Spring
2024

46705
Power Grid Analysis

Assignment 1

Subject: Power Flow in Python for Grid Analysis

Issued: ***06.02.2024***
Submission date: ***26.03.2024***

Introduction

The purpose of this assignment is: 1) to write a Python program that carries out a power flow analysis of a given network, and 2) study different scenarios to analyse whether voltage or generator limits have been violated. The power flow program created in this assignment will be re-used and modified during the final assignment in this course.

Learning Objectives for the Assignment

A student that has fulfilled the learning objectives for this assignment should be able to:

- Explain the goal of a power flow algorithm, that is, which system variables are calculated in an iterative process and returned as a solution to the power flow problem.
- Implement power flow analysis on a computer, including constructing a power flow model from given network data file.
- Apply power flow analysis on different operaional scenarios and interpret power flow results with respect to system security constraints.

Note: It is a good idea to address these points in your conclusion and keep them in mind when preparing for the final multiple choice exam.

Power Flow in Python

The aim of the assignment is to get familiar with the main steps associated with the process of obtaining a power flow solution. For that purpose, a small program will be written in Python which solves the power flow problem for a given electrical network. The main script for the power flow program is provided in table 1. The lines in the script, which are colored red denote functions and *.py files that you have to write in order to complete the assignment.

```
import PowerFlow_46705 as pf  # import Power Flow functions
import LoadNetworkData as lnd     # load the network data to global variables
max_iter = 30    # Iteration settings
err_tol = 1e-4
# Load the Network data ...
filename = "IEEE_14bus.txt"
lnd.LoadNetworkData(filename) # makes Ybus available as lnd.Ybus and etc.
# Carry out the power flow analysis ...
V,success,n = pf.PowerFlowNewton(lnd.Ybus,lnd.Sbus,lnd.V0,lnd.ref,lnd.pv_index,lnd.pq_index,
    max_iter,err_tol)
# Display results if the power flow analysis converged
if success:
    pf.DisplayResults(V,lnd) #Now we are just passing the lnd module as input

```

Table 1: A Python script that carries out a power flow analysis. In this assignment, the students are supposed to write the functions colored in red.

The functions that you have to write are:

- The function `LoadNetworkData()` (stored in `LoadNetworkData.py`), which reads in all the necessary information about the network (busses, generators, loads, lines and transformers). The function creates the matrices and arrays needed to carry out a power flow analysis.
- The function `PowerFlowNewton()` (to be stored in `PowerFlow_46705.py`), which is the essential part of the program. This function carries out the power flow calculation.
- The function `DisplayResults()` (to be stored in `PowerFlow_46705.py`), that displays the results from the power flow analysis.

In the following, each of the above mentioned functions and script will be constructed. The first step in the construction of the power flow program will be the writing of the `PowerFlowNewton()` function.

Task 1 - Construction of the `PowerFlowNewton()` function

On *DTU-LEARN*, under the link for assignment 1 you find the file `PowerFlow_46705.py` which contains empty definitions of the functions needed for the power flow computations. Your task in the following is to complete all of the functions in that file.

As mentioned above, the function `PowerFlowNewton()` is the core of the power flow program. A Python skeleton for the function is provided in table 2.

The input to the function `PowerFlowNewton()` is the $N \times N$ bus admittance matrix of the Network `Ybus`, the $N \times 1$ power injection vector `Sbus` and the vectors `,pv_index` and `pq_index` which contain the indexes of PV-busses and PQ-busses respectively. As output, the function returns the iterated bus voltages `V`, the flag `success` which indicates whether the iteration succeeded or not, and the number of iterations used `n`.

In the following, each of the functions colored red in table 2 will be written. A description of each of these functions together with helpful hints regarding the code writing of the functions can be found on DTU-LEARN in the lecture handouts concerning this assignment.

Construction of user-written sub-functions in `PowerFlowNewton()`

In the following you will be asked to write each of the functions colored red in table 2.

```python
def PowerFlowNewton(Ybus,Sbus,V0,ref,pv_index,pq_index,max_iter,err_tol):
  success = 0 #Initialization of status flag and iteration counter
  n = 0
  V = V0
  print('  iteration      maximum P & Q mismatch (pu)')
  print('  ---------      ---------------------------')
  # Determine mismatch between initial guess and and specified value for P and Q
  F = calculate_F(Ybus,Sbus,V,pv_index,pq_index)
  # Check if the desired tolerance is reached
  success = CheckTolerance(F,n,err_tol)
  # Start the Newton iteration loop
  while (not success) and (n < max_iter):
      n += 1  # Update counter
      # Compute derivatives and generate the Jacobian matrix
      J_dS_dVm,J_dS_dTheta = generate_Derivatives(Ybus,V)
      J = generate_Jacobian(J_dS_dVm,J_dS_dTheta,pv_index,pq_index)
      # Compute the update step
      dx = np.linalg.solve(J,F)
      # Update voltages and check if tolerance is now reached
      V = Update_Voltages(dx,V,pv_index,pq_index)
      F = calculate_F(Ybus,Sbus,V,pv_index,pq_index)
      success = CheckTolerance(F,n,err_tol)

  if success: #print out message concerning wether the power flow converged or not
      print('The Newton Rapson Power Flow Converged in %d iterations!' % (n,))
  else:
      print('No Convergence !!!\n Stopped after %d iterations without solution...' % (n,))
  return V,success,n
```

Table 2: Skeleton for the function `PowerFlowNewton()`. The part of the function colored red, represents functions which the students have to write.

Q.1.a Write the function `calculate_F()` which calculates the mismatch between the specified values of P and Q and the values calculated from the iterated bus voltage vector. (Note: See the handouts for hint). In the report, you are expected to describe how you came up with your answer (e.g. which equation you used and implemented in Python).

Q.1.b Write the function `CheckTolerence()` which checks whether the desired tolerance has been reached. The function should return an integer value which shall be equal to 1 if the tolerance has been reached, otherwise it should be 0. Describe how you came up with your answer (e.g. which equation you used and implemented in Python).

Q.1.c Write the function `generate_Derivatives()` that calculates the derivatives of active and reactive power in respect to voltage magnitude and angle. Read the handouts for further description. Describe how you came up with your answer (e.g. which equation you used and implemented in Python).

Q.1.d Write the function `generate_Jacobian()`, which assembles the Jacobian from the derivatives obtained by `generate_Derivatives()`. Describe how you came up with your answer.

Q.1.e Write the function `Update_Voltages()` that updates the voltage magnitudes and angles when ΔV and $\Delta\theta$ have been determined. Describe how you came up with your answer.

Task 2 - Construction of the `LoadNetworkData()` function

In this part of the assignment, a function (`LoadNetworkData()`) is written which contains the relevant model data needed to carry out power flow calculations. The model data is stored in a text

file that has to be read into Python. The Python file `ReadNetworkData.py`, which was uploaded with the assignment, provides a function that can read in the model data and stores the information in several objects. The code segment below shows how `ReadNetworkData` is imported and how the `read_network_data_from_file()` function is used to get the relevant information from a model file with the name `TestSystem.txt`.

```
import numpy as np
import ReadNetworkData as rd
filename = 'TestSystem.txt'
#read in the data from the file...
bus_data,load_data,gen_data,line_data, tran_data,mva_base, bus_to_ind, ind_to_bus = \
rd.read_network_data_from_file(filename)
```

The variables returned by `read_network_data_from_file()` are:

- `bus_data`: rows contain `[bus_nr, bus_label, Kv_level, code]`
- `load_data`: rows contain `[bus_nr, P_LD_MW, Q_LD_MW]` (loads in MW)
- `gen_data`: rows contain `[bus_nr, MVA_size, P_Gen]` (in MVA and MW)
- `line_data`: rows contain `[fr_bus, to_bus, ID_label, R, X, B/2]` (pu system base)
- `tran_data`: contains `[fr_bus, to_bus, ID_label, R_eq, X_eq, n_pu, ang_deg]` (`R_eq` and `X_eq` in pu on system base, `n_pu`: the pu turns ratio, `ang_deg`: is the phase shift)
- `mva_base`: the system MVA base
- `bus_to_ind`: mapping from bus numbers to the corresponding indices in the bus matrices and arrays
- `ind_to_bus`: containing the mapping from the indices to the busses (the opposite of the above)

The information provided in the variables above contains all the information needed to construct the matrices and arrays used for the power flow calculations and to display the results. Use the code above to read in the simple test system in the file `TestSystem.txt` and explore the content of the above variables.

Q.2 Write the `LoadNetworkData()` function, which creates automatically the following matrices and arrays:

- Admittance matrices: `Ybus`, `Y_from`, `Y_to`
- Bus indices for the branch from and to ends: `br_f`, `br_t`
- Information about the bus types: `buscode`
- Text label for each bus: `bus_labels`
- Array with the complex power load at each bus: `S_LD`
- The system MVA base: `MVA_base`

The first lines of the of the file `LoadNetworkData.py` could look something like displayed in table 3. The handouts from the lecture about Power Flow in Python contain examples of the data structures that are to be created automatically in `LoadNetworkData()`.

```python
import numpy as np
import ReadNetworkData as rd

def LoadNetworkData(filename):
    global Ybus, Sbus, V0, buscode, pq_index, pv_index, ref, Y_fr, Y_to, br_f,br_t,br_Y,S_LD,
     ind_to_bus, bus_to_ind, MVA_base, bus_labels
    #read in the data from the file...
    bus_data,load_data,gen_data,line_data, tran_data,mva_base, bus_to_ind, ind_to_bus = \
    rd.read_network_data_from_file(filename)

    ########################################################################
    # Construct the Ybus matrix from elements in the line_data and trans_data
    #
    # Sweep over the data arrays to get th
    #
    ########################################################################
    MVA_base = mva_base
    N = len(bus_data) #Number of buses
    M_lines = len(line_data)
    M_trans = len(tran_data)
    M_branches = M_lines + M_trans
    Ybus = np.zeros((N,N),dtype=complex)

# Continue with your code here and create the needed data types...
# sweep over the bus_data,load_data,gen_data,line_data, tran_data to fill in appropriate
    values
```

Table 3: Example of what the first few lines of `LoadNetworkData.py` (in Q.2) could look like.

Task 3 - Construction of the `DisplayResults()` function

When the power flow has been solved, the next step in the main script in table 1 is to display the results. The job of displaying the power flow results is carried out by the function `DisplayResults()`. The function shall display both the bus results (voltage and power injection) and the branch results (power flowing into the busses from each branch). Example of how such printout of results could look like is provided in table 4.

The following data (from the `LoadNetworkData` module) will be needed to print out the results together with the bus voltage array returned by `PowerFlowNewton()`:

- Admittance matrices: `Ybus`, `Y_from`, `Y_to`
- Bus indices for the branch from and to ends: `br_f`, `br_t`
- Mapping arrays from bus numbers to indices (and vice versa): `ind_to_bus`, `bus_to_ind`
- Information about the bus types: `buscode`
- Text label for each bus: `bus_labels`
- Array with the complex power load at each bus: `S_LD`
- The system MVA base: `MVA_base`

Q.3 Write the function `DisplayResults()`, which is described above. Table 5 shows example of what could be the first few lines of the function where the relevant data is accessed in the `lnd` object. In your answer, you need to describe which calculations you had to do to produce the displayed results.

```
=====================================================================
|                          Bus results                              |
=====================================================================
 Bus     Bus          Voltage          Generation          Load
 Nr     Label     Mag(pu)  Ang(deg)   P (pu)  Q(pu)     P (pu)  Q(pu)
----- ---------  -------  -------  -------  --------  -------  -------
  1    BUS1HV     1.000    -1.77    1.800    -0.507    0.000    -0.300
  2    BUS2HV     1.000    -9.31    0.650     0.697    0.600     0.250
  3    BUS3HV     1.000   -11.34    0.650     0.139    0.600     0.250
                        ........ and so on ..........

=========================================================
|                     Branch flow                       |
 =======================================================
|Branch  From   To     From Bus Inject.   To Bus Inject. |
| Nr     Bus    Bus    P (pu)   Q (pu)    P (pu)   Q (pu)|
 -----  -----  -----  -------  --------  -------  -------
   1      1      2     0.947   -0.252    -0.909    0.324
   2      1      5     0.853    0.045    -0.814    0.026
   3      2      3     0.170   -0.081    -0.169   -0.001
   4      2      4     0.385    0.080    -0.375   -0.117
               ........ and so on ..........
```

Table 4: Example showing which type of information should be printed out automatically when the function `DisplayResults()` is called.

```
def DisplayResults(V,lnd):
    Ybus=lnd.Ybus; Y_from=lnd.Y_fr; Y_to=lnd.Y_to; br_f=lnd.br_f; br_t=lnd.br_t;
    buscode=lnd.buscode; SLD=lnd.S_LD; ind_to_bus=lnd.ind_to_bus;
    bus_to_ind=lnd.bus_to_ind; MVA_base=lnd.MVA_base; bus_labels=lnd.bus_labels
```

Table 5: Example of how the first lines of `DisplayResults()` could look like.

Task 4 - Analysis of the Kundur's two area test system

The file `Kundur_two_area_system.txt` contains the model parameters for Kundur's two area test system. In the following, you will use your program to study different operating conditions for that system.

Q.4.a Run the power flow program on the `Kundur_two_area_system.txt` system, display the results and examen wether any generator is overloaded or if bus voltage limits of $\pm 0.05 pu$ is violated.

Q.4.a Repeat the above, but now with one of the lines between busses 6 and 7 disconnected (use `\\` in front of the line in `Kundur_two_area_system.txt` to comment the line out).

Guidelines for the Writing of the Report

When writing the report for the assignment, the below listed points should be followed:

- The report must start with a self-evaluation section, where the authors state whether the report represents an even contribution from all group members or whether some participants were not effectively contributing to the report.
- The front page of the report should contain your names and student numbers.
- Your Python/Matlab code used for the solution of this assignment has to be included in appendix.
- The report shall contain the answer to each of the questions. The answers shall be clearly separated from each other and come in the same order as the questions in the assignment. It is not necessary to repeat the text from the assignment in your report.
- When answering the questions, both the results as well as an explanation on how you came up with your results should be included in the answer.
- Always when asked to provide plots of something (time responses, curves etc.) you must provide your interpretation of the figure.
- Make sure that the assignment's learning objectives are reflected in your report.

- When your reports are evaluated, points are awarded for the general setup of the report. We are looking for a professional look of the report, where among others the following is considered:
  - Readability (appropriate font sizes, margins etc.) and consistence in styles applied (same look for body text, headings, captions etc. throughout the report).
  - Presentation of equations should be of an appropriate quality (do not copy/paste equations as screenshot pictures from the lecture handouts).
  - Plots and figures should be of good quality (no fuzzy looking screenshots in the report). You can always ask the teachers how you can export figures of good quality in Python/Matlab.
  - Figures and tables must have a label and caption (e.g. *Figure 1: plot of active power in respect to* ....).
  - If you are citing external material, you need to include a properly set-up reference list (you can use any citation style you prefer).
  - Does the report contain a conclusion section?
- You will also be awarded points for the general formulation of the report where focus is on:
  - The flow in the text when explaining how you came up with your answers and correct use of relevant terminology.
  - Interpretation and reflection of results when appropriate.
  - Grammar and spelling.

Guidelines for the Upload of the Report and Python Code

The report submission is carried out on DTU-LEARN where the following shall be uploaded:

- PDF version of your report, with your Python code in appendix.
- Your Python code used for solving the assignment (the *.py). The code shall include your comments where you explain the meaning behind the code.