Spring
2025

46705
Power Grid Analysis

Assignment 1

Subject: Power Flow Analysis and Security Assessment in Python

Issued: ***04.02.2025***
Submission date: ***17.03.2025***

Introduction

The purpose of this assignment is:

1. to write a Python program that carries out a power flow analysis of a given network
2. study different scenarios to analyse whether voltage or generator limits have been violated
3. to write a Python program capable of performing a contingency analysis of a given system

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 operational scenarios and interpret power flow results with respect to system security constraints.
- Describe the basic principle of power system security analysis, and apply steady state contingency analysis.
- Apply security assessment by extending the Python based power flow program so that it can carry out a contingency assessment of an arbitrary system.
- Use the program to perform security contingency analysis of the Nordic32 test system and analyze the results.

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.

Part I: Power Flow in Python

The aim of this part 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. The power flow program developed in part I of the assignment will be extended in part II to perform a security assessment.

```
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 `LoadNetworkData()` function that processes the system data for further use in the power flow algorithm.

Task 1 - 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`.

```python
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`: `[bus_nr, label, v_init, theta_init, code, kv_level, v_low, v_high]`

  `bus_nr` & `label` are bus identifiers; `v_init` & `theta_init` are the initial (or input depending on bus type) values for bus voltage and magnitude; `code` is the buscode, where 1=PQ, 2=PV & 3=ref., `kv_level` is the nominal voltage in kV and `v_low` & `v_high` are the lower and upper voltage limits of the bus in pu

- `load_data`: `[bus_nr, P_LD_MW, Q_LD_MW]`

  `P_LD_MW` & `Q_LD_MW` are the load active and reactive power in MW and MVAr specified using load convention

- `gen_data`: `[bus_nr, MVA_size, P_gen, P_max, Q_max, Q_min, X, X2, X0, Xn, grounding]`

  `MVA_size` is the MVA rating of the machine, `P_gen` & `P_max` are the active power injection and maximum active power capacity of the machine in MW, while `Q_max` & `Q_min` are the reactive power limits in MVAr

  `X,X2,X0,Xn` are positive, negative and zero sequence reactances specified on the generator base and `grounding` indicates whether the star point of the generator is grounded (these values are only relevant for performing fault analysis)

- `line_data`: `[fr_bus, to_bus, ID, R, X, B, MVA_rat, X2, X0]`

  `fr_bus, to_bus` are the busses that are connected through the line, `ID` is a unique identifier (primarily for parallel lines), `R,X,B` are the series impedances and shunt admittance in pu on the system base, `MVA_rat` is the maximal loading of the line.

  `X2,X0` are the negative and zero sequence reactances in pu on the system base (these values are only relevant for performing fault analysis)

- `tran_data`: `[fr_bus, to_bus, ID, R_eq, X_eq,  n_pu, ang_deg, MVA_rat,fr_con, to_con, X2, X0]`

  (Follows the same data structure as lines, additionally: `n_pu`: the pu turns ratio, `ang_deg`: is the phase shift)

  `fr_con, to_con` incidate the winding configuration of the transformer (1=Y, 2=Yn, 3=D) and `X2,X0` are the negative and zero sequence reactances in pu on the system base (these values are only relevant for performing fault analysis)

- `mva_base`: the system MVA base

- `bus_to_ind`: the mapping from bus numbers to the corresponding indices in the bus matrices and arrays

- `ind_to_bus`: 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.1 Write the `LoadNetworkData()` function, which automatically creates 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 2. The handouts from the lecture about Power Flow in Python contain examples of the data structures that are to be created automatically in `LoadNetworkData()`.

```
import numpy as np
import ReadNetworkData as rd

def LoadNetworkData(filename):
global Ybus, Sbus, V0, buscode, ref, pq_index, pv_index, Y_fr, Y_to, br_f,br_t,br_v_ind,br_Y,
    S_LD,  \
ind_to_bus, bus_to_ind, MVA_base, bus_labels, br_MVA, Gen_MVA,
br_id, br_Ymat, bus_kv, p_gen_max, q_gen_min, q_gen_max, v_min, v_max

#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 bus admittance matrix from elements in the line_data and trans_data
# Construct the branch admittance matrices Y_fr and Y_to used to determine the line flows
#############################################################################
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 2: Example of what the first few lines of `LoadNetworkData.py` (in Q.1) could look like.

Task 2 - 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.

```python
def PowerFlowNewton(Ybus,Sbus,V0,pv_index,pq_index,max_iter,err_tol,print_progress=True):
  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
  ␣␣if print_progress:
      print('The Newton Rapson Power Flow Converged in %d iterations!' % (n,))
  else:
  ␣␣if print_progress:
      print('No Convergence !!!\n Stopped after %d iterations without solution...' % (n,))
  return V,success,n
```

Table 3: Skeleton for the function `PowerFlowNewton()`. The part of the function colored red, represents functions which the students have to write.

As mentioned above, the function `PowerFlowNewton()` is the core of the power flow program. A Python skeleton for the function is provided in table 3.

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 3 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 3.

Q.2.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.2.b Write the function `CheckTolerance()` 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.2.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.2.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.2.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 3 - Construction of the `DisplayResults()` function

```
=====================================================================
|                          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.

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`

```python
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.

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.

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.4.1 Run the power flow program on the `Kundur_two_area_system.txt` system, display the results and examine wether any generator is overloaded or if any of the bus voltages is outside of the normal operating range, which in this particular case is assumed as $\pm 0.05pu$.

Q.4.2 Repeat the above, but now with one of the lines between busses 6 and 7 out of service (use `\\` in front of the line in `Kundur_two_area_system.txt` to comment the line out).

Part II: Static Security Assessment

Q.5 Essay Question (max 1 page): Briefly describe the concept of power system security and describe how the security assessment can be broken down into three major functions that are carried out in system operators' control centers.

Contingency Analysis in Python

The purpose of this part of the assignment is to create a Python program that carries out a contingency analysis of a given system. To achieve this, you will need to extend your Python code from part I as well as creating new functions for the contingency analysis. The core idea behind the Python program that carries out the contingency analysis is depicted as a flow-diagram in figure 1.

Table 6 shows an example script which has implemented some of the steps in the figure. Your Python code from part 1 is extended. The functions marked in red, represent the functions and library you need to write/modify.

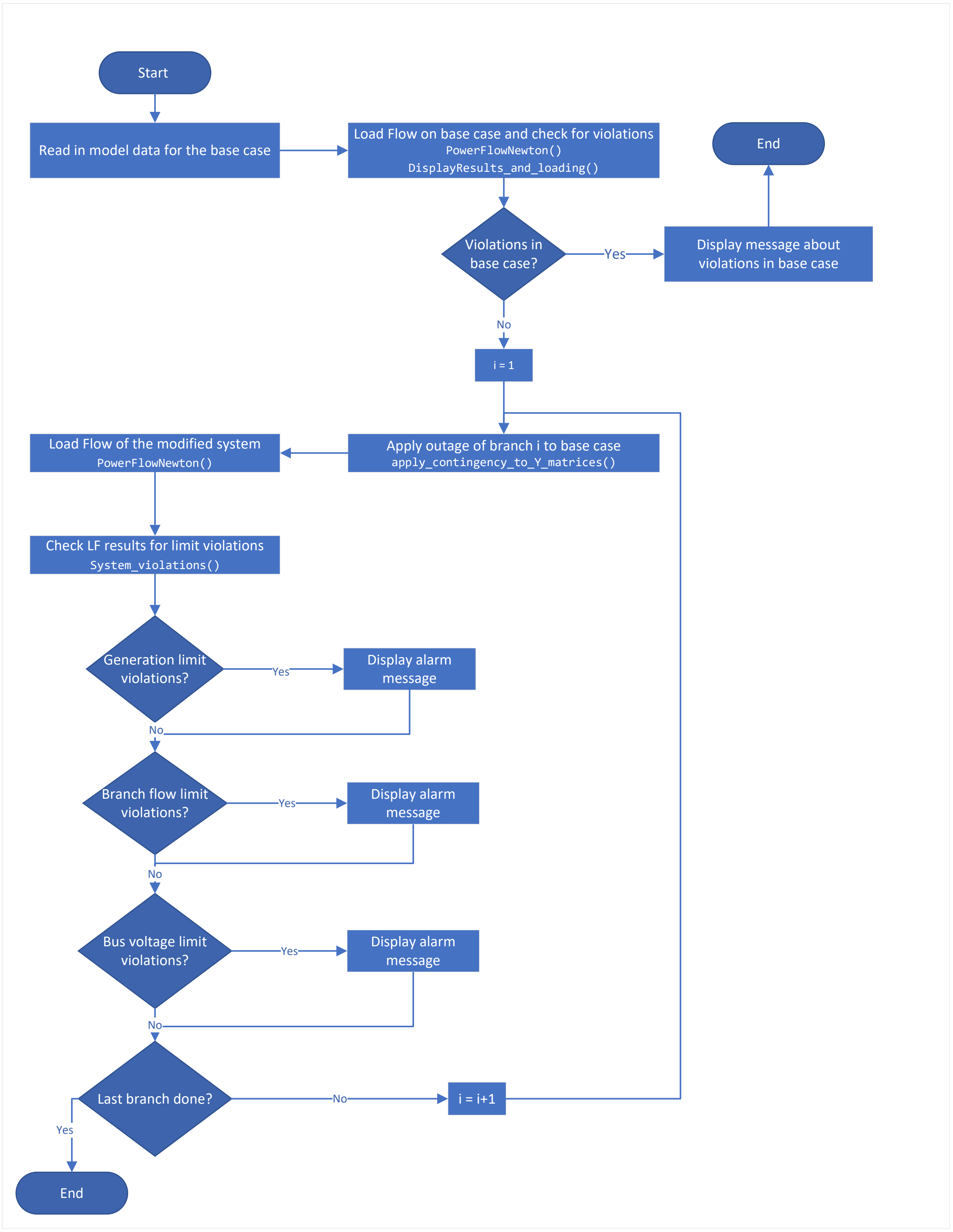
Start
Read in model data for the base case
Load Flow on base case and check for violations
PowerFlowNewton()
DisplayResults_and_loading()
End
Violations in base case?
Yes
Display message about violations in base case
No
i = 1
Load Flow of the modified system
PowerFlowNewton()
Apply outage of branch i to base case
apply_contingency_to_Y_matrices()
Check LF results for limit violations
System_violations()
Generation limit violations?
Yes
Display alarm message
No
Branch flow limit violations?
Yes
Display alarm message
No
Bus voltage limit violations?
Yes
Display alarm message
No
Last branch done?
No
i = i+1
Yes
End

Figure 1: Flow-diagram for a proposed structure of the contingency analysis program to be created in this assignment.

```
import numpy as np
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 = "./TestSystem4SA.txt"
lnd.LoadNetworkData(filename) # makes Ybus available as lnd.Ybus etc.
#%%
######################################################################
#   Part I:  Study the base case and display results (with % loading) #
######################################################################
V,success,n = pf.PowerFlowNewton(lnd.Ybus,lnd.Sbus,lnd.V0,lnd.pv_index,
lnd.pq_index,max_iter,err_tol)
if success: # Display results if the power flow analysis converged
pf.DisplayResults_and_loading(V,lnd)

#%%
#####################################################################
#  Part II:  Simplified contingency analysis (only branch outages)   #
#####################################################################
print('*'*50)
print('*             Contingency Analysis               *')
print('*'*50)

for i in range(len(lnd.br_f)): #sweep over branches
fr_ind = lnd.br_f[i]
to_ind = lnd.br_t[i]
br_ind = i
Ybr_mat = lnd.br_Ymat[i]
Ybus_mod,Yfr_mod,Yto_mod = \
apply_contingency_to_Y_matrices(lnd.Ybus,lnd.Y_fr,lnd.Y_to,
fr_ind,to_ind,br_ind,Ybr_mat)

str_status = '-'*63 + '\nTripping of branch {:} (bus {:} - bus{:})'.format(i+1,/
lnd.ind_to_bus[fr_ind],lnd.ind_to_bus[to_ind])

try:  #try the load flow, if it fails, display message
V,success,n = pf.PowerFlowNewton(Ybus_mod,lnd.Sbus,lnd.V0,lnd.pv_index,lnd.pq_index,/
max_iter,err_tol,print_progress=False)
except:
str_ = ' --> Load Flow error (Jacobian) when branch {:} (bus {:} - bus {:}) is tripped'
str_ = str_.format(i+1,lnd.ind_to_bus[fr_ind],lnd.ind_to_bus[to_ind])
print(str_status+ ' [CONVERGENCE ISSUES!]')
print(str_)
else:
if success: # Display results if the power flow analysis converged
violations = System_violations(V,Ybus_mod,Yfr_mod,Yto_mod,lnd)
if not violations: # no violations, print status and move on
print(str_status + ' [OK!]')
else: # if violation, display them
print(str_status + ' [Violations!]')
for str_ in violations:
print(str_)
else: #no convergence...
str_ = ' --> No load-flow convergence when branch {:} (bus {:} - bus {:}) is tripped'.
str_ = str_.format(i+1,lnd.ind_to_bus[fr_ind],lnd.ind_to_bus[to_ind])
print(str_status + ' [CONVERGENCE ISSUES!]')
print(str_)
␣␣␣
```

Table 6: Example script for carrying out contingency analysis in Python. The functions or library colored in red are the one you need to create/modify during the compared to part I.

Q.6 Modify your previous `DisplayResults()` function in such a way that it shows the loading levels of the generators (in respect to their MVA rating) and branches in percentages (the modified function is called `DisplayResults_and_loading()`). Furthermore, extend the function such that it can appropriately handle generation and load connected to one bus. The output from the modified function could resemble the one shown in table 7.

```
===================================================================================
|                                      Bus results                                |
===================================================================================
Bus       Bus          Voltage            Generation                    Load
nr      Label     Mag(pu)  Ang(deg)   P (pu)  Q(pu)  loading     P (pu)  Q(pu)
-----   ---------   -------  -------   ------- -------- -------   -------  -------
1011    BUS1011     0.989     1.02      -         -       -        2.000    0.800
1012    BUS1012     1.000     4.80     6.000     1.851    78.49%   3.000    1.000
1013    BUS1013     1.000     9.76     3.000     0.808    51.78%   1.000    0.400
1014    BUS1014     1.000    13.14     5.500    -0.543    78.95%     -        -
........ and so on ..........

==========================================================================
|                           Branch flow                                  |
==========================================================================
|Branch  From   To        From Bus Injection          To Bus Injection    |
|  nr    Bus    Bus    P (pu)   Q (pu)  loading     P (pu)   Q (pu) loading|
-----  -----  -----  -------  -------- -------     ------- ------- -------
1    1011   1013    -2.103   0.296    42.47%      2.149   0.013   42.98%
2    1011   1013    -2.103   0.296    42.47%      2.149   0.013   42.98%
3    1012   1014    -1.556   0.351    31.89%      1.591  -0.139   31.95%
4    1012   1014    -1.556   0.351    31.89%      1.591  -0.139   31.95%
........ and so on ..........
```

Table 7: Example showing which type of information could be printed out automatically when the modified function `DisplayResults_and_loading()` is called.

Q.7 Write the function `System_violations()` that automatically investigates whether any power flow limits or voltage limits have been violated. The function should return a list of strings that describe the found violations. Table 8 shows a template for the function.

Q.8 Write the function `apply_contingency_to_Y_matrices()` which takes as input the original admittance matrices (`Ybus`, `Y_fr` and `Y_to`) and returns the modified version of those matrices where the contingency (branch tripping) has been applied. A template for the function is provided in table 9.

```python
def System_violations(V,Ybus,Y_from,Y_to,lnd):
# Inputs:
# V = results from the load flow
# Ybus = the bus admittance matrix used in the load flow
# Y_from,Y_to = tha admittance matrices used to determine the branch flows
# lnd = the LoadNetworkData object for easy access to other model data

#store variables as more convenient names
br_f=lnd.br_f; br_t=lnd.br_t;   #from and to branch indices
ind_to_bus=lnd.ind_to_bus;      # the ind_to_bus mapping object
bus_to_ind=lnd.bus_to_ind;      # the bus_to_ind mapping object
br_MVA = lnd.br_MVA             # object containing the MVA ratings of the branches
br_id  = lnd.br_id              # (you have to update LoadNetworkData for this)


#line flows and generators injection....
S_to = V[br_t]*(Y_to.dot(V)).conj()         # the flow in the to end..
S_from = V[br_f]*(Y_from.dot(V)).conj()     # the flow in the from end
S_inj = V*(Ybus.dot(V)).conj()              # the injected power
SLD=lnd.S_LD                                # The defined loads on the PQ busses
S_gen = S_inj + SLD                         # the generator arrays

violations = []   #empty list that will store strings describing each violation

##################################################################################
#
#    YOUR CODE COMES HERE:
#
#    1. Check flow in all branches (both ends) and report if limits are violated
#    2. Check output of all generators and see if limits are exceeded
#    3. Check voltages on all busses and see if it remains between 0.9 and 1.1 pu
#
####################################################################################

return violations #return the list with description of all of the violations
␣
```

Table 8: Template for the function `System_violations()` which is to be written in Q.7.

```python
def apply_contingency_to_Y_matrices(Ybus,Yfr,Yto,fr_ind,to_ind,br_ind,Ybr_mat):
# input:
# The original admitance matirces: Ybus,Yfr,Yto
# The from and to end indices for the branch (fr_ind, to_ind)
# The indice for where the branch is in the branch list (br_ind)
# The 2x2 admittance matrix for the branch  Ybr_mat
##########################################################

# This is important, you must copy the original matrices
Ybus_mod = Ybus.copy()  # otherwise you will change the original Ybus matrix
Yfr_mod = Yfr.copy()    # when ever you make changes to Ybus_mod
Yto_mod = Yto.copy()    # using the .copy() function avoids this

##################################################################################
#
#    YOUR CODE COMES HERE:
#
#    1. Remove the branch from the Ybus_mod matrix
#    2. Remove the branch from the Yto and Yfr matrices
#
####################################################################################

return Ybus_mod,Yfr_mod,Yto_mod

␣
```

Table 9: Template for the function `apply_contingency_to_Y_matrices()` which is to be written in Q.8.

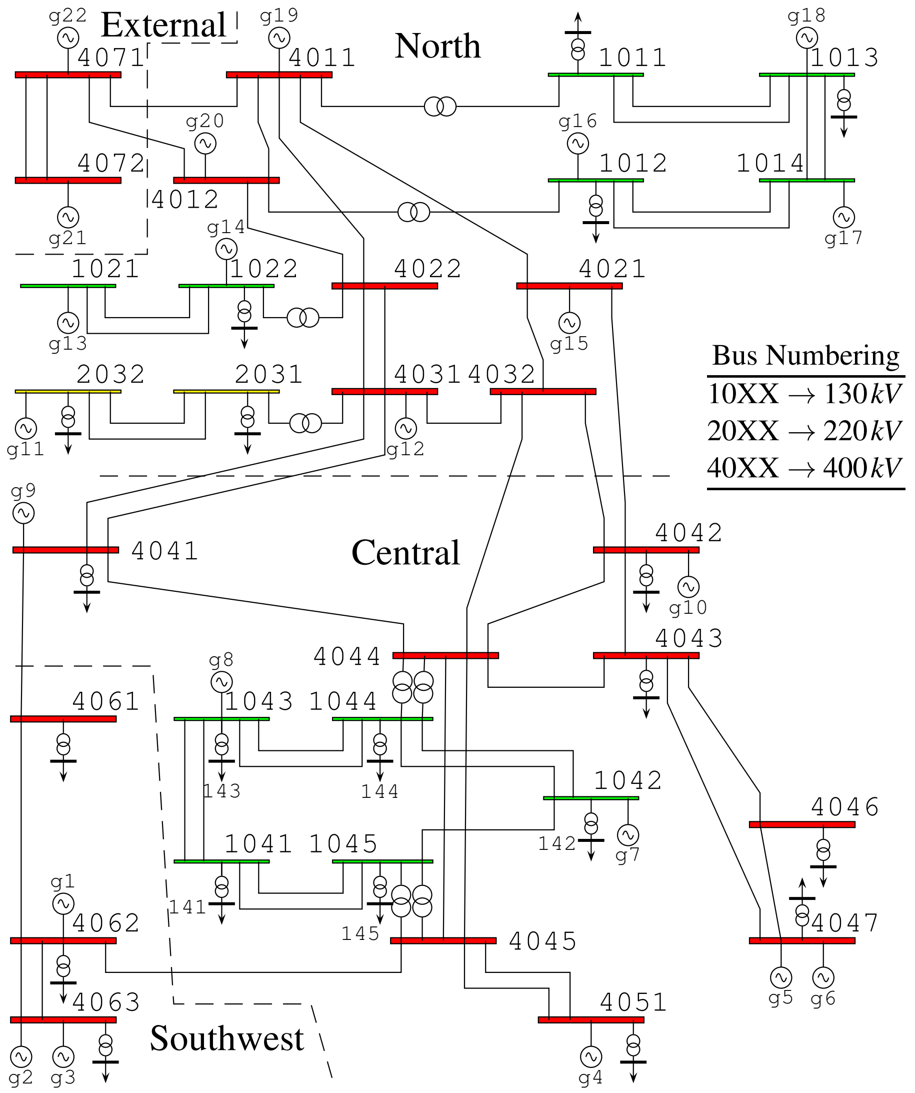

Figure 2: One-line diagram of the Nordic32 system studied in Q.9 and Q.10.

Q.9 Load the Nordic32 model (Nordic32_SA.txt) and run the load flow. Examine whether there are any flow limit or voltage limit violations in the base case. Figure 2 shows the topology of the Nordic32 system.

Q.10 Run the contingency analysis study the results. Give examples of contingency that result in no violations, contingencies that result in voltage limit violations, and contingencies that result in no load-flow convergence.

Bonus Pick one of the scenarios from Q.10 that was causing any violations and try to fix the problem assuming that you can change voltage setpoints of the generators, redispatch the active power generation and adjust transformer taps (assume a range of ±0.05 pu for all transformers). Briefly describe the measures that were succesful.

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.