlab. This is the strongest reason to use Python.
- Python is free.
- Python is portable. The same Python code runs on Apple, PC, and Unix.
- Python performance is similar to Matlab. (Based on a shallow review of the Web. Performance is not an issue in this project.)
- Matlab matrix operation syntax is cleaner than Python, but Python is cleaner on everything else.
- About the only web site that prefers Matlab to Python is Mathworks – who sell Matlab.

A Python Tutorial

The "official" Python tutorial is 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.

Downloading Python

https://www.python.org

Python Version

Use Python 3, and not Python 2. The version I used in September 2018 was 3.6.5. The latest version (Oct 20, 2018) is 3.7.1. There are a few syntax changes between Python 2 and Python 3, notably in the print statement. 3.6.5 and 3.7.1 should not differ markedly for purposes of the project.

Packages

Where Matlab has *toolboxes*, Python has *packages* (also called libraries or modules), collections of functions for specific purposes. Python requires packages for some of the functions that appear in vanilla Matlab. While some Python functions are essentially identical to Matlab (`sin`, `cos`), others have different names, arguments or invocations (as a method in Python, for example). Packages require some set up outside the program (installation in your Python interpreter) and invocation in the program (`import` statement).

Installing packages: see

https://docs.python.org/3/installing/index.html

Relatively few packages are needed for this project. My trial implementation used:

- `math` (for `sin, cos`)
- `cmath` (for `rect`, conversion from polar to rectangular coordinates, `.conjugate` method)
- `openpyxl` (for `load_workbook`, `.save` method)
- `numpy` – for vector `max` and `min`, and `linalg.solve` for Ax = b.