DTU

CEE

Implementation of an on-line contingency screening method into real-time test platform

Yihao Sun

Master Thesis, August 2019, Kongens Lyngby, Denmark

DANMARKS TEKNISKE UNIVERSITET
Center for Electric Power and Energy (CEE)
DTU Electrical Engineering

Implementation of an on-line contingency screening method into real-time test platform

Dissertation, by Yihao Sun

Supervisors:

Supervisor No. 1, Hjortur Johannsson

Supervisor No. 2, Jakob Glarbo Møller

DTU - Technical University of Denmark, Kongens Lyngby - August 2019

Implementation of an on-line contingency screening method into real-time test platform

This thesis was prepared by:
Yihao Sun

Supervisors:
Supervisor No. 1, Hjortur Johannsson
Supervisor No. 2, Jakob Glarbo Møller

Center for Electric Power and Energy (CEE)
DTU Electrical Engineering

Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark

Tel: (+45) 4525 3500
Fax: (+45) 4588 6111
E-mail: cee@elektro.dtu.dk

Release date: 19.07.2019

Edition: Draft V01

Class: Public

Field: Electrical Engineering

Remarks: The dissertation is presented to the Department of Electrical Engineering of the Technical University of Denmark in partial fulfillment of the requirements for the degree of Master.

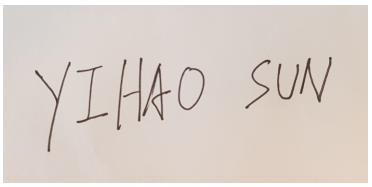
ISBN: 000-00-00000-00-0

Preface

This thesis is prepared at the Department of Electrical Engineering of the Technical University of Denmark in partial fulfillment of the requirements for acquiring the degree of Master in Engineering.

This dissertation summarizes the work carried out by the author during his Master project. It started on 28th 01 2019, and it was completed on 19th 07 2019.

The thesis is composed of seven chapters



Yihao Sun
19.07.2019

Acknowledgements

At first, I would like to thank my supervisor, **Hjortur Johannsson**. Every time when I have a misunderstanding or misconception about my project, he is always patient and will to lead a correct path to me. Thank you, professor **Hjortur**!

Meanwhile, since I need to build the entire system from the scratch, so in this process I did meet a lot of problems that I cannot solve them by myself. Then at the time like this, I always went to my mentor, **Jakob Glarbo Møller** for help. He is so nice and patient, and he always can help me solve the problem. Right now, all the objectives proposed in this master thesis project have been accomplished, and I need to say, I cannot achieve this without him. Thank you, **Jakob**!

Yihao Sun

Lyngby, Denmark, 2019

Table of Contents

List of Figures

Abstract

This paper mainly focuses on the fast contingency assessment screening method for the first-swing transient stability in a Nordic 20-Generators system. To realize that, a state of the art **Coupling-Strength Clustering method** will need to be implemented. As the crucial part of the entire project, this '**Coupling-Strength Clustering method**' will cluster all the machines into two groups, the Critical Machines (CMs) and Non-Critical Machines (NMs) at first, and then basing on this Clustering result, implement the **One Machine Infinite Bus (OMIB)** algorithm to analyze the transient stability situation. The advanced place of this method is that, rather than using the full-time domain simulation in PSS/E who is very time-consuming, the simulation in PSS/E will only need to be executed until three time cycles after the fault clearance. By having the short-time simulating data as the database, the entire program can take it and execute the algorithms for the analysis.

Then in the programming level, by using the '**Generalized Laplacian Algorithm**', the admittance matrix in pre-fault and post-fault can be generated automatically, and their results have been validated with the admittance matrices generated by PSS/E. Secondly, the physical meaning of the 'Coupling-Strength' is the potential relations between different machines. However, this relation can actually be quantified by the 'Coupling-Strength Coefficient' and be visualized by the heatmap. And after having the difference matrix of the 'Coupling-Strength Coefficient' between the post-fault and pre-fault, it can be applied to the **Depth-First Search (DFS)** technique for the machine clustering. The essence of the **DFS** is to simulate the eye recognition of humanity by analyzing the graph theory. To cope with this target, this paper also developed three different strategies to help the DFS do a better clustering. Afterwards, the **OMIB** algorithm will take this Clustering result and execute a first-swing transient stability analysis.

In summary, this paper has achieved a system, that combines the '**PSS/E short-time simulation**', '**Admittance matrix automatic generation**', '**Coupling-strength machine clustering**' and the '**OMIB method for transient stability analysis**'. The entire system is built in Python script from scratch, and so far, it is totally executable

and reliable. Furthermore, by validating the new system's results with the full-time domain simulation in PSS/E, it shows that the system can accomplish the goal in a high accuracy. Besides, this paper also developed a smarter code that can run every one fault in each branch automatically (different N-1 scenario each time), which can be treated as the first step of the online applying.

Acronyms

TSA: Transient Stability Assessment

PMUS: Phasor Measurement Units

OME: One Machine Equivalent

CMs: Critical Machines

NMs: Non-Critical Machines

DFS: Deep-First Search

EAC: Equal Area Criterion

acc: acceleration

dec: deceleration

I/O: Input/Output

OMIB: One Machine Infinite Bus

CHAPTER 1

Background Research

1.1 Motivation Interpretation

The advancement of humanity has no boundary, but this has not been without its price. The technology of mankind in recent hundreds of years has far exceeded that of the previous thousand years, while the damage to the nature from our species is unprecedentedly serious. When humans enjoy the energy coming from the fossil fuels, however, at the same time, they are suffering from the consequences brought by them. The emission of the carbon dioxide and the methane from the traditional power system and steel manufacturing lead to the global warming, which results in the sea-level rise and uncertain temperature variations. The direct result of the sea-level rise can lead to the disappearance of the coastal cities, and for the uncertain temperature variations, it will cause some unintended catastrophes such as typhoon or EI Nino and make them happen much more frequently.[1]

So for the perspective of stopping the deterioration of our environment, the renewable energy has become the future direction of the development. However, though the renewable energy is totally green and no greenhouse gases emission, the sources of the renewable energy is not very stable compared with the traditional fossil fuels. The first reason is that renewable energy is not always available. For instance, the wind source and solar source are both highly influenced by the season, which influences the energy production dramatically. Secondly, even in one single season, due to the its uncertainty, the fluctuation of the renewable energy sources results in instability of its energy output, which will effect the frequency in the grid. In summary, to realize the higher sharing of the renewable energy in the power system in the future, a stronger and robuster power system will be necessary to face the renewable source uncertainty as an external problem. In order to achieve this goal, a system that can quickly assess the internal contingencies from the power system and well visualized for daily prevention will be necessary. Therefore, this is the reason why an improved on-line contingency screening real time method for assessment is needed for now.

1.2 Project Description

1.2.1 Specialized Background Research

As the statement of this project, the basic idea for the transient stability assessment (TSA) is to get the data (complex bus voltage, current flows in each unit) from the snapshots of the system firstly, and then implement different contingencies scenarios as a N-1 situation, to test if the system can still maintain its stability in each case or not. The majority of current methods for the TSA is basing on the off-line computation, which takes too much time to cope with the future power system who has a fluctuating energy source. On the other hand, the time-domain simulation, along with ranking the respective contingencies by their adverse effects basing on the calculating results has been proposed in [2]. Furthermore, to some extent, a reduction of calculation amount will lead to a considerable decrease in the simulation time, which can be beneficial for the system, that more reaction time can be given to the observer to make a better decision. In [3], an approach to filter the non-severe disturbances by using transient energy function was proposed. In [4], basing on the homotopy method, the controlling unstable equilibrium point can be easier to identify. Also for the same purpose, the extended equal area criterion (EEAC) was proposed in [5], to filter the most stable cases after fault clearance and leave the computing resources for the remainings. In addition, a method [6] that basing on the artificial network and its corresponding phasor measurement units (PMUs) can directly detect the situation of the power system and execute the necessary measures.

1.2.2 Concentration of this project

In this project, according to the requirement of the statement, the approaches that proposed in [7] and [8] will be realized and improved if necessary. Basing on the model proposed in [8], the whole model can be divided into four comprehensive blocks: 'Snapshots Reading' (Phasor U and I, from wild area measurements or from tiny-time PSS/E simulation basing on the .raw file), 'Power System Initialing' (e.g. Y bus matrix, initial states, Initialize contingencies), 'Fast Contingency TSA' and the 'Publishing Results'. As the crucial part of the whole system, inside the 'Fast Contingency TSA', three major challenges need to be concerned. The first one is, when the fault has been cleared by the relay, the branch connection in the diagram will be different from the original one, which means that an algorithm that can

update the entire admittance matrix shall be mandatory. The second challenge is the analyzing for the loss of synchronism of a multi-machine power system. Since the stability analysis of a multi-machines' system is really complicated because machines themselves will influence each other by their own behaviors. Unless a One Equivalent Machine that can represent all the machines shows up in the system. This actually can be achieved by the technique named 'One Machine Infinite Bus', the 'OMIB' method. As the definition of OMIB in [7] and [9], that the OMIB can be treated as 'a transformation of the multidimensional multimachine dynamic equations into a single dynamic Equation', which will highly simplify the whole system and make the calculation possible. To achieve the OMIB, as the third challenge in this paper, the generators inside the power system need to be split into two groups, the Critical Machines (CMs) and the Non-Critical Machines (NMs), respectively. This can be achieved by the calculation of the 'Couple-Strength Coefficient', which reveals the potential relations between all the machines in service. The Critical Machines (CMs) in this paper are termed as those who hold the minus largest 'Couple-Strength Coefficient' difference between its post-fault and pre-fault. And the Non-Critical Machines (NMs) are those who do not belong to the CMs.

1.3 Project Learning Objectives

According to the project description in chapter 1.2, the learning objectives of this project have been divided into four main targets as followed:

(i) Understand and explain the instability mechanism regarding to first-swing transient stability in multi-machines system

(ii) Basing on the data coming from tiny time-domain simulation in PSS/E or the PMUs snapshots for the selected multi-machines system, develop the Python scripts for the 'Fast Contingencies Assessment' by implementing state of the art Machine Clustering method. Then double check the algorithm and the Clustering results with the PSS/E simulation to make sure it works well

(iii) Afterwards basing on the Clustering results from above, implement One Machine Infinite Bus (OMIB) method to evaluate the stability margin for selected multi-machines system

(iv) Finally achieve the automation of the assessment of all possible branches faults (all possible N-1 scenarios) in the system

After fulfilling these targets above, a capable system that combines the '**PSS/E short-time simulation**', '**Admittance matrix automatic generation**', '**Coupling-strength machine clustering**' and the '**OMIB method for transient stability analysis**' can be achieved and validated. Meanwhile, as the crucial part of this project, the Clustering algorithm will also be visualized through the Couple-Strength matrix, along with the adjacency matrix to tell that which part of generators has already been separated from the entirety. Finally, each N-1 scenario will go through this automatic assessment system to help test the its accuracy.

1.4 Thesis Guide

In this section, there will be the guidance for the reader to quickly get familiar with the paper structure. In Chapter 2 the '**Methodology Statement**' holds the particular orientations of every method applied in this project. In these orientations, both the mathematical expression and their physical meaning behind will be interpreted into details. Besides, the software used in this project, the 'PSS/E', will also be introduced to the reader as well as the hardware 'Phasor Measurement Units', the 'PMUs'. Furthermore, as this project needs a short-time loadflow simulation in PSS/E, the implementation of PSS/E will be introduced in Chapter 3, the '**Short-time PSS/E Simulation**'. And since in this project, instead of using the PSS/E interface, the functionalities of PSS/E are activating through the Python script command, therefore, Chapter 3 will mainly focuses on how these commands work and the meanings behind them.

Then the Chapter 4 and Chapter 5 are regarding to the **implementation of state of the art Clustering method** and **One Machine Infinite Bus (OMIB)**, respectively. Their mathematical expressions can be found in Chapter 2, however, in Chapter 4 and 5, there will be a further and deeper discussion regrading to the programming level about how to achieve the targets and encoding them into Python language. Besides, in Chapter 4, the Clustering results will be visualized and validated with PSS/E simulation. Then in Chapter 5 the Stability Margin will be calculated as the stability criterion, and it will be compared with PSS/E full-time domain simulation for the validation. After the targets in the Chapters above have been fulfilled, a smarter code will be developed to run all the possible N-1 scenarios

automatically in Chapter 6, the '**Assessment Automation with Every One Fault in Each Branch**'. Since the Clustering method and One Machine Equivalent are the crucial part of this project, and the data source of the same system will not actually influence the results, so the interconnection with PMUs will not be included in this project due to the limit of time.

CHAPTER 2

Methodology Statement

In this chapter the methodologies applied in this project will be elaborated both in the mathematical expressions and their corresponding physical meanings. While the implementations of these methodologies will be elaborated in the following chapters and at that moment they will be quoted directly without repeating.

2.1 Introduction to Nordic-20 Generators System

The Nordic-20 Generators System has been picked up as the investigative object for this project. This system has 20 generators, 74 buses and 102 branches (52 strippable branches among) in total. From Bus 1 to Bus 72 are the buses that connect to the load, and from Bus 901 to Bus 920 are the buses that connect to the generators. The rest buses from Bus 1011 to Bus 4072 are the buses connect to branch admittances and the branch shunts, while some of them also connect to the fixed shunts for the reactive power compensation. Furthermore, the branches between loads and general buses and the branches between generators and general buses are connecting through transformers for the voltage adjustments. Finally, all the data associated with this system are coming from the .raw file and .dyr file and can be easily checked in PSS/E.

2.2 Introduction to PSS/E

PSS/E, in short of the **Power System Simulator for Engineering**, is the powerful software to simulate the electrical power transmission networks both in steady state and dynamic conditions. In the initial stage of this project, PSS/E is providing the upstream data for the algorithms basing on the .raw file and .dyr file. Unlike the full-time domain simulation, PSS/E here will only need to simulate until three time cycles after the fault has been cleared due to the actual need, which is less

than 1.2 sec in total.

The upstream data includes machine rotor angles, machine speed deviations from nominal value, complex terminal voltage of all the buses and machine power flows, which also can be achieved through the snapshots from Phasor Measurement Units (PMUs). Since the initial situation for a certain system shall be exactly the same, therefore there is no difference in the clustering results when the upstream data is coming from PSS/E or PMUs. So before the application of PMUs, the short-time simulation results from PSS/E can be treated as the 'simulation' of the PMUs' snapshots. Besides, instead of using the PSS/E interface, every functionalities activated in PSS/E are achieved by the commands through the Python script, which is more directly and effectively compared with manual operation in the interface. Because in the further research a automation algorithm that can run all the possible branch faults will be achieved and each new branch fault will need a PSS/E tiny time simulation respectively. The detailed parameter settings will be elaborated in Chapter 3, the **'Short-Time PSS/E Simulation'**, along with the interpretation and discussion.

2.3 Introduction to PMUs

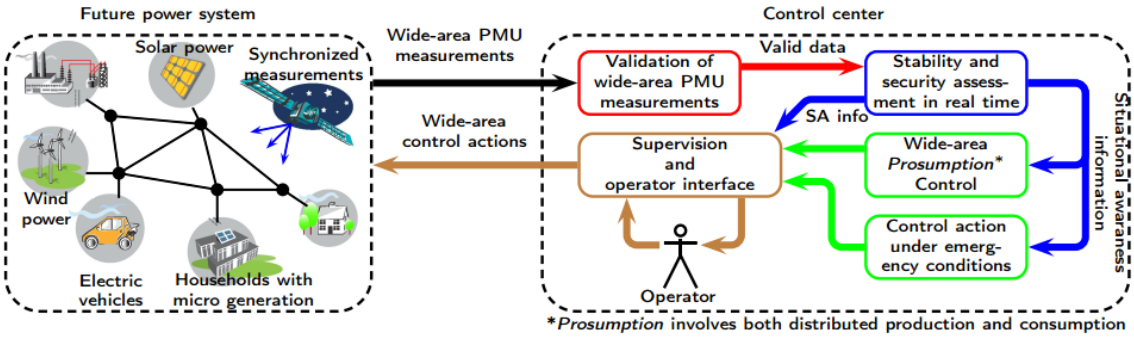


Figure 2.1: Configuration of future power system with PMUs

PMUs, in short of **'Phasor Measurement Units'**, is the general name for the hard devices that monitor both the magnitude, complex angle of the voltage and injected current or even the power flow in the whole network. As what can be observed from the Figure.2.1 above the PMUs can almost be suitable for any kinds of electrical equipment and all these signals can be sent to the control part through the GPS in the synchronised satellites. The commercial PMUs can normally report

the measurement in order of 30-60 times per second, which is apparently high temporal resolution compared with the traditional SCADA system. The high temporal resolution of PMUs can dramatically enhance the monitor ability of the power system controller and help them make decision much more precisely.

In this project, the PMUs is not actually applied but be represented by the tiny-time simulation in PSS/E. Since different data source of one single system shall lead to the same result, no matter for the clustering result or the stability assessment.

2.4 Introduction to Admittance Matrix Generation

The admittance matrix is something significant for the analysis of the power system in both the load flow, transient stability, voltage stability and so on. Therefore, creating the admittance matrix correctly and properly for a selected system is a very essential target for the whole project. Since the admittance matrix is the basestone for the further implementations, therefore this paper put it here as the beginning of the technical methodologies introduction. Furthermore, as the admittance matrix is widely used, so the method to achieve it shall be generalized enough to cope with the power systems that holds different complexities. Basing on the actual needs, the '**Generalized Laplacian algorithm**' is proposed here to solve the problem. Also, since what are counting in the further Clustering method are actually the internal nodes, then the reduced admittance matrix will also need to be achieved at the same time.

2.4.1 Generalized Laplacian Algorithm

In this section, the '**Generalized Laplacian algorithm**' method, that can generate the admittance matrix automatically and update the admittance matrix easily after the fault clearance will be elaborated as followed. The entire method is basing on the theory of Linear Algebra and its first step can be described as:

$$J^T \cdot Y_{prim} \cdot J = Y_{bus\_infant} \tag{2.1}$$

, where the matrix J is the incident matrix and the matrix Y_{prim} is the admittance matrix that only holds elements in its diagonal. This method can be validated by

the theory in linear algebra as the aggregation of the linear equations that describes the network configuration.

Incident Matrix J

The matrix J in Equation [2.1] is the incident matrix that shows the connection situations between different buses. The components regarding to its row shall be the entire branches in this system and the components regarding to its column shall be the entire buses in the system. The profile of this matrix can be described as below in Figure.2.2.

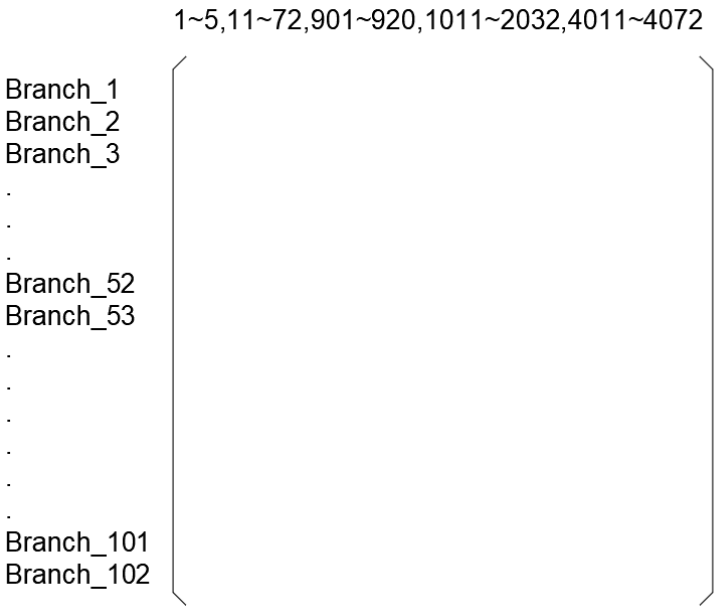


Figure 2.2: J matrix sketch map

The matrix initializes with a 102x74 zero matrix who corresponds to 102 branches and 74 buses in the system, respectively. And for the convenience of implementation, from the Branch_1 to Branch_52 are setting as the normal branches and from Branch_53 to Branch_102 are setting as the transformer branches. In the normal branches, the elements are filling row by row from Branch_1 to Branch_52 by setting the 'from_bus'-regarding elements to 1 and 'to_bus'-regarding elements to -1 in their corresponding column. While, when fill the transformer branches, the elements regarding to the 'from_bus' are still setting to one, however, those

elements regarding to the 'to_bus', shall be setting as $-1/ratio$, where the ratio is that the number of primary winding divides by the secondary winding. Afterwards, the incident matrix that can describe all the existing branches in the system network is accomplished.

Diagonal Admittance matrix Y_{prim}

$$\begin{bmatrix} Y_{B\_1} & 0 & \cdots & 0 \\ 0 & Y_{B\_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Y_{B\_102} \end{bmatrix} \tag{2.2}$$

As the second part constituting the Equation [2.1], the matrix Y_{prim} shall be a 102x102, squared matrix that only holds the branch admittance in its diagonal. And to cope with the incident matrix J, the Y_{prim}'s row and column shall have the same order of all the branches as in J matrix. Besides, due to matter of mathematical expression, the branch admittance here in Equatiion [2.2] can only account for the branch admittances and the transformer admittances.

After constituting the J matrix and Y_{prim} matrix, the Equation [2.1] can be calculated and the result there is a 74x74 matrix. However, since the branch shunts, fixed shunts and loads have not been included yet, so the admittance matrix in Equation [2.1] is called Y_{bus_infant} and its 'elder' version will be developed in the further sections.

Branch Shunt & Fixed Shunt

As the branch shunt and fixed shunt are both between their corresponding buses and ground, so they shall only occur in the diagonal of the admittance matrix since they can only be described in the place where the 'from_bus' and 'to_bus' are the same.

As a branch shunt, since half of it is connecting to the 'from_bus' and the other half is

connecting to the 'to_bus', so the way of adding it can be described as Equation [2.3]:

$$\begin{cases} Y_{bus\_infant}\left[from\_bus, from\_bus\right] += j \cdot \frac{B_{ch}}{2} \\ Y_{bus\_infant}\left[to\_bus, to\_bus\right] += j \cdot \frac{B_{ch}}{2} \end{cases} \tag{2.3}$$

, where the 'from_bus' and 'to_bus' are the indices regarding to corresponding bus locations, and the 'B_{ch}' is the branch shunt admittances.

And for the fixed shunt, as it is only connecting to a certain bus, so the way of adding it can be described as Equation [2.4]:

$$Y_{bus\_infant}\left[loc, loc\right] += G + j \cdot B_f \tag{2.4}$$

, where the 'loc' is the index regarding to the corresponding bus and the G and B_f are the real part and imaginary part of the fixed shunt, respectively. And after the supplementing the branch shunt and fixed shunt through Equation [2.4], this admittance matrix can be called Y_{bus_junior} in order to distinguish.

Load Representation

After achieving the Y_{bus_junior}, just supplement the load admittance to the Y_{bus_junior} then the admittance matrix can be complete. However, since the load information is given by power flow instead of admittance value directly, then a special approach to calculate the load admittance is applied as followed. Firstly, the current injection will need to be calculated as:

$$I_{inj} = Y_{bus\_junior} \times V_{bus} \tag{2.5}$$

, where the V_{bus} is the vector that holds every bus voltage in p.u.value, which can be achieved by the simulating results in PSS/E or snapshot from PMUs. Meanwhile, as Equation [2.5] is a matrix multiplication, and the scale of Y_{bus_junior} is 74x74 while for V_{bus} is 74x1, as the result the I_{inj} shall be a 74x1 vector.

Then after calculating this I_{inj}, the load admittance can be described as:

$$Y_{Load} = I_{inj}/V_{bus} \tag{2.6}$$

what is describing in Equation [2.6] is actually a vector division, as the result the Y_{Load} keeps its scale, 74x1. Furthermore, since the calculation process does not influence the order of what the row and column correspond to, so the indices of Y_{Load} shall also regard to the entire buses and it can be described as Figure.2.3, where the 'area 1' regards to the load buses, 'area 2' regards to the generators and 'area 3' regards to the rest of buses.

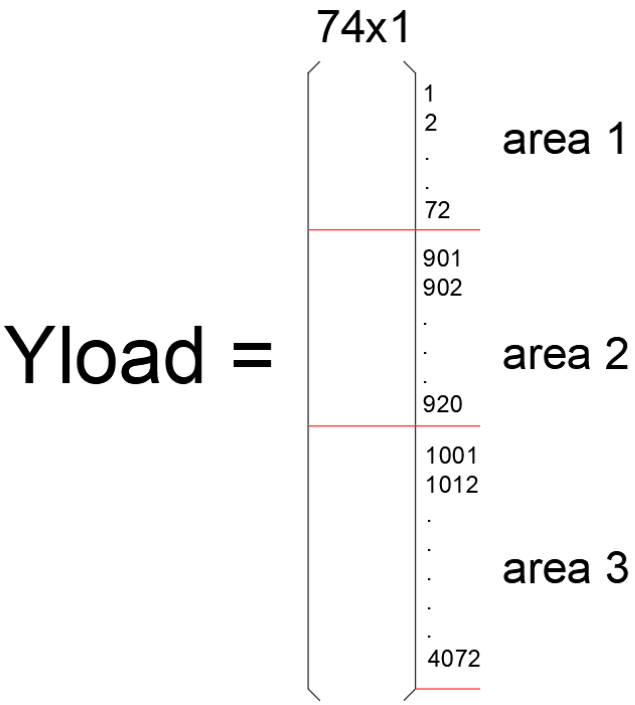


Figure 2.3: Load admittance matrix structure

Finally, to complete the admittance matrix, the load will be represented by taking the elements of the 'area_1' and 'area_3' from the calculated Y_{Load} vector. Since the loads also have the same attribute as the branch shunt and fixed shunt about their connection ways, therefore the load admittances only need to be added into corresponding place in the diagonal of the Y_{bus_junior}. Then the complete admittance matrix can be achieved by:

$$Y_{bus} = Y_{bus\_junior}\left[index, index\right] + Y_{Load}\left[index\right] \tag{2.7}$$

, where the index regards to the matrix index of the 'area_1' and 'area_3'. After the achievement of Equation [2.7], the admittance matrix construction is already complete.

2.4.2 Reduced Admittance Matrix Components

The reduced admittance matrix is widely used when the load flow calculation only focuses on the internal nodes. However, to generate a reduced admittance matrix, some extra components that belong to the extended admittance matrix will become something necessary, since those are where the load flow of the internal nodes operate. But before 'shrinking' the load flow calculation only onto the internal nodes, an extension from the general nodes to the internal nodes needs to be achieved at first.

Extended Admittance Matrix Introduction

According to the knowledge from [7] and [8], when the load flow counting the internal nodes, then the admittance matrix shall be 'enlarged' to its extended version, the 'extended admittance matrix'. The extended admittance matrix can be described as an aggregation of four different matrices: the **augmented admittance matrix** 'Y_{aug}', the **squared machine transient admittance matrix** 'Y_{gg}', the **machine & buses connection situation matrix** 'Y_{bg}' and its transpose.

$$Y_{ex} = \begin{bmatrix} Y_{aug} & Y_{bg} \\ Y_{bg}^{T} & Y_{gg} \end{bmatrix} \tag{2.8}$$

, where the augmented admittance matrix is the complete admittance matrix Y_{bus} plus the generator transient admittances in its corresponding diagonal location, which can be described as:

$$Y_{aug} = Y_{bus}\left[index_{gen}, index_{gen}\right] + \frac{1}{j \cdot X_{d}'\left[nr\_G\right]} \tag{2.9}$$

, where the $index_{gen}$ relates to the buses connecting to the generators, and the nr_G regards to their sequential orders.

Y_{bg} & Y_{gg} Construction

Y_{bg} is the matrix that represents the connections between the machine internal nodes and the respective terminal buses. Then constructing its mathematical

expression basing on its physical meaning, which can be described as Figure.2.4. The rows of Y_{bg} regard to the entire branches in the system, and the columns of Y_{bg} are related to different generators. The sequential order of the row follows the order of the complete admittance matrix Y_{bus} in Equation [2.7], which means Y_{bg} can also be divided into 3 areas that correspond to loads, generators and the rest parts, respectively. The generators' order in the column just starts from generator 1 to generator 20.

Then according to its physical meanings, the non-zero elements inside Y_{bg} shall only occur into 'area_2', which means the 'area_1' and 'area_3' are full of zero. Since each generator has a unique, corresponding terminal bus connection, and the sequential order of the rows in area_2 also arranged from Bus 901 to Bus 920, which regards to from Generator 1 to Generator 20. So the non-zero elements in 'area_2' only occur in its 'diagonal' and shall be the negative transient admittance values, $-1/jX'_{d_i}$, where i regards to different generators.

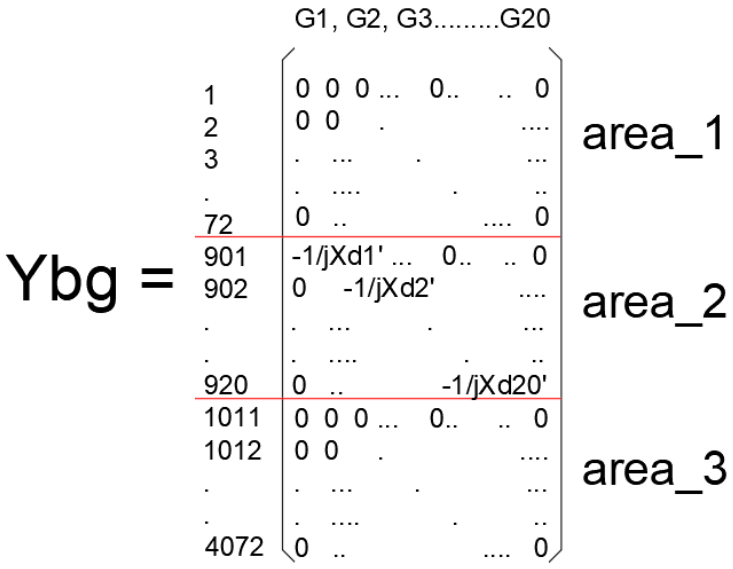


Figure 2.4: Load admittance matrix structure

Afterwards it is the construction of Y_{gg}, who holds the transient admittance in its diagonal. The scale of this squared matrix shall be equal to the number of generators, which in our case, 20. And what the sequential order of its rows and columns represents shall be equal to both the Y_{bus} and the Y_{bg}. Basing on these ideas, the matrix form of Y_{gg} can be expressed as Equation[2.10]:

$$Y_{gg} = \begin{bmatrix} 1/jX'_{d_1} & 0 & \cdots & 0 \\ 0 & 1/jX'_{d_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/jX'_{d_{20}} \end{bmatrix} \tag{2.10}$$

The mathematical validation for both the structure and element values of Y_{aug}, Y_{bg} and Y_{gg} can be achieved by the aggregation of the network equations, however, the validation process will not be provided in this paper due to its universality.

2.4.3 Reduced Admittance Matrix Construction

According to the knowledge in [8], the relations between extended admittance and reduced admittance matrix can be described as what shows in Equation [2.11] and Equation [2.12]:

$$\begin{bmatrix} 0 \\ I_{tr} \end{bmatrix} = \begin{bmatrix} Y_{aug} & Y_{bg} \\ Y_{bg}^T & Y_{gg} \end{bmatrix} \times \begin{bmatrix} V_{bus} \\ E^{'} \end{bmatrix} \tag{2.11}$$

$$I_{tr} = Y_{red} \times E^{'} \tag{2.12}$$

Then by decomposing the Equation [2.11] and replacing the V_{bus} with all other components, the Equation [2.11] can be derived into Equation [2.13] as below:

$$I_{tr} = (Y_{gg} - Y_{bg}^T \times Y_{aug}^{-1} \times Y_{bg}) \times E^{'} \tag{2.13}$$

, which compared with Equation [2.12], the reduced admittance matrix can be defined as:

$$Y_{red} = Y_{gg} - Y_{bg}^T \times Y_{aug}^{-1} \times Y_{bg} \tag{2.14}$$

2.4.4 Admittance Matrix Updating in the Post-fault

As there is also branch tripping issue happens in this project, so the whole network diagram will be different after the fault clearance, as well as the majority of matrices introduced above. Therefore a update method of the all the admittance matrices will be necessary. Fortunately, this is also the advantage of the 'Generalized Laplacian algorithm', that only a tiny alter can achieve this 'cumbersome' target. As we know, when a branch is tripped, its impedance can be treated as 'infinite', while, its corresponding admittance, on the other hand, will be 1 divide by infinite, which is a zero. And since the branch tripping issue will only influence the behaviors of branch admittance, transformer admittance and the branch shunt admittance. So the approach of how to apply this 'tiny' alter, is just by setting the diagonal locations' elements in the Y_{prim} matrix that corresponds to the tripped buses to zero, as well as the related branch shunt admittance values when generating the Y_{bus_infant} through Equation [2.1], [2.2] and [2.3], which can be described as the Equation [2.15] below:

$$\begin{cases} Y_{prim}\left[trip_{index}, trip_{index}\right] = 0 \\ B_{ch}\left[trip_{index}\right] = 0 \end{cases} \tag{2.15}$$

,where the $trip_{index}$ corresponds to the tripped branch. And by updating the Y_{bus_infant} with Equation [2.15] and keeping the rest of steps the same as previous, then all the admittance matrices can be updated successfully.

So far, the approach to calculate the reduced admittance matrix has been elaborated completely, both for the pre-fault and post-fault situation. The implementation details will be described in Chapter 5, along with the implementation of the Clustering method.

2.5 Introduction to State of the Art Clustering Method

In this section the state of the art Clustering method will be officially introduced. The elaboration will go through from the basic preparation calculation at first, then introduce the active power representation of the one machine equivalent (OME) and discuss its necessity. Afterwards the term 'Coupling-Strength' will be

interpreted into details from its physical meanings to its mathematical expression. In final, the 'Depth First Search', the 'DFS' algorithm will be introduced and its three different corresponding strategies that applied in this project will also be elaborated.

2.5.1 Coupling-Strength Algorithm

Preparation Calculation

Since the Couple-Strength Algorithm mainly focuses on the relative relations between each pair of machines in the system, so as the first step, the associated parameters of generators shall be reformulating into a relative form. However, from other perspective this relative relation between a pair of machines can also be treated as a OME (one machine equivalent) between this two machines. Therefore, calculate the relative rotor angle, rotor speed and the aggregated inertia coefficient for this OME at first respectively:

$$\begin{cases} \phi_{ij} = \delta_i - \delta_j \\ d\phi_{ij} = \omega_i - \omega_j \\ M_{ij} = \frac{M_i M_j}{M_i + M_j} \end{cases} \tag{2.16}$$

Then basing on the dynamic description of this OME, derive the mechnical power and active power for it:

$$P_{m,ij} = \frac{(M_j P_{m,i} - M_i P_{m,j})}{M_i + M_j} \tag{2.17}$$

$$P_{e,ij} = \frac{(M_j P_{e,i} - M_i P_{e,j})}{M_i + M_j} \tag{2.18}$$

, where the δ, ω, P_m and P_e can be achieved through the simulating data from PSS/E, and the M can be obtained from the .dyr file directly.

Active Power Reformulation

Afterwards, the Equation [2.18] above needs to be reformulated into a new form in Equation [2.19], which takes the ϕ_{ij} as the variable:

$$P_{e,ij}\left(\phi_{ij}\right) = P_{c,ij} + P_{max,ij} \cdot sin(\phi_{ij} - \nu_{ij}) \tag{2.19}$$

The reason of this reformulation will be interpreted into details in the next section, while in this section there will only be an presentation of how to achieved the Equation [2.19]. Besides, there is one thing need to be mentioned that in the orientation paper of this project, the ϕ_{ij} in Equation [2.16] is calculated as the difference between the machine rotor angles. However, due to the complexity difference in the generator model applying, in this project the ϕ_{ij} will be represented by the difference of the internal voltage angles between machines, the $\Delta angle(E_{ij}^{'})$, which is a more generalised way used in the load flow calculation.

Then, firstly, the 'constant' term $P_{c,ij}$ is expressed as:

$$P_{c,ij} = \frac{[M_j {E_i}'^2 Y_{ii} cos(\theta_{ii}) - M_i {E_j}'^2 Y_{jj} cos(\theta_{jj})]}{M_i + M_j} \tag{2.20}$$

, where both the M and $E^{'}$ are using the absolute values, and the Y and θ are the absolute admittance value and its corresponding impedance angle from the reduced admittance matrix calculated before.

After that, as the second term of Equation [2.19], the $P_{max,ij}$, can be understood as the absolute value of the apparent power of this OME, which shall be expressed as the combination of active power and reactive power:

$$P_{max,ij} = \sqrt{C_{ij}^2 + D_{ij}^2} \tag{2.21}$$

, where the C_{ij} and D_{ij} are expressed as the partial OME active power and reactive power respectively, which can be expressed as:

$$\begin{cases} C_{ij} = + \frac{[M_j \sum_{k=1,k\neq i}^{m} E_i^{'} E_k^{'} Y_{ik} cos(\delta_j - \delta_k - \theta_{ik}) - M_i \sum_{k=1,k\neq j}^{m} E_j^{'} E_k^{'} Y_{jk} cos(\delta_i - \delta_k - \theta_{jk})]}{M_i + M_j} \\ D_{ij} = - \frac{[M_j \sum_{k=1,k\neq i}^{m} E_i^{'} E_k^{'} Y_{ik} sin(\delta_j - \delta_k - \theta_{ik}) + M_i \sum_{k=1,k\neq j}^{m} E_j^{'} E_k^{'} Y_{jk} sin(\delta_i - \delta_k - \theta_{jk})]}{M_i + M_j} \end{cases} \tag{2.22}$$

Then it is the final term, the ν_{ij}. The physical meaning of this angle is described in the vector diagram in Figure.2.5, where shows that the φ_{ij} is the impedance angle of this OME, and the ν_{ij} is actually the 'minus complement angle of φ_{ij}'. So the mathematical expression of ν_{ij} shall be described as:

$$\nu_{ij} = -arctan(\frac{C_{ij}}{D_{ij}}) \tag{2.23}$$

Besides, basing on their relations shows in Figure.2.5, that:

$$\phi_{ij} + (-\nu_{ij}) = \frac{\pi}{2} \tag{2.24}$$

,then the sinusoidal part of the Equation [2.19] can also be derived into a more easier understanding form:

$$\begin{aligned} sin(\phi_{ij} - \nu_{ij}) &= cos(\frac{\pi}{2} + \nu_{ij} - \phi_{ij}) \\ &= cos(\varphi_{ij} - \phi_{ij}) \\ &= cos(\phi_{ij} - \varphi_{ij}) \end{aligned} \tag{2.25}$$

, where the ϕ_{ij} are the machine internal voltage difference and the φ_{ij} is the impedance angle. So the purpose of the ν_{ij} here in Equation [2.19] is just for coordinating between the cosinusoidal form in Equation [2.25] and the sinusoidal form in Equation [2.19]. Since in the way of Equation [2.23] the angle can be easier expressed, so the sinusoidal part in Equation [2.19] will still be the applying form, while the Equation [2.24], Equation [2.25] and the Figure.2.5 can be its validation.

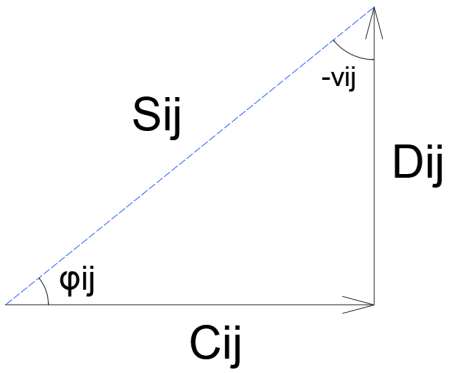


Figure 2.5: Vector diagram of the relation in Equation [19]

Coupling-Strength Calculation for the Pre-fault

As the most crucial part of the entire algorithms, before expressing the couple-strength coefficient into its mathematical form, a description of its physical meaning needs to be elaborated here. The couple-strength can reveal the potential relations between each pair of machines by digitizing this relation into the couple-strength coefficient. The definition of the couple-strength coefficient of each OME is the difference between its available dissipated energy and its extra kinetic energy getting from the un-equilibrium in the during-fault stage. This is also the reason why the way of how to calculate the couple-strength coefficient is different between the pre-fault and the post-fault, that in the pre-fault the 'extra kinetic energy' has not been generated yet.

Then basing on the idea above, draw the schematic diagram for a OME regarding to one pair of machines in the pre-fault stage for a better understanding in Figure.2.6:

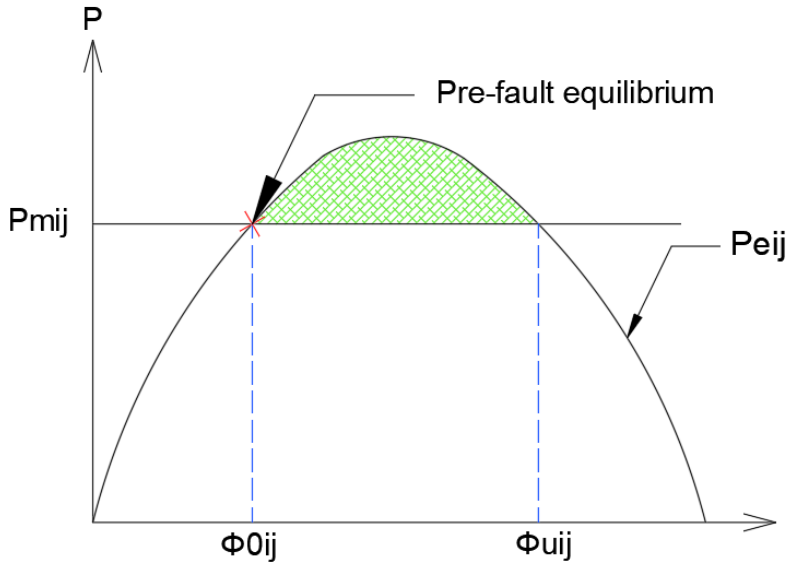


Figure 2.6: Schematic diagram for the couple-strength in pre-fault for a OME

From the Figure.2.6 it can be observed that the operating point of this OME is stabilizing in the red cross $\phi_{0,ij}$, where is the equilibrium between the OME mechanical power and its active power. Since in the pre-fault the 'extra kinetic energy' has not been generated yet, and all the generators shall be working in their

rated rotor speed, which in p.u.system, are both 1. So the mathematical expression of this term can be represented as Equation [2.26] and [2.27]:

$$E_{kin,ij} = \frac{M_{ij}}{2\omega_0} (d\phi_{ij})^2 \tag{2.26}$$

, which in pre-fault:

$$d\phi_{ij} = 0,\ E_{kin,ij} = 0 \tag{2.27}$$

, where the ω_0 is 1 in p.u.system and the rest terms can be found in Equation [2.16].

Afterwards it is the calculation of the pre-fault available dissipated energy. As what has been shown in the Figure.2.6, to accelerate the internal voltage angle of this OME from $\phi_{0,ij}$ to its unstable critical angle $\phi_{u,ij}$, the available dissipated energy that equals to this green area will need to be consumed at first, which means this green area is the entire available dissipated energy in the pre-fault. Basing on this theory and the schematic diagram, its mathematical expression can be described as:

$$E_{dis,ij} = \int_{\phi_{0,ij}}^{\phi_{u,ij}} P_{e,ij} (\phi_{ij}) - P_{m,ij}\, d\phi_{ij} \tag{2.28}$$

, by taking $P_{e,ij}$ from Equation [2.19], Equation [2.28] can be expressed as:

$$E_{dis,ij} = \int_{\phi_{0,ij}}^{\phi_{u,ij}} [P_{c,ij} + P_{max,ij} \cdot sin (\phi_{ij} - \nu_{ij}) - P_{m,ij}]\, d\phi_{ij} \tag{2.29}$$

Then simplify the this integral form into an equation form, which can be expressed as:

$$E_{dis,ij} = (P_{c,ij} - P_{m,ij})\phi_{ij}|_{\phi_{0,ij}}^{\phi_{u,ij}} - P_{max,ij} cos(\phi_{ij} - \nu_{ij})|_{\phi_{0,ij}}^{\phi_{u,ij}} \tag{2.30}$$

, where its inside components are coming from Equation [2.20], [2.21], [2.22], [2.23]. Therefore, the only thing that needs to be achieved is the upper limit and down limit for this integral. Since what can be observed from Figure.2.6, the upper and down limit are the two equilibrium points between $P_{m,ij}$ and $P_{e,ij}$, which can be expressed as:

$$P_{c,ij} + P_{max,ij} \cdot sin (\phi_{ij} - \nu_{ij}) = P_{m,ij} \tag{2.31}$$

, by solving this equation, the down limit can be expressed as:

$$sin(\phi_{ij} - \nu_{ij}) = \frac{(P_{m,ij} - P_{c,ij})}{P_{max,ij}} \tag{2.32}$$

$$\phi_{ij} - \nu_{ij} = arcsin(\frac{(P_{m,ij} - P_{c,ij})}{P_{max,ij}}) \tag{2.33}$$

$$\phi_{0,ij} = arcsin(\frac{(P_{m,ij} - P_{c,ij})}{P_{max,ij}}) + \nu_{ij} \tag{2.34}$$

This is also the reason why there is a reformulation from Equation [2.18] to Equation [2.19]. Because in Equation [2.19], it takes Pe_{ij} as a function of variable ϕ_{ij}, which can help calculate the upper and down limit for the integral in the Equation [2.29] and [2.30]. Otherwise only the format of Equation [2.18] cannot find a solution for the corresponding angles.

And then, since $\phi_{u,ij}$ is the supplementary angle of $\phi_{0,ij}$, so the unstable critical angle of this OME can be calculated as:

$$\phi_{u,ij} = \pi - \phi_{0,ij} \tag{2.35}$$

In final, for the coupling-strength coefficient in the pre-fault, that basing on all the discussion above, can be expressed as the entire available dissipated energy calculated in Equation [2.30]:

$$H_{pre,ij} = E_{dis,ij} \tag{2.36}$$

Coupling-Strength Calculation for the Post-fault

Though the theory of how to calculate the coupling-strength in the post-fault shall be the same with the pre-fault condition, however, three calculation details need to be changed to coordinate the post-fault condition. Also, for a better understanding of the interpretation, a schematic diagram is presented as followed in Figure.2.7, where the black line, grey line and green line correspond to the pre-fault, during-fault and post-fault, respectively. And the red curve is the operating process after

the pre-fault:

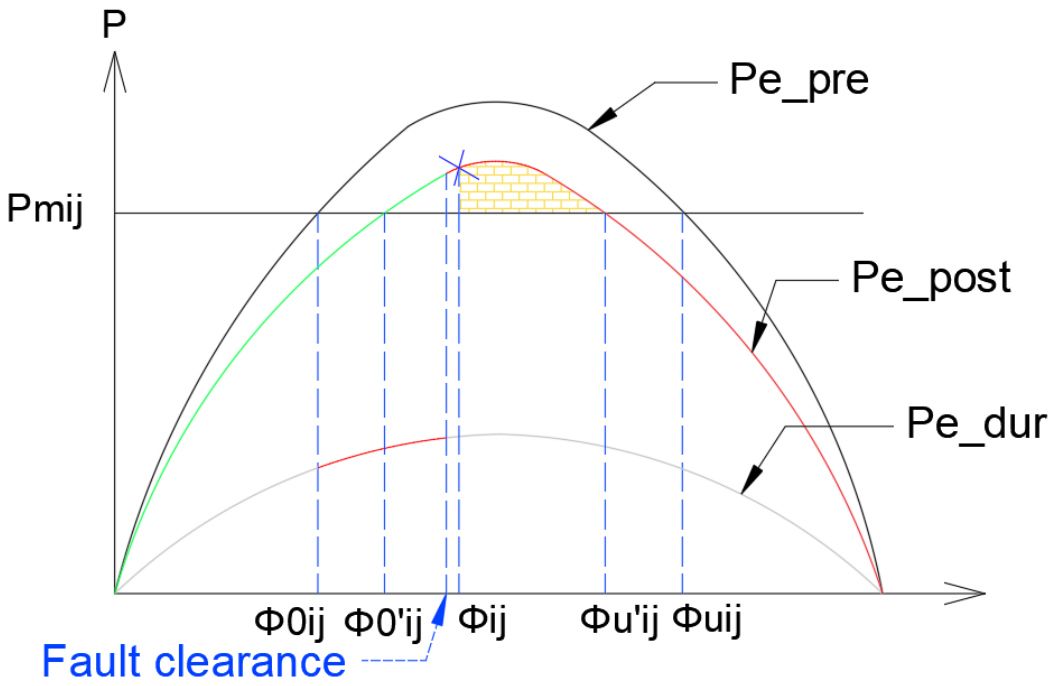


Figure 2.7: Schematic diagram for the couple-strength in post-fault for a OME

Then the first problem is regarding to the active power expression in the post-fault. Since the reduced admittance matrix will be updated and the load flow will be significantly different in the post-fault, therefore, all the components in Equation [2.19] will need to be re-calculated from the black curve to the green curve at first. Then the critical unstable angle $\phi_{u,ij}$ shall be updated to $\phi'_{u,ij}$ by updating the $\phi_{0,ij}$ to $\phi'_{0,ij}$, as they are the new equilibrium points between Pe_post and $P_{m,ij}$.

The second thing need to be reconsidered is about the available dissipated energy. As in the post-fault the PSS/E will only simulate 3 extra time cycles after the fault clearance, which ends up in the blue cross. So the available dissipated energy will no longer be the entire green area showing in Figure.2.6, but part of the area that surrounding by the new OME active power curve and its mechanical power, which is represented by the yellow block area in Figure.2.7. In mathematical form this can also be described as the update of new down limit of Equation [2.29] and [2.30]. The ϕ_{ij} is corresponding to where the simulation stops, which shall be expressed as:

$$\begin{cases} E_{dis,ij} = \int_{\phi_{ij}}^{\phi'_{u,ij}} [P_{c,ij} + P_{max,ij} \cdot sin\,(\phi_{ij} - \nu_{ij}) - P_{m,ij}]\, d\phi_{ij} \\ E_{dis,ij} = (P_{c,ij} - P_{m,ij})\phi_{ij}|_{\phi_{ij}}^{\phi'_{u,ij}} - P_{max,ij} cos(\phi_{ij} - \nu_{ij})|_{\phi_{ij}}^{\phi'_{u,ij}} \end{cases} \tag{2.37}$$

, where each component of Equation [2.37] shall be regarding to the post-fault.

And the last thing that need to be concerned is the 'extra kinetic energy' got from the during fault. Unlike the pre-fault, in the post-fault the machines' rotor speed has been influenced by the disturbance in the during-fault so that their values shall be deviating from each other, which means, Equation [2.26] shall no longer be zero in the post-fault. Therefore the coupling-strength coefficient in the post-fault shall be expressed as:

$$H_{post,ij} = E_{dis,ij} - E_{kin,ij} \tag{2.38}$$

, where the $E_{dis,ij}$ is regarding to the post-fault and calculated by Equation [2.37], and the $E_{kin,ij}$ is the extra kinetic energy calculated by Equation [2.26] through the measurement of the term $d\phi_{ij}$ in PSS/E directly.

Meanwhile, since this potential relation is between each pair of the machines, so the scale of the coupling-strength coefficient matrix shall be the square of the number of generators in the system, which in our case, shall be a 20×20 matrix. In final, after having both the coupling-strength coefficient for the pre-fault and post-fault, then a coupling-strength coefficient difference matrix, the ΔH, can be achieved as the their subtraction, which can be expressed as Equation [2.39]. The coupling-strength coefficient difference ΔH can tell the story about how this potential relations changes after the fault clearance, and it will be utilized into the next Depth-First-Search algorithm for the machine clustering.

$$\Delta H_{ij} = H_{post,ij} - H_{pre,ij} \tag{2.39}$$

Besides, the equations from the Equation [2.16] to Equation [2.22], and the equations from the Equation [2.26] to Equation [2.30], are both referred from [8].

2.5.2 Depth-First Search (DFS) Algorithm for Clustering

Initialization to the DFS algorithm

The base idea of the Depth-First Search (DFS) algorithm, is to simulate the eye recognition of humanity by analyzing the graph theory. To execute this algorithm,

the ΔH matrix needs to be decomposed at first, and then be recombined into a new vector Δh by ranking the elements of ΔH into a ascending list. And during the transformation process, the elements in the Δh vector shall also record their previous indices that belong to the ΔH matrix. After that, initializing the adjacency matrix by a One matrix with a scale of the square of generator numbers, which in our case, shall be 20×20.

Applied DFS Strategies

Then in the next step, basing on the several tests and experience of the DFS implementation in this project, this paper developed three alternative strategies to help implement the DFS algorithm, which is called 'Numerical method', 'Extended Numerical Method' and the 'Clock method', respectively. The application of these three methods along with the DFS initialization can be described by the flow chart in Figure.2.8 below, where the key is equal to 1 in the beginning:

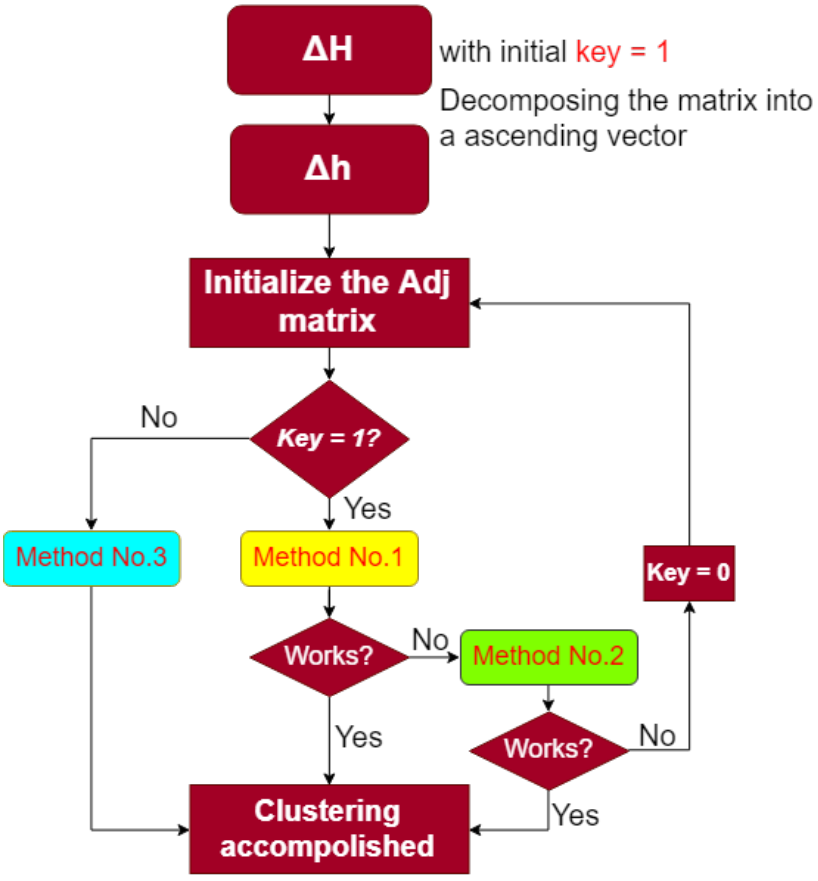


Figure 2.8: Flow chart of the DFS strategies

In the first 'Numerical method', all the elements in the Δh vector that smaller than -5 will be removed, along with turning the adjacency matrix elements that correspond to the Δh elements' previous indices in ΔH into zero. And when the sum of one column of the adjacency matrix is not larger than 4, which means this machine has lost relations with at least 15 other machines, then the machine that corresponds to this column in the adjacency matrix shall be detected as the a Critical Machine.

However, in some special cases, the majority of coupling-strength coefficient will not decrease less than 5, which means the 'Numerical Method' might cannot find a critical machine as expected. In the situation like this, the second method 'Extended Numerical Method' will be necessary as the supplement of the first method. As the role this method plays, the trigger of activating it is when the first method fail to have a detection of the critical machine. Then this method will count that how many elements does a column have that less than -5, and if the number of these elements in one column are not less than 8, then the machine corresponding to this column will be detected as a critical machine. The idea behind this method is that though there might be not so much '-5' in the pattern of one machine versus the others, however, when the '-5' issue occur larger than 8 times in this column, then this column should also be paid attention to, which in our case, means being clustered as a Critical Machine.

Besides, if the first two methods are both invalid, then the third one the 'Clock Method' will be activated. When run the DFS in the third method, the adjacency matrix will be initialized again at first, then the elements in the Δh vector will be removed one by one from the smallest as well as the turning in the corresponding place in the adjacency matrix until the first critical machine occurs. The criterion of the Critical Machine is the same with the first method, and after the first Critical Machine occurs, the removing in Δh and turning in adjacency matrix will continue 20 extra loops since in case of that other Critical Machines occur just after the first one. Compared with the first two methods, the third one is a bit rougher, which is the reason why the 'Clock Method' is the last priority among these three DFS strategies as a backup.

Afterwards, when the clustering is accomplished, machines in the system will be clustered as Critical group and Non-Critical group as well as their parameters like inertia coefficient, rotor angle, rotor speed, mechanical power and active power. The subscript 'k' is for the Critical and 'j' is for the Non-Critical, respectively.

2.6 Introduction to Multi-Machine System Transient Stability Analysis

Due to the complexity of the multi-machine system, a equivalent machine that can represent all the machines in the system will be necessary for their transient stability analysis. The technique of this is called 'One Machine Infinite Bus', who combines the CMs and NMs that come from the Clustering algorithm, into one equivalent machine but at the same time keeps the accuracy of stability assessment prediction. This paper will take the OMIB as an available, valid technique instead of discussing its feasibility. However, the OMIB method will be explained in the normal mathematical expression, significant angles calculation and stability margin calculation in the following three sub-paragraphs, respectively.

2.6.1 One Machine Infinite Bus (OMIB)

According to the related literature, before applying the OMIB, the generators inside the system need to be separated into Critical Group and Non-Critical Group, which is already achieved by going through the Clustering algorithm. Then the steps of the OMIB is as followed:

(i) Firstly, calculate the Critical- and Non-Critical Machine inertia:

$$M_C = \sum_{k \in C} M_k, \ M_N = \sum_{j \in N} M_j \tag{2.40}$$

(ii) Then the aggregation of corresponding Center of Angle (COA) of Critical Group and Non-Critical Group, respectively:

$$\begin{cases} \delta_C(t) = M_C^{-1} \sum_{k \in C} M_k \delta_k(t) \\ \delta_N(t) = M_N^{-1} \sum_{j \in N} M_j \delta_j(t) \end{cases} \tag{2.41}$$

and similarly, for the rotor speed of the Critical Group and Non-Critical Group:

$$\begin{cases} \omega_C(t) = M_C^{-1} \sum_{k \in C} M_k \omega_k(t) \\ \omega_N(t) = M_N^{-1} \sum_{j \in N} M_j \omega_j(t) \end{cases} \tag{2.42}$$

With the implementation of these equations above, the generators in the system have been divided into a two machine equivalent, where the subscript C stands for Critical Group and N for Non-Critical Group, respectively.

(iii) Afterwards, as the definition of OMIB method, a one machine equivalent will need to be achieved basing on the two machine equivalent obtained as CMs and Nms above. Basing on the COA and rotor speed calculated by Equation [2.41] and Equation [2.42], the rotor angle and rotor speed for this one machine equivalent can be expressed as:

$$\begin{cases} \delta_{OMIB}(t) = \delta_C(t) - \delta_N(t) \\ \omega_{OMIB}(t) = \omega_C(t) - \omega_N(t) \end{cases} \tag{2.43}$$

Besides, this one machine equivalent inertia is defined as:

$$M_{OMIB} = \frac{M_C M_N}{M_C + M_N} \tag{2.44}$$

(iv) Finally, after the transformation of generators' fundamental parameters, the active power, reactive power and the acceleration power of this one machine equivalent will also need to be achieved for the further Equal Area Criterion (EAC) calculation. The mathematical expressions of its power flow can be described as followed in Equation [2.45], where the P_m and P_e is actually obtained by PSS/E short-time simulation or snapshots from PMUs. Meanwhile they have already been divided into Critical Group and Non-Critical Group after going through the Clustering method and the subscripts have the same meaning with the Equation [2.40], [2.41] and [2.42].

$$\begin{cases} P_{m\,OMIB}(t) = M_{OMIB}(M_C^{-1} \sum_{k \in C} P_{m_k}(t) - M_N^{-1} \sum_{j \in N} P_{m_j}(t)) \\ P_{e\,OMIB}(t) = M_{OMIB}(M_C^{-1} \sum_{k \in C} P_{e_k}(t) - M_N^{-1} \sum_{j \in N} P_{e_j}(t)) \\ P_{a\,OMIB}(t) = P_{m\,OMIB}(t) - P_{e\,OMIB}(t) \end{cases} \tag{2.45}$$

So far all the basic parameters of the generators in system have been transformed into the type of one machine equivalent (OME). In the following chapters, the way how to apply all these parameters will be elaborated and discussed. Besides, the equations from the Equation [2.40] to Equation [2.45] are referred from [7], [8] and

[10].

2.6.2 Margin Calculation

The transient stability margin is the only criteria for the stability prediction. Since it highly depends on the application of the Equal Area Criterion (EAC), so before the mathematical description of margin calculation, a recap of the EAC combing with the time-series structure of this project will be elaborated along with the simplification assumption.

Assumption and Mathematical Simplification

Firstly, basing on the tolerable assumption, the active power in the post fault of this one machine equivalent can still be simplified as Equation [2.46]:

$$P_{e\,OMIB}(t > t_{clear}) = \frac{E'\,E_B}{X_T^P} \cdot sin(\delta_{OMIB}(t)) \tag{2.46}$$

While after the fault has been cleared, the configuration of the whole diagram will be settled then the term $\frac{E'\,E_B}{X_T^P}$ can be a constant value. So that the Equation [2.46] can be represented by Equation [2.47] below:

$$P_{e\,OMIB}(t > t_{clear}) = K_{post} \cdot sin(\delta_{OMIB}(t)) \tag{2.47}$$

As what has been stated in Chapter 3.3, the short-time PSS/E simulation will end up at three time cycles just after the fault clearance. So this constant K_{post} can be calculated basing on the data from these three extra time cycles who stands for the post fault. Then K_{post} can be calculated as followed in Equation [2.48]:

$$K_{post} = mean(\sum \frac{P_{e\,OMIB}(t > t_{clear})}{sin(\delta_{OMIB}(t > t_{clear}))}) \tag{2.48}$$

The purpose of this simplification is also the purpose of the OMIB method. Since basing on the reasonable simplification the active power output of this one

machine equivalent can be predicted as a sinusoidal function that only associated with $\delta_{OMIB}(t)$. Instead of using real time domain simulation who is both time- and computation-consuming, the new approach that described in Equation [2.47] and [2.48] can not only help saving time, but also be beneficial for the calculation of the deceleration area (A_{dec}), which will be discussed in the following paragraph.

EAC recap & Project time-series structure

According to the knowledge of EAC, when in the during-fault period the active power of generator will decrease dramatically, while at same time the mechanical power $P_{m\,OMIB}(t)$ absorbing from the prime mover will not change that much. Then the $P_{m\,OMIB}(t)$ will be significantly larger than the $P_{e\,OMIB}(t)$, which means the acceleration power $P_{a\,OMIB}(t)$ will be positive and the machine rotor will be kept accelerating from its rated speed until the fault is cleared. Basing on its mathematical expression this acceleration power can be represented by the area of A_{acc} in Figure.2.9 and it is opposite with the machine's synchronism. Fortunately, when the fault has been cleared, the machine active power curve will enhance dramatically and be apparently larger than its mechanical power, which makes the $P_{a\,OMIB}(t)$ back to negative and decelerates the rotor. At this time the $P_{a\,OMIB}(t)$ can be described as the opposite number of the area of A_{dec}. The schematic diagram describing the entire statement shows in Figure.2.9:

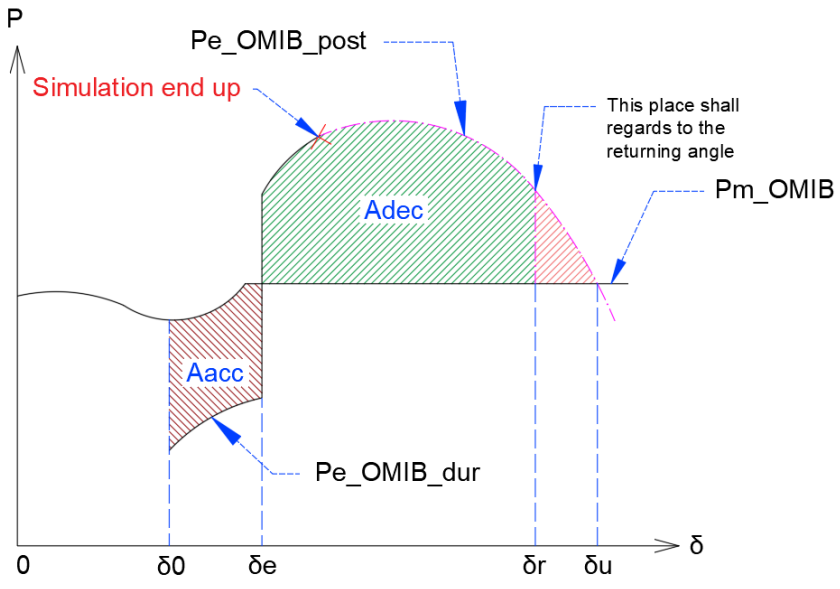


Figure 2.9: One Machine Infinite Bus schematic diagram

As what can be observed from the Figure.2.9 above, the area of the A_{acc} is the acceleration power that given to rotor and the green area is the available dissipated energy that help consume the detrimental A_{acc}. Since the PSS/E simulation will be executed from steady state until 3 time cycles after the fault clearance (where the red arrow points at), then it means all the machine data in the during-fault period will be available and the area of A_{acc} can be directly calculated by using the finite element integral of the known P_a to δ. This can be achieved by using the sum of the finite areas division of the A_{acc}.

On the other hand, for the area of A_{dec}, since the analytic expression of $P_{e\,OMIB}(t > t_{clear})$ is already available after knowing the coefficient 'K_{post}', and the mechanical power is assumed constant in this project, then the equation for describing the A_{dec} can be expressed as followed:

$$A_{dec} = \int_{\delta_{cross}}^{\delta_r} (K_{post} \cdot sin(\delta_{OMIB}) - P_{m\,OMIB})\, d\delta_{OMIB} \tag{2.49}$$

, where the δ_{cross} is the angle regarding to when the $P_{e\,OMIB}$ exceeds the $P_{m\,OMIB}$, and δ_r is the returning angle when the machine's rotor speed starts getting negative from zero. Then by simplifying the Equation [2.49], the A_{dec} can be reformulated as Equation [2.50] below:

$$A_{dec} = (-K_{post} \cdot cos\delta_{OMIB} - P_{m\,OMIB} \cdot \delta_{OMIB})|_{\delta_{cross}}^{\delta_r} \tag{2.50}$$

2.6.3 OMIB Angle Calculation

Returning Angle Calculation

As the definition of the machine returning angle, it occurs only if this one machine equivalent has already dissipated its extra kinetic energy before running out the entire available dissipated energy (green area in Figure.2.9). This means at that point the acceleration power A_{acc} shall have an equilibrium with part of the entire dissipated energy. Basing on this idea, the equal between A_{acc} and A_{dec} can be constructed as followed:

$$A_{dec} = A_{acc} \tag{2.51}$$

and by taking the Equation [2.50] of A_{dec} and the value of A_{acc} calculated from the integral, the Equation [2.51] can be reformulated as:

$$(-K_{post} \cdot cos\delta_{OMIB} - P_{m_{OMIB}} \cdot \delta_{OMIB})|_{\delta_{cross}}^{\delta_r} = A_{acc} \tag{2.52}$$

then by simplifying it, the equation that holds the δ_r as a variable can be described as:

$$P_{m_{OMIB}} \cdot \delta_r + K_{post} \cdot cos(\delta_r) = K_{post} \cdot cos(\delta_{cross}) + P_{m_{OMIB}} \cdot \delta_{cross} - A_{acc} \tag{2.53}$$

It is obvious that the Equation [2.53] is a transcendental function, who both holds the variable and its cosine value. Since the δ_{cross} can be directly detected or calculated when the fault is cleared, then the entire right side of Equation [2.53] can be a constant value, which means the Equation [2.53] can also be treated as a form like this:

$$A \cdot \delta_r + B \cdot cos(\delta_r) = C \tag{2.54}$$

, where:

$$\begin{cases} A = P_{m_{OMIB}} \\ B = K_{post} \\ C = K_{post} \cdot cos(\delta_{cross}) + P_{m_{OMIB}} \cdot \delta_{cross} - A_{acc} \end{cases} \tag{2.55}$$

Though a transcendental function cannot be solved directly, it can still get a arithmetic solution through the iteration in Python. Basing on the Repeated Root Iterative Method expressed in Equation [2.56] below, and by setting the 'maxstep' and tolerance for the solution error, the solution that meets the tolerance can be achieved. Otherwise, if the 'maxstep' is already achieved but still there is no solution, which means, the equation above does not have a root to balance its left and right side at the same time. When this happens, it can be said that there is no returning angle for this one machine equivalent, and the rotor angle will just increase and cross the critical unstable angle boundary and then lose synchronism forever.

$$x_{k+1} = x_k - \frac{f(x_k)\ f'(x_k)}{[f'(x_k)]^2 - f(x_k)f''(x_k)} \tag{2.56}$$

Unstable Critical Angle Calculation

The unstable critical angle is a boundary that better not cross. Since after crossing the boundary the mechanical power of the OME will be always larger than its active power, so that the rotor of the OME will be accelerated forever as long as the $P_{m_{OMIB}}$ does not change. So this is also why that the δ_u normally used as the upper limit of the integral of the entire available dissipated energy. To calculate δ_u, the initial equilibrium, the δ_0 needs to be achieved at first, since they have a relation as:

$$\delta_u = \pi - \delta_0 \tag{2.57}$$

and the δ_0 of the OME can be calculated by using the equilibrium between the OME mechanical power and OME post-fault active power as Equation [2.58]:

$$P_{m_{OMIB}} = K_{post} \cdot sin(\delta_0) \tag{2.58}$$

then the critical unstable angle of the OME can be calculated as:

$$\delta_u = \pi - arcsin(\frac{P_{m_{OMIB}}}{K_{post}}) \tag{2.59}$$

2.6.4 Margin Value Calculation & Stability Judgement

After all the preparation calculation of the OMIB method, the final Margin calculation who can directly judge the system as stable or unstable is going to be implemented as followed. Since the stability situation can be distinguished by the existence of the returning angle, and basing on the physical meaning of the stability margin, it can be divided into three conditions, the unstable case, the stable case and the marginal stable case.

First of all, for the unstable case, the machine rotor will keep accelerating till crossing the boundary, which means even the entire available dissipated energy has been consumed there is still some kinetic energy remaining. Therefore, the definition of Margin value of a unstable condition is the remaining kinetic energy after rotor angle crossing the unstable critical boundary. Thus, the Margin value

of the unstable case can be determined as:

$$\eta_{unstable} = \int_{\delta_{cross}}^{\delta_u} (K_{post} \cdot sin(\delta_{OMIB}) - P_{m\,OMIB})\, d\delta_{OMIB} - A_{acc} \tag{2.60}$$

since the entire available dissipated energy is smaller than the acceleration energy getting from the during-fault (the A_{acc}), the calculated result of this unstable case shall be a negative value.

On the other hand, when the rotor can actually have a returning angle in the end, then the case can be treated as stable. In the stable case situation, before running out the entire available dissipated energy, the kinetic energy getting from the during-fault has already been consumed completely. In a stable case, the Margin of the system is defined as the remaining available dissipated energy when the rotor starts to roll back. Basing on this physical meaning the mathematical expression of stable margin is as followed:

$$\eta_{stable} = \int_{\delta_r}^{\delta_u} (K_{post} \cdot sin(\delta_{OMIB}) - P_{m\,OMIB})\, d\delta_{OMIB} \tag{2.61}$$

, which is the integral of a minus acceleration power ($-P_a$) from returning angle to the critical unstable angle. The value of this term shall also equal to the magenta area in Figure.2.9, which is the area regarding to 'the entire available dissipated minus the A_{dec}'.

Furthermore, when the calculated margin value for the stable case is quite low, which in our case, lower than 2, then this stable case need to be treated as the third option, the marginal stable case. A margin stable case means that when the stability margin for the entire system is very closed to the boundary 'zero', then the condition of the system can be concerned as very uncertain since a very tiny disturbance can influence the sign of the margin value and make it unstable. Therefore, in this project, those who has a calculated margin value that lower than 10 will be marked as 'Marginal Stability'. And to truly figure out the stability situation of this system, a full-time domain simulation in PSS/E will be necessary.

Besides, the scope of the application of 'OMIB' method also needs to be discussed here. Since this project mainly focuses on the first-swing transient stability, hence the 'OMIB' can be the downstream algorithm that takes the clustering result from the 'Coupling-Strength Algorithm'. However, in some cases of the N-1,

the system will have a multi-swing problem that the system can survive in the first-swing but fail in afterwards, which cannot be detected by the 'OMIB' method even when the clustering result is correct. For these cases a evolved 'OMIB' will need to be applied but since this project only focuses on the first-swing, therefore the evolved 'OMIB' will not be implemented here this time.

2.7 Methodology Summary

This chapter mainly focuses on the using methodology interpretation and derivation. Among everything inside, highlighting the admittance automatic generation, the Coupling-Strength Clustering method under pre-fault and post-fault and the One Machine Infinite Bus (OMIB) method for the stability analysis. Also there is a discussion about the applying scope of the OMIB method. All statements of this Chapter 2 will be highly coupled with the further chapters, which are the implementations of this project in both PSS/E and Python scripts.

CHAPTER 3
Short-Time PSS/E Simulation

The functionality of PSS/E in this project works as a simulation of the PMUs' snapshot. By taking the .raw file and .dyr file, the PSS/E can provide a database including generator associated parameters and the loadflow results. The implementation of the PSS/E will be described in the perspective of its input/output, necessary running time and general implementation, respectively.

3.1 PSS/E I/O

As the simulation of the PMUs' snapshot, there must be a corresponding data as the input for this simulation, and a output that can work as a snapshot. The first input of the PSS/E is a .raw file, who is the most common file format for uncompressed data, which can be directly opened by the software $'Notepad++'$ and checked. The second input of the PSS/E is a .dyr file, which holds the information of the dynamic models of the selected system and need to be applied along with the .raw file to create the steady-state environment for the system.

On the other hand, the output of the PSS/E simulation is a .out file. The .out file is a special format that can only be read by a specific function named '***read_psse_outfile***'. Since this function is available at the beginning of this project so it can be treated as a starting condition and can be directly used.

3.2 PSS/E Running Time Setting

Then before the implementation of PSS/E by Python script, the only thing needs to be discussed is about the running time setting in the system. First of all, the system will accomplish all the settings and running from 0 [sec] to 1 [sec] for the simulation of the steady-state. Afterwards, the branch fault shall happens and

system goes into the during-fault period. Since the reaction time of the relay is around 10ms to 500 ms, so in this project 200ms is taken as the reaction time of the relay. So the simulation time for the during-fault shall be from 1 [sec] to 1.2 [sec].

After the during-fault, the system goes into the post-fault condition and reacts differently. However, according to the methodology statement in Chapter 2, there is no need to simulate the entire post-fault period since our clustering method and stability assessment shall works well without a real time domain simulation. Therefore, the simulation time will end up at three time cycles just after the fault clearance. As the simulation time interval is around 0.01 [sec], so the final simulating time is set as:

$$t_{final} = t_{clear} + 3 \times 0.01 \tag{3.1}$$

3.3 PSS/E Implementation

The entire PSS/E implementation can be achieved by the Python commands because of their seamless interconnection. And for the convenience of the reader, the implementation process is expressed by a flow chart, which can be observed in Figure.3.1. The blue rounded blocks stand for the I/O part of PSS/E, while the red parts are for the normal operating schemes. The yellow highlighted blocks are the ones who need to be interpreted detailedly and those green blocks are regarding to the 'running' issues.

The implementation starts from taking the initial data from the .raw file and .dyr file, who represent the loadflow data and dynamic model, respectively. After that, to set solution parameters for them, the **'solution_parameters_4'** and **'dynamics_solution_param_2'**, who are the built-in function of package **'psspy'**, will be implemented basing on the orientation of Book 'API'. Then after creating the output file by function **'change_channel_out_file'**, the necessary variables that need to be measured will be picked up among all 37 availably monitoring variables in function **chsb**. Since transient stability is the analysis against generators, so the machine rotor angle, rotor speed, active, reactive and mechanical power will be necessary for the further calculation before the Clustering, as well as the complex machine terminal voltage for calculating the elements of the admittance matrix.

Afterwards, by monitoring the rotor angle, a angle reference will need to be chosen for the entire system. The reference base chosen here is the system weighted average angle. Then on the other hand, since the loads are represented by their corresponding apparent power, while when using the constant power load for the dynamic simulation, the numeric problems probably will happen. Therefore, a transformation from the constant MVA load to a mixture of constant MVA, constant current and constant admittance load characteristics will be necessary to make the calculation executable. Then by adjusting the system components for the switching studies, and initializing a PSS/E dynamic simulation for the state-space simulations, the system can finally be capable of running the simulation. As what has been discussed in section 3.2, the entire system will simulate the steady-state for 1 [sec], during-fault period for another 0.2 [sec], and three extra time cycles for the post-fault period. In final, the output data will be recorded into a .out file, which will be the input data of the Clustering Algorithm and can only be read by the built-in function '**read_psse_outfile**.'

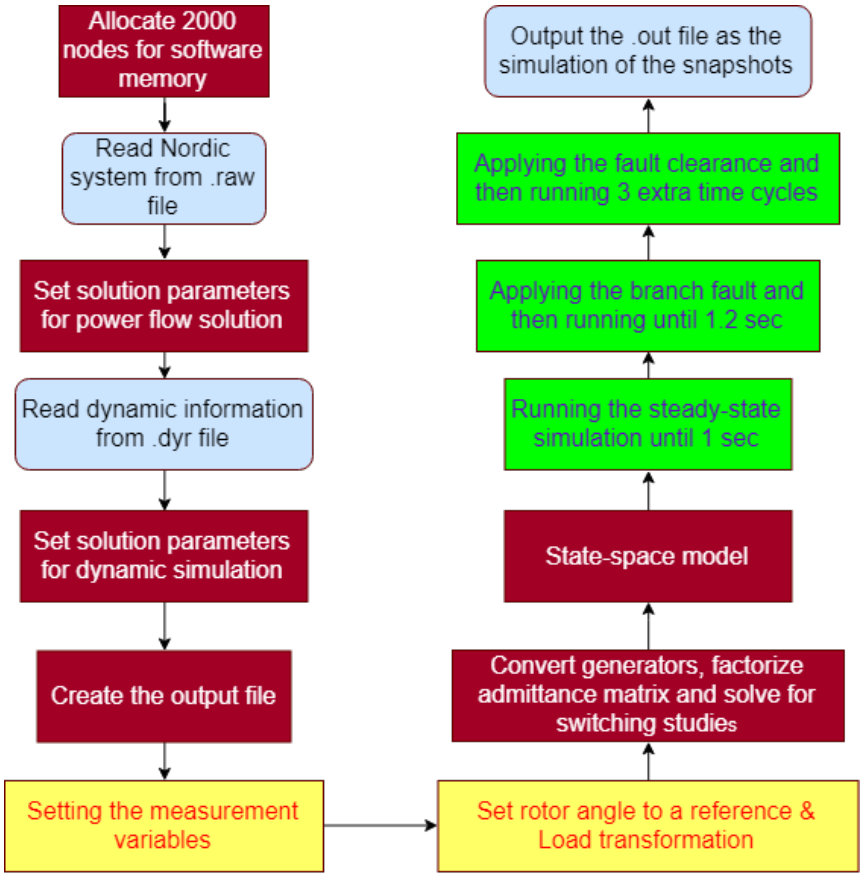


Figure 3.1: PSS/E implementation diagram

CHAPTER 4

Clustering Method Implementation

From this chapter, the specific implementation process will be introduced and discussed. Since the theories regarding to the whole process have already been elaborated in Chapter 2, so the statements here will mainly focus on the associated programming level where can make the code executed as expected. The orientation will start from the beginning when the system takes the branch information, then go through the short-time simulation in PSS/E and the implementation of the Clustering method. In final, three different cases will be taken as the examples to go through the Clustering algorithm and provide visualized results. However, for the convenience of readers, the specific implementation process of OMIB will be elaborated in the next chapter due to its complexity.

4.1 Branch Information into Python

First of all, the information of the entire buses needs to be input into the Python working domain. As what has been discussed in the Chapter 2, since the index tracking of the corresponding buses is necessary for the whole project, so for the convenience of recalling anytime, the buses' names information are recorded into a dictionary, whose keys are all the buses' names, respectively. The dictionary is called **'B'**, and what regarding to its keys, are the indices from 0 to 73 as there are 74 buses in the system in total. The benefit for this application is that when there is a need of tracking the bus corresponding index, for instance, Bus 4011, then the command:

B[4011]

can directly give back the corresponding index Bus 4011 is regarding to. Except

the simplicity, this method also makes the code readable and can be easily applied for the further implementation.

Afterwards, it is the branches information that need to be put into the system. Since there are 102 branches (52 strippable branches and 50 transformer branches) in total in the entire system, so an array called **'direction'** is initialized by a 102×2 int zero array. And this initial one will be updated by taking the 'from_bus' corresponding index in the first column and the 'to_bus' corresponding index in the second one, which can be directly achieved by using the dictionary created in the beginning who holds the buses name and their indices.

Besides, the function made for this part is called **'Branch_information.py'**, which shall be called in the beginning of the entire program and its variables should also be set as the global variables. The code for this part can be found in the appendix.{Branch Information}.

4.2 PSS/E Short-time Simulation

The majority of the PSS/E implementation has been elaborated in the Chapter 3, however, this section will mainly discuss one of its programming details. Since the PSS/E simulation has been packaged up to a single function, and the code is expected to have an automation for all the possible branch faults. So the interconnection between this function and its strippable branch term 'trip_index' will be necessary. This is actually achieved by setting 'trip_index' from 0 to 51 that regarding to all the strippable branches at first, and then use the commands below to recall the corresponding names of the tripped 'from_bus' and 'to_bus' by taking the elements from the array **'direction'**. The function in this part is called **PSSE_simulation.py** and can be found in the appendix.{PSS/E Simulation}.

```
trip_branch_from = list(B.keys())[list(B.values()).index(direction[trip_index,0])]
trip_branch_to   = list(B.keys())[list(B.values()).index(direction[trip_index,1])]
```

4.3 Short-time Simulation Results Reading from PSS/E

After the simulation in PSS/E, its result data is stored into a .out file, however, to read this format, a specific function called **read_psse_outfile.py** will be necessary to decode it. Since this function is pre-completed before this project, so in this paper there will be no discussion about how it works but just take it as an available tool.

But on the other hand, though the decoding the .out file can be achieved by an available tool, the output of this function is only a complicated dictionary variable called **'data'**, which means a further and efficient 'decomposion' of this variable shall be necessary. This can be achieved by creating the corresponding list that holds the keys to the dictionary and using the array traversal to separate the big messy dictionary variable **'data'** and allocate the elements to the corresponding terms.

Besides, two problems need to be mentioned here. The first one is that when taking the mechanical power results, the P_m from PSS/E, it actually gives them under the machine bases rather than the system base. So before output this term, an unity of the base from the machine to the system shall be implemented as followed in Equation [4.1].

$$P_{m,sys} = P_{m,mach} \times \frac{M_{base}}{S_{base}} \tag{4.1}$$

Then the second problem is about the moment of inertia, who is achieved by the relation with inertia constant rather than directly measured. In the p.u.system their relation can be expressed as:

$$M_{pu} = 2 \times H_{pu} \tag{4.2}$$

, where the H_{pu} can be directly achieved by checking the .dyr file in PSS/E. The function of this part is called **results_Reading.py** and can be found in the appendix{Results Reading}.

4.4 Admittance Matrix Generation

The theories behind the generation of the admittance matrix in the pre-fault and post-fault have been elaborated already in the Chapter 2.4, and the implementation in the programming level shall just follow the equations and the rules described over there. However, there are only three problems need to be stated here. The first one is about the unit of the fixed shunt admittance. Since the data given for the fixed shunts is by using the unit [Mvar], then a coordination between its original unit and the p.u. system shall be achieved. Basing on the knowledge of the p.u. system, the unity of the units can be expressed by the Equation [4.3] as below:

$$B_{f,sys} = 1j * \frac{B_{f,norm}}{Sbase \cdot (V_{bus} \cdot \widetilde{V_{bus}})} \tag{4.3}$$

,where the $B_{f,norm}$ is the value with [Mvar] unit. The V_{bus} and the $\widetilde{V_{bus}}$ is the corresponding terminal voltage vector and its conjugate version.

Then the second problem is about the generator reactance using for calculating the augmented admittance matrix Y_{aug}, connection condition matrix Y_{bg} and the generator transient admittance matrix Y_{gg}. Since the whole project is coping with the transient stability, so the reactance using here shall be the transient reactance, the X'_d, which can be taken from the .dyr file directly. However, since the value of this data is also given by the machine base, so a translation to system base shall be necessary, which can de described as Equation [4.4]:

$$X'_{d,sys} = 1j * X'_{d,mach} \times \frac{S_{base}}{M_{base}} \tag{4.4}$$

Then the last one is regarding to the structure of matrix Y_{bg} in Figure 2.4 and the Equation [2.11]. When decomposing the first row of the matrix calculation in Equation [2.11], an equation can be derived as followed:

$$Y_{bg} \times E' = -Y_{aug} \times V_{bus} \tag{4.5}$$

While, the problem of this is, since the structure of Y_{bg} is sort of 'unique' that only holds non-zero elements in its area_2, so its multiple with the machine internal

voltage vector E' shall also give back a similar vector that only holds non-zero elements in its 'area_2'. On the other hand, duo to the equilibrium between the two sides in Equation [4.5], the right side of Equation [4.5], shall also have a very similar result with the left side. This relation can works as the fundamental validation of the admittance matrix calculation, since this is the mathematical relations behind it and should be strictly met.

The functions of this part are called **red_Ybus_pre.py** and **red_Ybus_post.py**, which is for pre-fault and post-fault, respectively. And they can be found in the appendix{Admittance Matrix in Pre- & Post-fault}.

4.5 Coupling-Strength Algorithm Calculation

Since the theories behind the coupling-strength algorithm have already been elaborated in Chapter 2.5.1, and the implementation process of it shall just be the linear order by following from the Equation [2.16] to Equation [2.39]. However, in the programming level, there are still two problems that need to be mentioned here since they might generate errors in the programming process. The first one is regarding to Equation [2.22], when calculating the partial OME active power C_{ij} and reactive power D_{ij}. As what can be observed from these two equations, when calculating the active power or reactive power for each machine, that the term under the symbol $\sum$ should not equal to itself. This can be explained as in Equation [2.20], that the calculation of the term $P_{c,ij}$ has already taken the elements regarding to themselves so there is no need for the repeating.

And the second problem is about relation between the Equation [2.18] and [2.19]. The purpose of this reformulation has already been discussed in the paragraph **Coupling-Strength Calculation for the Pre-fault** in section 2.5.1. Since the Equation [2.19] is the reformulation of Equation [2.18], so the calculation result of Equation [2.19] shall be very closed to or even equal to Equation [2.18]. As this term '$P_{e,ij}$' is very sensitive to the input, therefore even a very tiny hand mistake can make a disequilibrium between Equation [2.18] and Equation [2.19]. Since sometimes, tiny errors can be made by hand but very hard to be detected at the same time, so this relation between Equal [2.18] and Equal [2.19] can be treated as the another fundamental validation that help find if there is an error in the programming of the Clustering method or not.

Besides, the functions of this part are also separated for the pre-fault and post-fault. There are called **coupling_strength_pre.py** and **coupling_strength_post.py** respectively and can be found in the appendix{Coupling-Strength Calculation for Pre- & Post-fault}.

4.6 DFS Grouping Algorithm Implementation

The methodology of the Depth-First-Search, the DFS algorithm has been elaborated in the Chapter 2.5.2, along with its three different running strategies in Figure.2.8 and their running priority. There is only one thing need to be mentioned here as a programming artifice to help achieve the goal of the machine grouping just after the clustering. As the clustering result is stored in the term 'critical_index', who is a 20×1 vector in our case. And its inside element '1' stands for critical machine, while 0 stands for the non-critical machine in the system. Therefore, a 'shortcut' can be achieved by using the **'filter'** function. For example, when there is a will to distinguish Critical Machine rotor angle from the Non-Critical ones, a command inside the time-loop can directly achieve the goals:

*delta_K[:,i] = np.array(filter(None, (critical_index*delta[:,i])))*

, where the $[:, i]$ stands for 20 generators' rotor angles in different time cycle. Since the function **'filter'** can directly remove those zero elements inside the *(critical_index*delta[:,i])*, so the Non-Critical Machines have been directly removed and what left will be the Critical ones. Meanwhile, the same method can also be used to detect the Non-Critical Machines, just by calculating the $noncritical\_index$ vector through Equation [4.6] as below:

$$noncritical\_index = (critical\_index - 1) \times (-1) \tag{4.6}$$

, where its inside elements corresponding to the Critical Machines turn to 0 while those corresponding to the Non-Critical Machines become 1 this time. So the grouping for the Non-Critical Machine rotor angles can be achieved by the command:

*delta_j[:,i] = np.array(filter(None, (noncritical_index*delta[:,i])))*

The functions of this part are called **Clustering** and **Grouping**, respectively. The first one is to implement the DFS algorithm elaborated in Chapter 2.5.2 and the second one is to take the clustering results from the first one, and then accomplish the machine grouping. These two functions can be found in the appendix{Clustering & Grouping}.

4.7 Clustering Method Results' Visualization & Comparison with the PSS/E Short-time Simulation

In this section the entire clustering method will be implemented and returns several visualized results for the discussion. And the clustering results shall be compared with the PSS/E short-time simulation for the double check, since normally the Critical-Machines can be directly distinguished by the unusual growth of their trajectory. In this section three cases are picked up as the demonstration of this method, which is Case 1, Case 7 and Case 19, respectively. The test of the entire 52 strippable branches will be implemented in the Chapter 6, the **Assessment Automation with Every One Fault in Each Branch**, where more possibilities will occur and be discussed.

4.7.1 Branch Fault Case 1

Branch fault Case 1, regarding to the trip_index '0' in Python, is the branch between Bus 1011 and Bus 1013. Since this branch only holds its branch admittance and branch shunt, which is the simplest N-1 case in this project, therefore it is taken as the first one for the methodology validation.

First of all, set the trip_index to zero in Python script and then run the entire code for the Machine Clustering. After having the results from Python, above all, the first priority is to check the fundamental validations mentioned in section 4.4 and 4.5, that regarding to the admittance matrix calculation and coupling-strength calculation, respectively. The importance of these two fundamental validations has been elaborated in the corresponding paragraphs. For the first one, it can be checked by printing and comparing the calculation results, which is presented as below in Figure.4.1 and 4.2. As what can be observed from these two figures, due to the calculation accuracy in Python, though the figure on the right side has some 'noise' values in its 'area_1' and 'area_3', but they are all around ten to the power of minus 14, which are so tiny that can be ignored. And in the 'area_2' of them their calculation value are almost exactly the same. So in this case the first validation regarding to the admittacne matrix calculation is passed.

```
In [18]: Ybg.dot(E_dot)
Out[18]:
array([  0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
        -7.12596876 +34.51240975j,  -5.05992623 +25.19956003j,
       -10.38292497 +28.50052124j,  -7.27492169 +24.3764206j ,
         0.48428164 +11.09987251j,  11.26227783 +10.43952484j,
         6.48827419  +4.2137972j ,   4.43997739 +38.4201931j ,
        -5.41306907 +42.26616718j,  -6.45691609 +34.99089625j,
         4.25374115 +12.52468919j,   5.67941158 +14.65246885j,
         8.95380366  +6.24546987j,  17.69695519 +22.29406931j,
        26.7582966  +38.73943951j,  22.17531054 +16.46592042j,
        11.87322023 +17.876321j  ,  23.47787084 +38.86270178j,
        -2.91557161 +21.79216259j, -21.09655257+187.03748858j,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ])
```

Figure 4.1: $Y_{bg} \times E^{'}$

```
In [19]: -Yaug_pre.dot(Vbus[:,100])
Out[19]:
array([-0.00000000e+00-0.00000000e+00j, -0.00000000e+00-3.55271368e-15j,
       -0.00000000e+00-0.00000000e+00j, -2.84217094e-14-0.00000000e+00j,
       -0.00000000e+00-7.10542736e-15j, -8.88178420e-16-7.10542736e-15j,
       -4.44089210e-16+7.10542736e-15j, -1.11022302e-16-0.00000000e+00j,
       -0.00000000e+00-0.00000000e+00j,  1.77635684e-15-0.00000000e+00j,
       -0.00000000e+00-7.10542736e-15j, -0.00000000e+00-0.00000000e+00j,
        1.42108547e-14-0.00000000e+00j, -0.00000000e+00-0.00000000e+00j,
       -1.42108547e-14-0.00000000e+00j, -3.55271368e-15-0.00000000e+00j,
       -0.00000000e+00-0.00000000e+00j, -0.00000000e+00-7.10542736e-15j,
       -7.10542736e-15-0.00000000e+00j, -0.00000000e+00-0.00000000e+00j,
       -0.00000000e+00+7.10542736e-15j, -3.55271368e-15-0.00000000e+00j,
       -7.12052869e+00+3.45115281e+01j, -5.05590154e+00+2.51988039e+01j,
       -1.03782075e+01+2.84996790e+01j, -7.27114538e+00+2.43748942e+01j,
        4.85944698e-01+1.10997400e+01j,  1.12637102e+01+1.04382811e+01j,
        6.48889454e+00+4.21301913e+00j,  4.44540067e+00+3.84200672e+01j,
       -5.40658134e+00+4.22672066e+01j, -6.45178213e+00+3.49917877e+01j,
        4.25561775e+00+1.25242130e+01j,  5.68154462e+00+1.46517841e+01j,
        8.95541061e+00+6.24481423e+00j,  1.76999655e+01+2.22921619e+01j,
        2.67623989e+01+3.87356026e+01j,  2.21778799e+01+1.64633874e+01j,
        1.18760569e+01+1.78747832e+01j,  2.34836827e+01+3.88600324e+01j,
       -2.91188826e+00+2.17935088e+01j, -2.10608922e+01+1.87054193e+02j,
       -0.00000000e+00+2.84217094e-14j, -0.00000000e+00-2.84217094e-14j,
       -0.00000000e+00-7.10542736e-15j, -0.00000000e+00+1.42108547e-14j,
        2.22044605e-16-1.77635684e-15j,  7.10542736e-15+2.84217094e-14j,
       -2.30926389e-14-1.77635684e-15j, -4.44089210e-16-1.77635684e-15j,
       -7.10542736e-15-0.00000000e+00j,  5.68434189e-14-0.00000000e+00j,
       -0.00000000e+00-1.42108547e-14j, -0.00000000e+00-1.42108547e-14j,
       -0.00000000e+00+1.42108547e-14j,  1.77635684e-15-3.55271368e-15j,
       -4.21884749e-15-2.13162821e-14j,  8.88178420e-15-1.77635684e-15j,
       -3.55271368e-15-1.42108547e-14j, -3.55271368e-14-7.10542736e-15j,
        1.06581410e-14+1.95399252e-14j, -3.55271368e-15-5.32907052e-15j,
        4.26325641e-14+1.06581410e-14j,  2.13162821e-14-3.55271368e-15j,
       -1.42108547e-14+1.42108547e-14j,  4.97379915e-14+1.77635684e-15j,
       -3.55271368e-14-2.84217094e-14j, -0.00000000e+00+1.42108547e-14j,
        2.84217094e-14-0.00000000e+00j,  5.68434189e-14-0.00000000e+00j,
        2.84217094e-14+2.13162821e-14j,  2.84217094e-14-2.84217094e-14j,
       -2.66453526e-15-0.00000000e+00j, -7.10542736e-15-0.00000000e+00j])
```

Figure 4.2: $-Y_{aug} \times V_{bus}$

Then for the second fundamental validation regarding to the coupling-strength calculation, since both of result from Equation [2.18] and from Equation [2.19] are holding tons of elements, so the double-check can be achieved by finding the maximum among these two equations and check if this value is small enough. This can be directly achieved through the **Max** function in Python script:

```
In [5]: np.max(Pee_ij - Pe_ij)
Out[5]: 0.0005064960941614238
```

Figure 4.3: $P_{e,ij}$ comparison in Case 1

, where the $P_{ee,ij}$ is the simulating result calculating in Equation [2.18], and the $P_{e,ij}$ is the calculation result from Equation [2.19]. As what can be observed from Figure.4.3, the maximum of their difference is 0.0005064960941614238, which is small enough to be ignored. Therefore, so far the case 1 has met both the two fundamental validations and can provide a preliminarily convincing clustering result. Afterwards, plot the pre-fault, post-fault coupling-strength coefficient matrix, and its difference matrix ΔH respectively as followed:

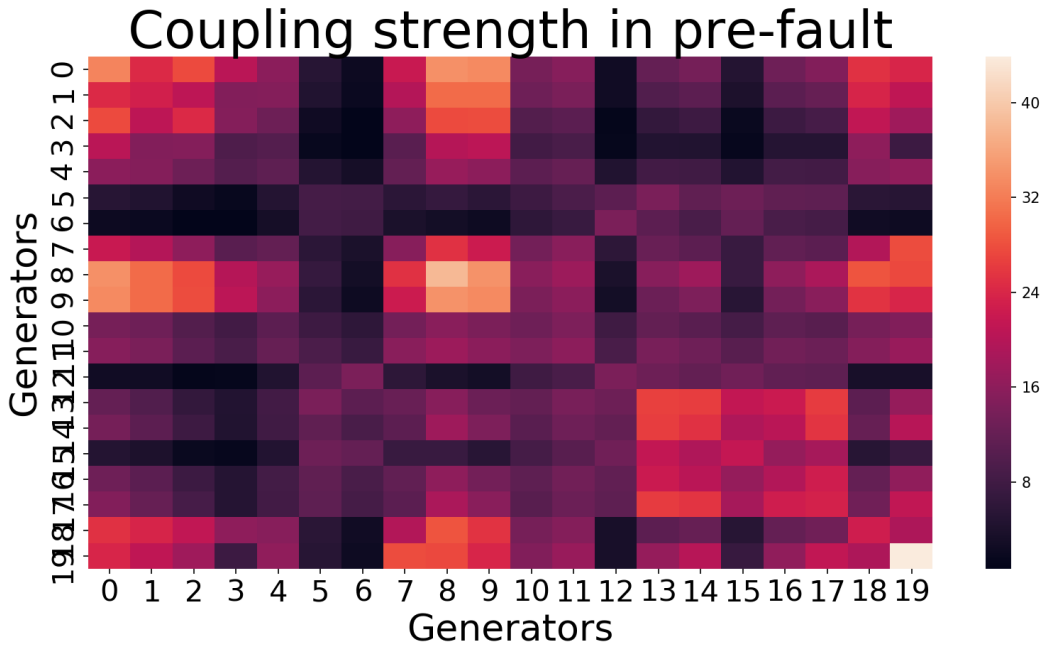


Figure 4.4: Coupling-Strength Coefficient Matrix in pre-fault for Case 1

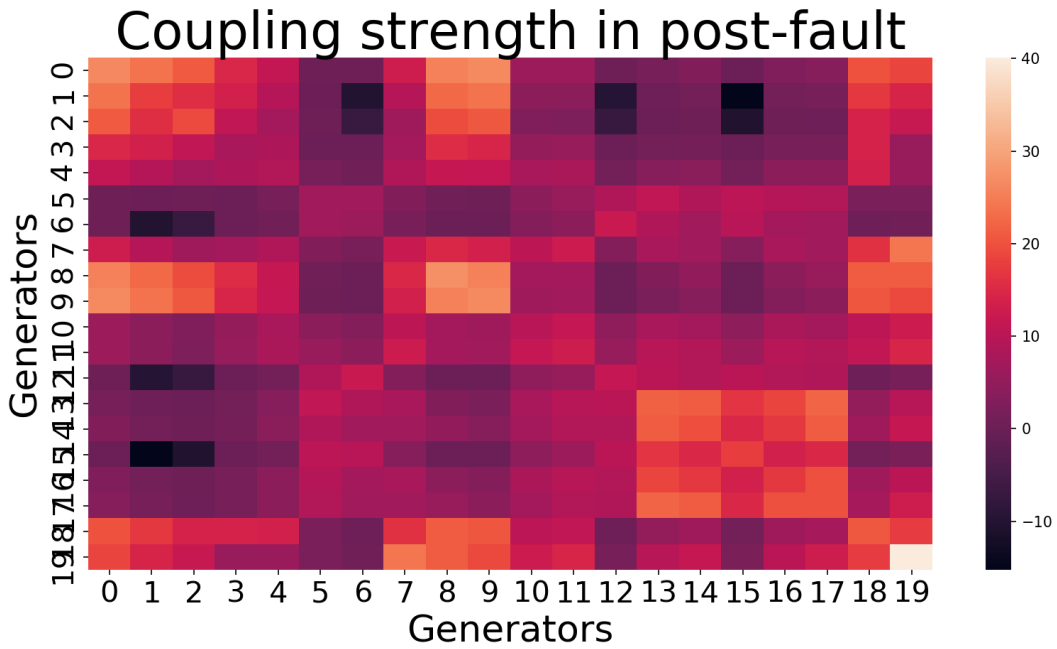


Figure 4.5: Coupling-Strength Coefficient Matrix in post-fault for Case 1

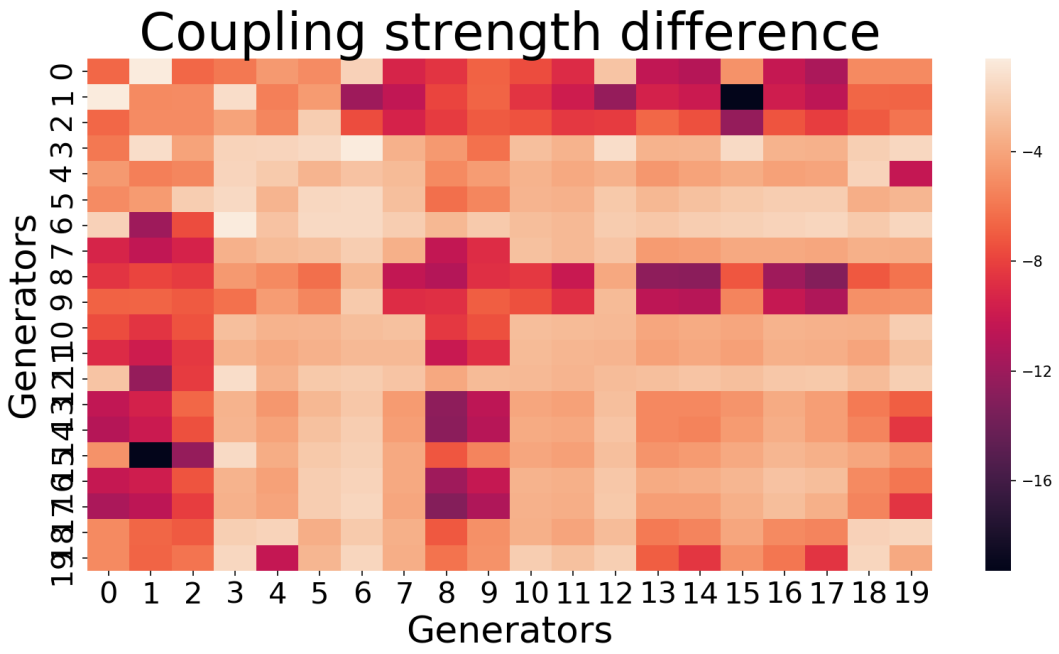


Figure 4.6: Coupling-Strength Difference Matrix for Case 1

As what shows in both the Figure.4.4, 4.5 and 4.6, their X-axis and Y-axis are all starting from Generator 1 (Python index 0) to Generator 20 (Python index 19). Furthermore, since the coupling-strength coefficient actually shows a relation between machines, so this relation shall be the same no matter how the direction of the observation is, which means, all these three matrices shall be symmetrical. This phenomenon actually can be observed from the left upper point, where can be treated as the common beginning point of both the X-axis and Y-axis, to the right down point, where can be treated as the common end point of both the X-axis and Y-axis.

And then by checking the Figure.4.6, it shows that almost all the coupling-strength coefficients regarding to all pairs of machines in the post-fault have a decrease in different degrees. This can be explained as after the disturbance in the system, the corresponding potential relations among all the machines have been weakened in different levels due to the difference of their locations, capacities and connection situation. Furthermore, in Figure.4.6, the situation becomes severer as the color darkens, which means, where the deep color shows up more frequently than the others, then the regarding place will more likely be a Critical Machine. Comparing with the other machines in Figure.4.6, the deep color can be apparently observed in the index 0, 1, 2, 8, 9, which regard to the Generator 1, 2, 3, 9, 10, respectively.

Afterwards, show the DFS method applying and the DFS clustering result as followed:

```
######################
Branch tripped case 1
######################
#################################
#%%%%%%% DFS Method Report %%%%%%%#
#################################
Using the Numerical method
```

Figure 4.7: Clustering Report in Case 1

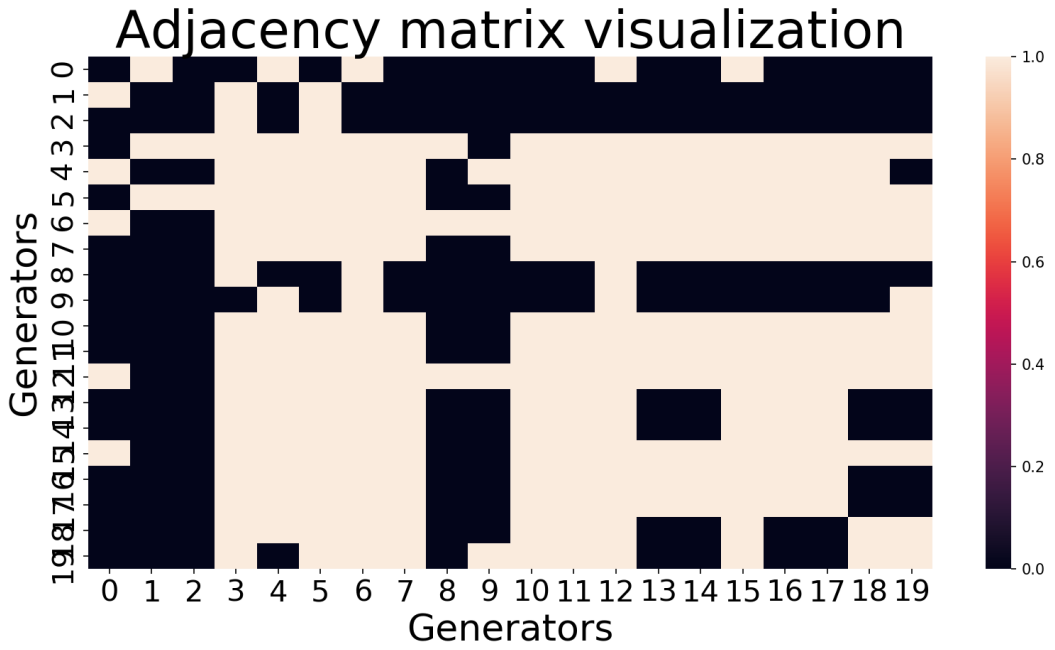


Figure 4.8: DFS Result in the Adjacency Matrix in Case 1

, where the X-axis and Y-axis has the same representation with the coupling-strength coefficient matrix showing in Figure.4.4, 4.5 and 4.6 before. And according to the report from Python Console window, this Case 1 just takes the first DFS strategy and detects the machine 1, 2, 3, 9 and 10 as the Critical Machines. The reason why the first strategy works can be explained by checking the Figure.4.6, where shows that the majority of the ΔH is smaller than -5 so that the first strategy can have a detection, then the rest strategies will not be activated any more. Besides, this result meets the calculation of coupling-strength coefficient difference that shows

in Figure.4.6, and can be double checked with the short-time PSS/E simulation in next section.

Checking Case 1 Clustering Results with PSS/E Simulation

Since for the analysis of the first-swing transient stability, the rotor angle unusual variation can be directly observed through the PSS/E short-time simulation. So the trajectory of the rotor angle can be used as the correct 'result' to test the entire machine Clustering algorithm. Then plot the trajectory of the rotor angles in Case 1:

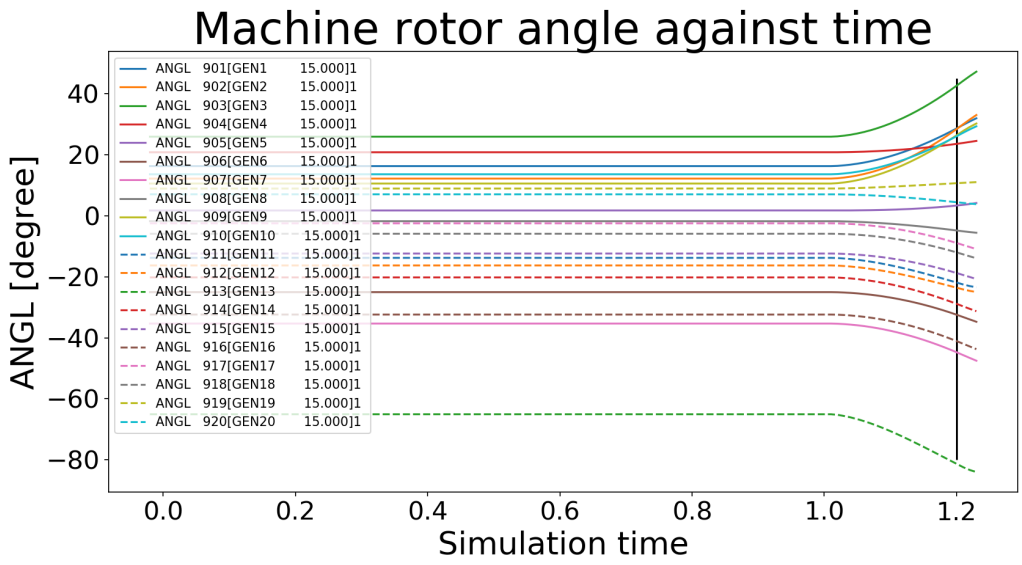


Figure 4.9: Machine rotor angle trajectory in Case 1

First of all, the vertical line in Figure.4.9 represents the fault clearing time in Case 1. And as what can be observed from the Figure.4.9, the five rotor angle trajectories that occur at the top of this figure, have unusual increase compared with the rest of machines. These increases started from the beginning of the during-fault and already have certain growths when the fault has been cleared. According to the legend on the left, these curves represent machine 3, 2, 1, 9 and 10, respectively (from top to down). And this is also what exactly the Clustering algorithm detected before. On the other hand, the dashed green line, who regards to the machine 13, also has an unusual decrease among all the machines. However,

since in the transient stability problem, the Critical Machines shall be those who are accelerated to asynchronism, rather than being decelerated. So this is the reason why machine 13 does not be selected as the Critical Machine by the DFS algorithm. Besides, a full-time domain simulation will be provided in the next chapter for the Case 1, then at that time the complete behavior of machine 13 can be observed, which could also validate this clustering result here. In summary, the clustering result for the branch fault Case 1 has been discussed and been proved correct.

4.7.2 Branch Fault Case 7

Branch fault Case 7, whose trip_index is 6 in Python, is the branch between Bus 1021 and Bus 1022. Since Bus 1022 also has connections with a tap-transformer and a fixed shunt, so its connecting situation is a little bit complicated with the first case. Then basing on what has been done in Case 1, check the first fundamental validations regarding to Y_{bg} matrix structure at first.

```
In [7]: Ybg.dot(E_dot)
Out[7]:
array([  0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
        -7.12596876 +34.51240975j,  -5.05992623 +25.19956003j,
       -10.38292497 +28.50052124j,  -7.27492169 +24.3764206j ,
         0.48428164 +11.09987251j,  11.26227783 +10.43952484j,
         6.48827419  +4.2137972j ,   4.43997739 +38.4201931j ,
        -5.41306907 +42.26616718j,  -6.45691609 +34.99089625j,
         4.25374115 +12.52468919j,   5.67941158 +14.65246885j,
         8.95380366  +6.24546987j,  17.69695519 +22.29406931j,
        26.7582966  +38.73943951j,  22.17531054 +16.46592042j,
        11.87322023 +17.876321j  ,  23.47787084 +38.86270178j,
        -2.91557161 +21.79216259j, -21.09655257+187.03748858j,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ,
         0.          +0.j        ,   0.          +0.j        ])
```

Figure 4.10: $Y_{bg} \times E^{'}$

```
In [8]: -Yaug_pre.dot(Vbus[:,100])
Out[8]:
array([-0.00000000e+00-0.00000000e+00j, -0.00000000e+00-3.55271368e-15j,
        -0.00000000e+00-0.00000000e+00j, -2.84217094e-14-0.00000000e+00j,
        -0.00000000e+00-7.10542736e-15j, -8.88178420e-16-7.10542736e-15j,
        -4.44089210e-16+7.10542736e-15j, -1.11022302e-16-0.00000000e+00j,
        -0.00000000e+00-0.00000000e+00j,  1.77635684e-15-0.00000000e+00j,
        -0.00000000e+00-7.10542736e-15j, -0.00000000e+00-0.00000000e+00j,
         1.42108547e-14-0.00000000e+00j, -0.00000000e+00-0.00000000e+00j,
        -1.42108547e-14-0.00000000e+00j, -3.55271368e-15-0.00000000e+00j,
        -0.00000000e+00-0.00000000e+00j, -0.00000000e+00-7.10542736e-15j,
        -7.10542736e-15-0.00000000e+00j, -0.00000000e+00-0.00000000e+00j,
        -0.00000000e+00+7.10542736e-15j, -3.55271368e-15-0.00000000e+00j,
        -7.12052869e+00+3.45115281e+01j, -5.05590154e+00+2.51988039e+01j,
        -1.03782075e+01+2.84996790e+01j, -7.27114538e+00+2.43748942e+01j,
         4.85944698e-01+1.10997400e+01j,  1.12637102e+01+1.04382811e+01j,
         6.48889454e+00+4.21301913e+00j,  4.44540067e+00+3.84200672e+01j,
        -5.40658134e+00+4.22672066e+01j, -6.45178213e+00+3.49917877e+01j,
         4.25561775e+00+1.25242130e+01j,  5.68154462e+00+1.46517841e+01j,
         8.95541061e+00+6.24481423e+00j,  1.76999655e+01+2.22921619e+01j,
         2.67623989e+01+3.87356026e+01j,  2.21778799e+01+1.64633874e+01j,
         1.18760569e+01+1.78747832e+01j,  2.34836827e+01+3.88600324e+01j,
        -2.91188826e+00+2.17935088e+01j, -2.10608922e+01+1.87054193e+02j,
        -0.00000000e+00+2.84217094e-14j, -0.00000000e+00-2.84217094e-14j,
        -0.00000000e+00-7.10542736e-15j, -0.00000000e+00+1.42108547e-14j,
         2.22044605e-16-1.77635684e-15j,  7.10542736e-15+2.84217094e-14j,
        -2.30926389e-14-1.77635684e-15j, -4.44089210e-16-1.77635684e-15j,
        -7.10542736e-15-0.00000000e+00j,  5.68434189e-14-0.00000000e+00j,
        -0.00000000e+00-1.42108547e-14j, -0.00000000e+00-1.42108547e-14j,
        -0.00000000e+00+1.42108547e-14j,  1.77635684e-15-3.55271368e-15j,
        -4.21884749e-15-2.13162821e-14j,  8.88178420e-15-1.77635684e-15j,
        -3.55271368e-15-1.42108547e-14j, -3.55271368e-14-7.10542736e-15j,
         1.06581410e-14+1.95399252e-14j, -3.55271368e-15-5.32907052e-15j,
         4.26325641e-14+1.06581410e-14j,  2.13162821e-14-3.55271368e-15j,
        -1.42108547e-14+1.42108547e-14j,  4.97379915e-14+1.77635684e-15j,
        -3.55271368e-14-2.84217094e-14j, -0.00000000e+00+1.42108547e-14j,
         2.84217094e-14-0.00000000e+00j,  5.68434189e-14-0.00000000e+00j,
         2.84217094e-14+2.13162821e-14j,  2.84217094e-14-2.84217094e-14j,
        -2.66453526e-15-0.00000000e+00j, -7.10542736e-15-0.00000000e+00j])
```

Figure 4.11: $-Y_{aug} \times V_{bus}$

According to the Figure.4.10 and 4.11, it can be observed that in the 'area_2' of them they have almost exactly the same value and at the same time the result vectors hold zero or very close to zero values in their area_1 and area_3. So the first fundamental validation is as expected and for the second one, print the maximum difference between the simulating $P_{e,ij}$ and calculating $P_{e,ij}$:

```
In [10]: np.max(Pee_ij - Pe_ij)
Out[10]: 0.00050649609416142 38
```

Figure 4.12: $P_{e,ij}$ comparison in Case 7

, which is also tiny enough as their max difference that can be ignored. So far, the Case 7 has met the two fundamental validations and can provide a preliminarily convincing clustering result. Afterwards, plot the pre-fault, post-fault coupling-strength coefficient matrix, and its difference matrix ΔH respectively as followed:

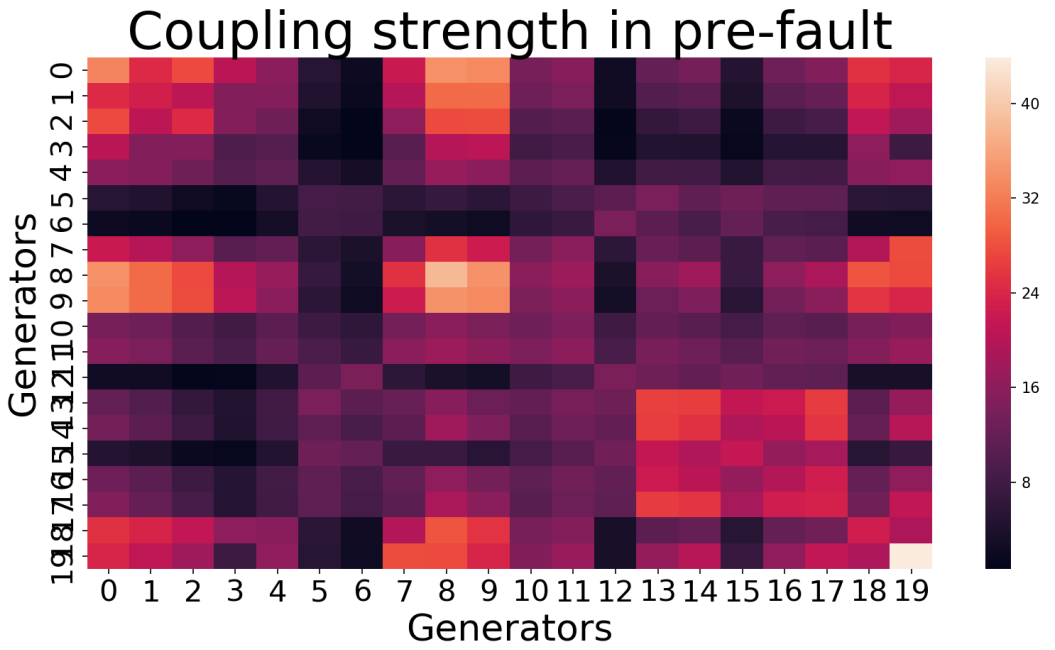


Figure 4.13: Coupling-Strength Coefficient Matrix in pre-fault for Case 7

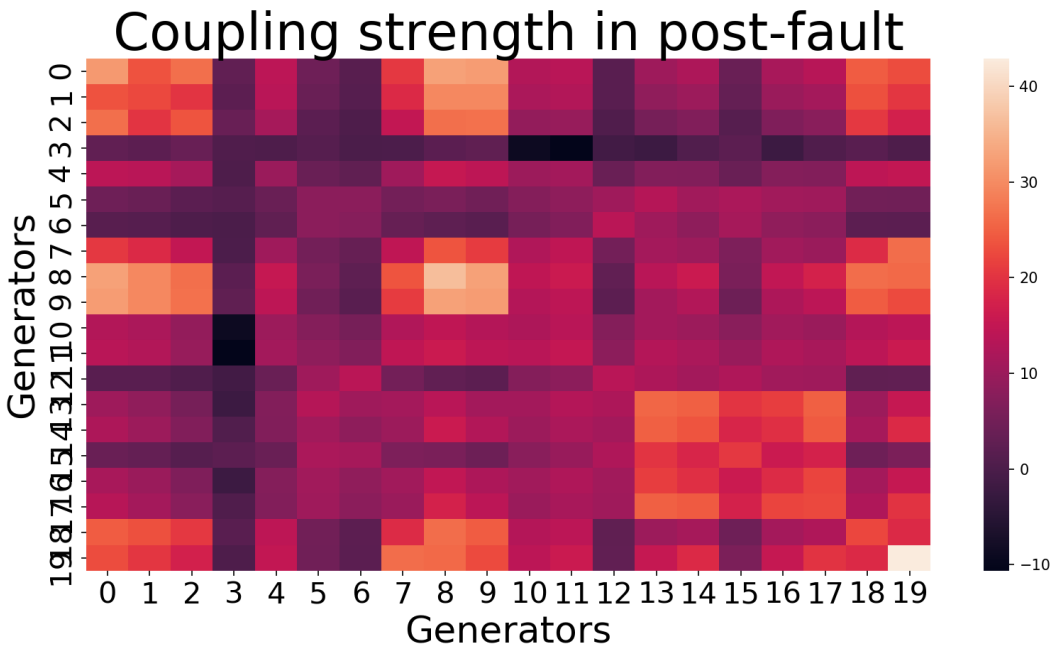


Figure 4.14: Coupling-Strength Coefficient Matrix in post-fault for Case 7

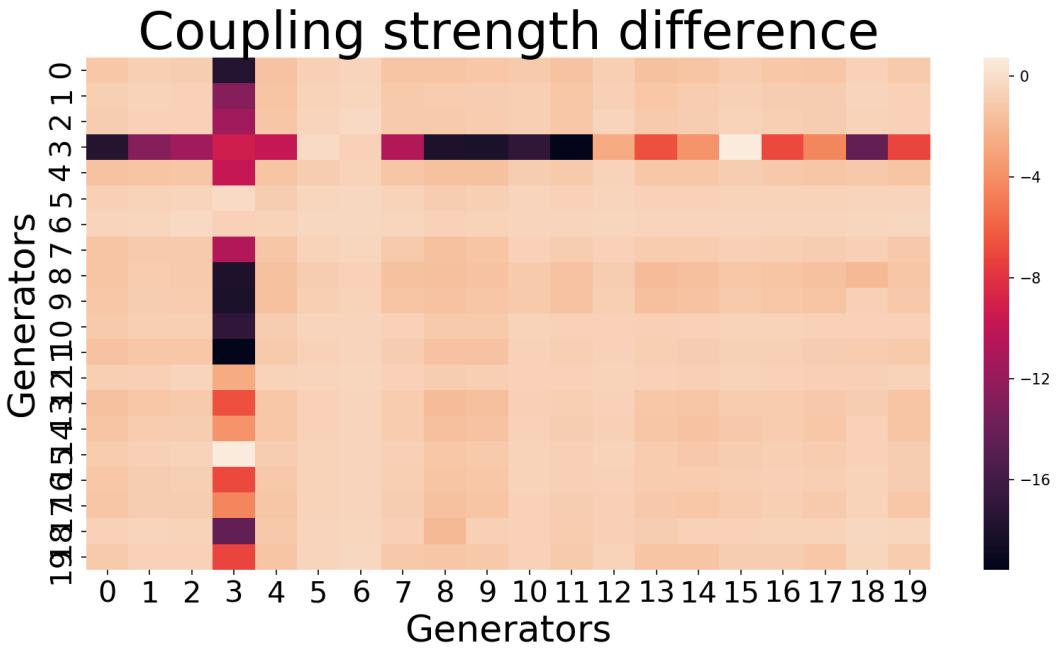


Figure 4.15: Coupling-Strength Difference Matrix for Case 7

Then according to the legend showing up in the Figure.4.15, the coupling-strength coefficients after the disturbance both have varying degrees of decline. However,

the most obvious one among all the machines, is the machine 4 (Python index 3) who has the most deep colors in its column. Afterwards, show the applying DFS method and the DFS Clustering result as followed:

```
####################
Branch tripped case 7
####################
##################################
#%%%%%%% DFS Method Report %%%%%%%#
##################################
Using the Extended numerical method
```

Figure 4.16: Clustering Report in Case 7

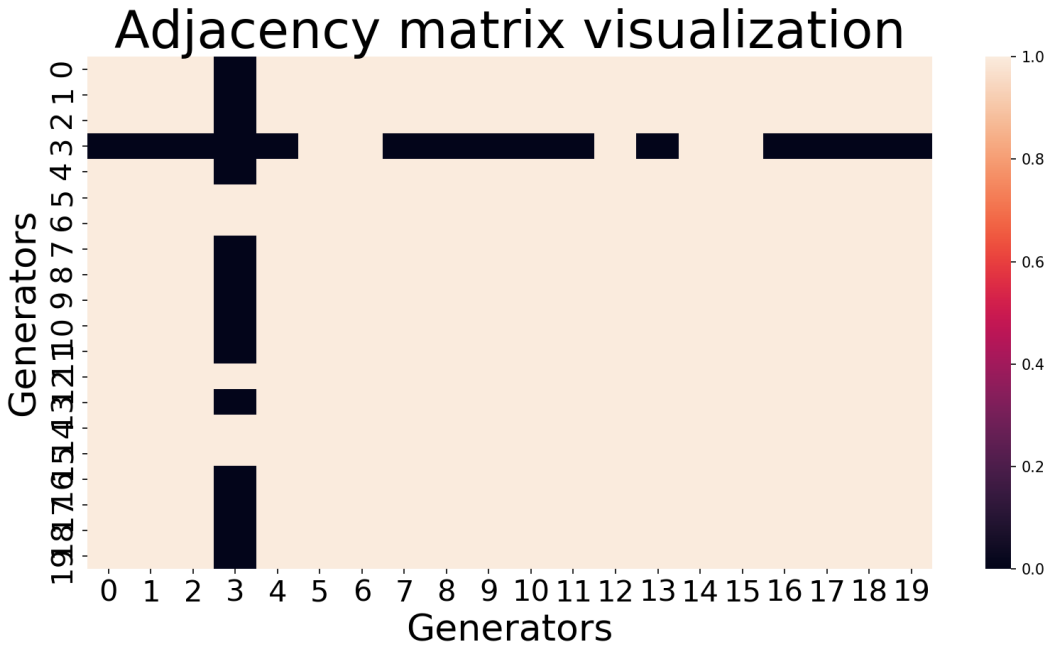


Figure 4.17: DFS Result in the Adjacency Matrix in Case 7

As what can be observed from the Python Console report in Figure.4.16 and the DFS Clustering result in Figure.4.17. The machine 4 (Python index 3) has been detected as the only Critical Machine in the Case 7, and since the majority of the decline of coupling-strength coefficient is not smaller than -5, so the 'Extended numerical method' has been activated as expected.

Checking Case 7 Clustering Results with PSS/E Simulation

Afterwards, plot the short-time PSS/E simulation in Figure.4.18 for the Case 7 Clustering validation.

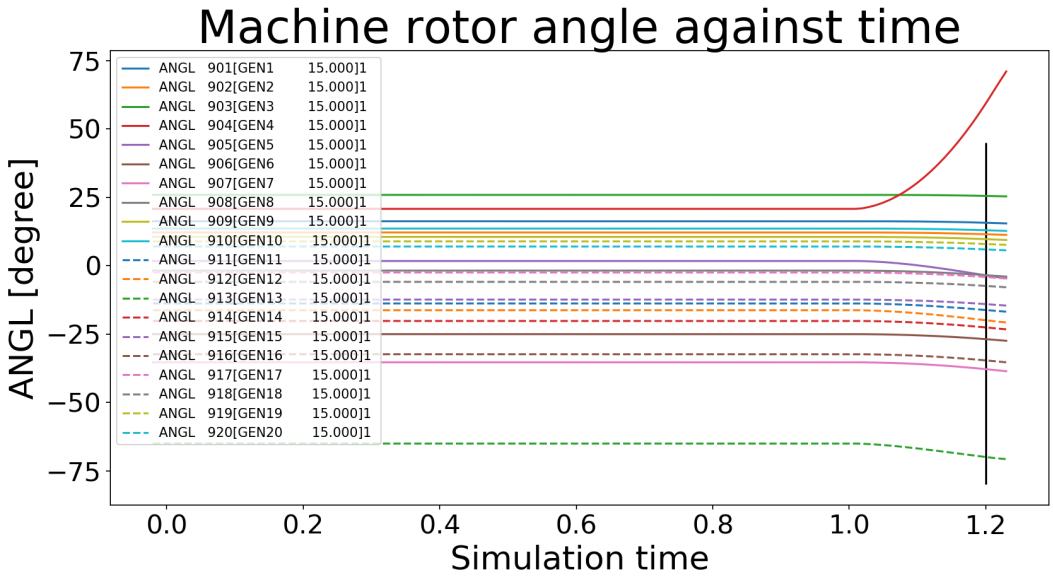


Figure 4.18: Machine rotor angle trajectory in Case 7

From what can be observed in Figure.4.18, that the machine 4 is the only one who has an unusual increase in its rotor angle, which makes it the only Critical Machine. And the rest of machines just keep steady and shall be clustered as the Non-Critical ones. This observation result from Figure.4.18 meets the Clustering result in Figure.4.17, which proves its correctness.

4.7.3 Branch Fault Case 19

Then it is the last branch fault case that being tested in Chapter 4 and 5. The branch Case 19, whose trip_index is 18 in Python, is the branch between Bus 2031 and Bus 2032. In this time both two terminals of this branch are connecting to fixed shunts and transformers. As same as what has been done to Case 1 and Case 7, check the first fundamental validations regarding to Y_{bg} matrix structure at first.

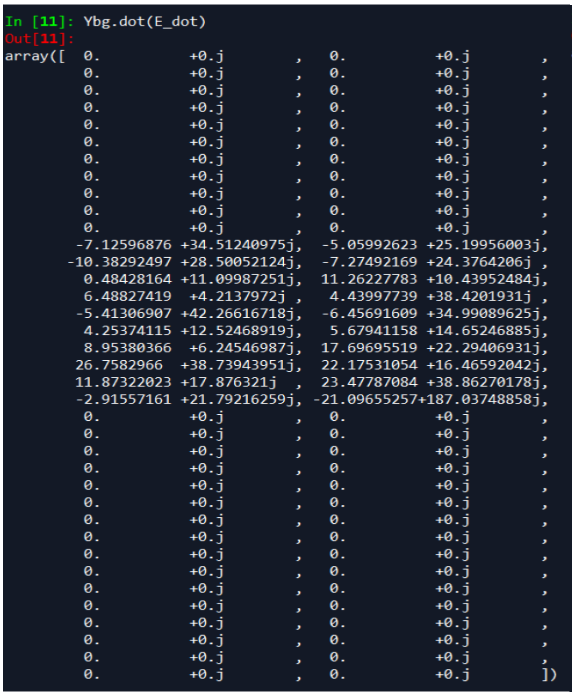


Figure 4.19: $Y_{bg} \times E^{'}$

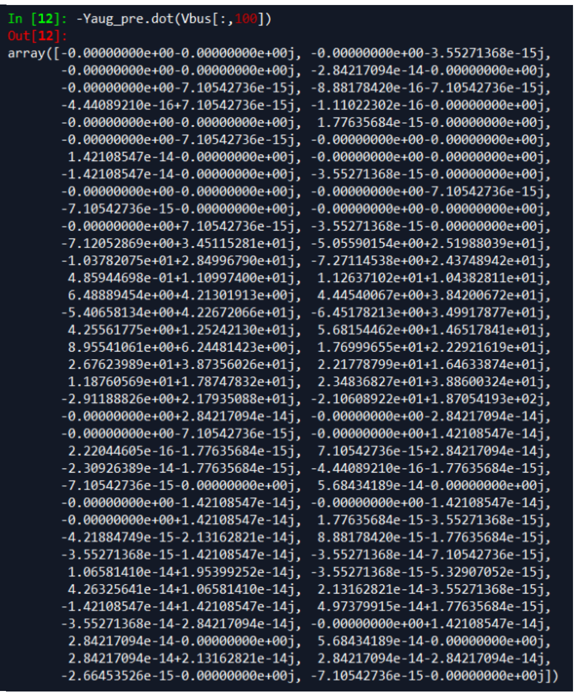


Figure 4.20: $-Y_{aug} \times V_{bus}$

According to the Figure.4.19 and 4.20, it can be observed that in the 'area_2' of them they have almost exactly the same values and at the same time the result vectors hold zero or very close to zero values in their area_1 and area_3, respectively. So the first fundamental validation is as expected. And for the second one, print the max difference between the simulating $P_{e,ij}$ and calculating $P_{e,ij}$:

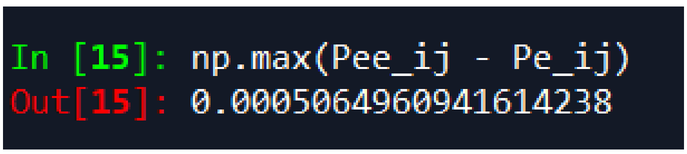


Figure 4.21: $P_{e,ij}$ comparison in Case 19

, which is also tiny enough as their max difference that can be ignored. So far, the Case 19 has met the two fundamental validations and can provide a preliminarily convincing clustering result. Afterwards, plot its pre-fault, post-fault coupling-strength coefficient matrix, and the difference matrix ΔH respectively as followed:

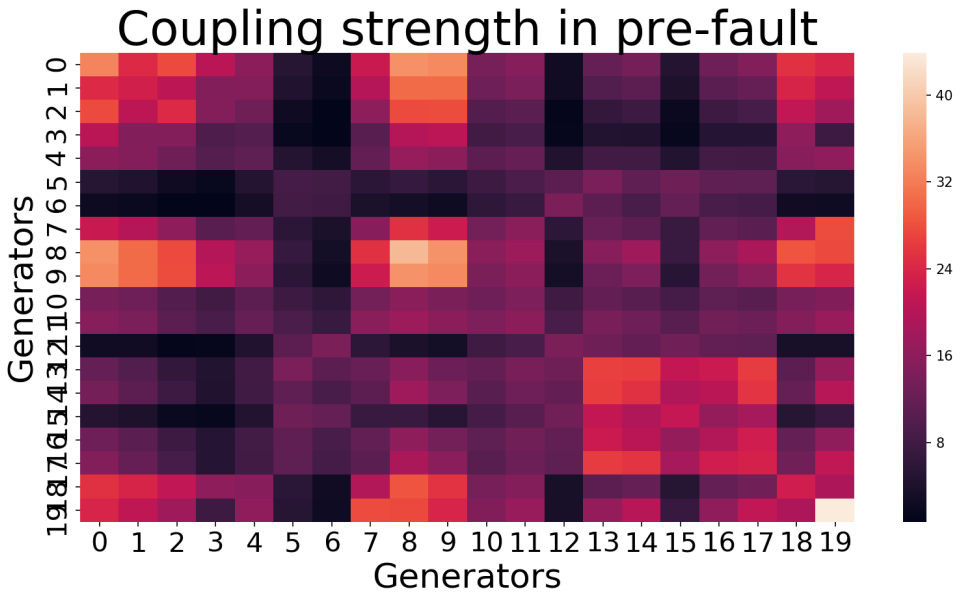


Figure 4.22: Coupling-Strength Coefficient Matrix in pre-fault for Case 19

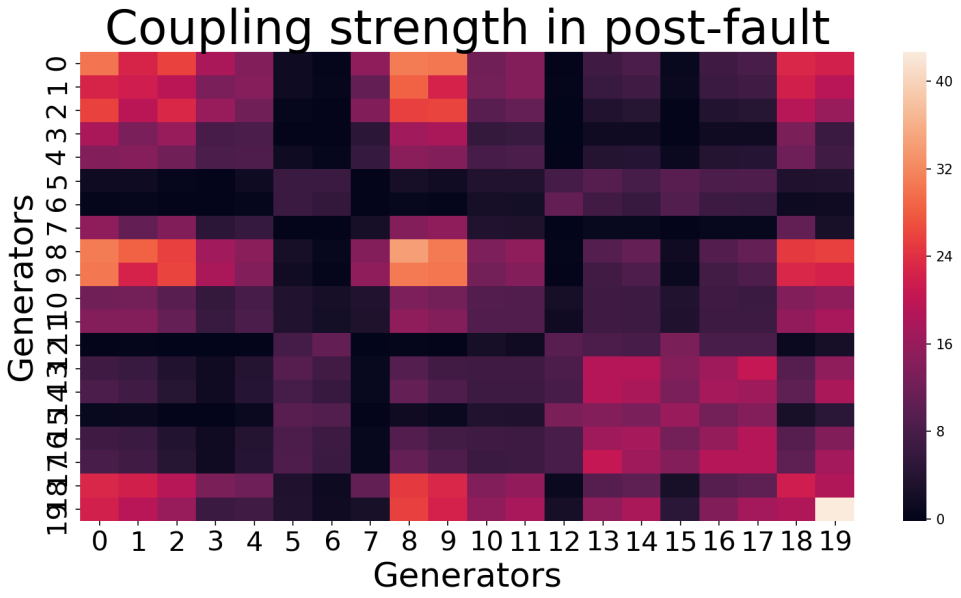


Figure 4.23: Coupling-Strength Coefficient Matrix in post-fault for Case 19

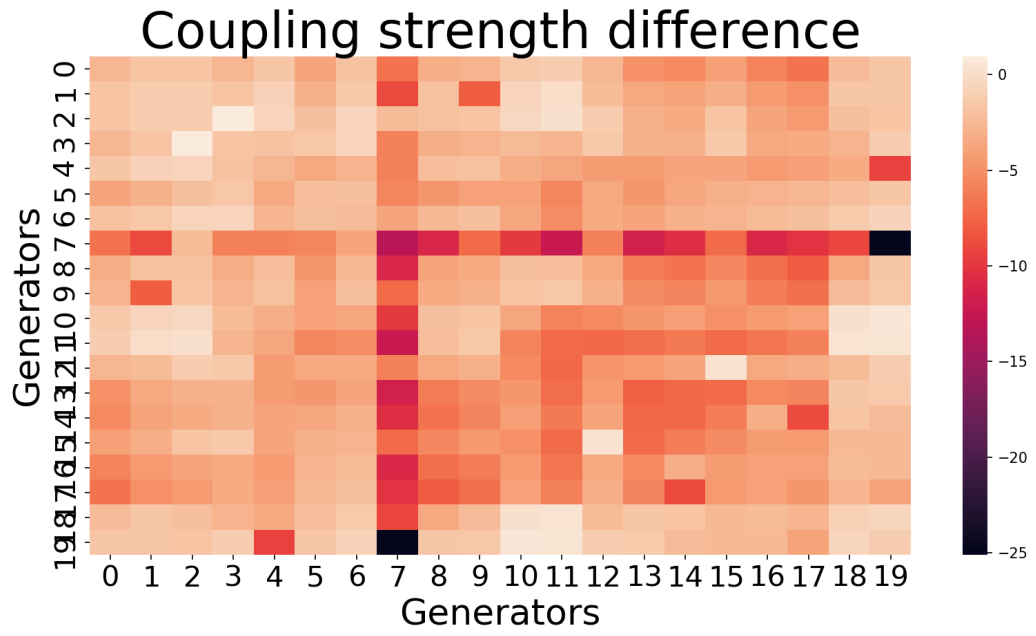


Figure 4.24: Coupling-Strength Difference Matrix for Case 19

Then according to the legend showing up in the Figure.4.24, that the majority of the ΔH elements are less than zero, which means all the relations between machines in the system have been weakened. And among all these colors it can be observed that there is a clearly deep color line appearing in the location of machine 8 (Python index 7), which takes the majority of the negative values. Afterwards, show the applying DFS method and the DFS Clustering result as followed:

```
######################
Branch tripped case 19
######################
####################################
#%%%%%%%% DFS Method Report %%%%%%%%#
####################################
Using the Numerical method
```

Figure 4.25: Clustering Report in Case 19

Since for the relations regarding to the machine 8 have a lot of values that less than -5, so the first DFS method, the 'Numerical method' can directly work without activating the rest of strategies. Meanwhile, by observing the Figure.4.26,

it is clearly that the machine 8 has been detected as the Critical Machine, which follows the intuition when observing the Figure.4.24.

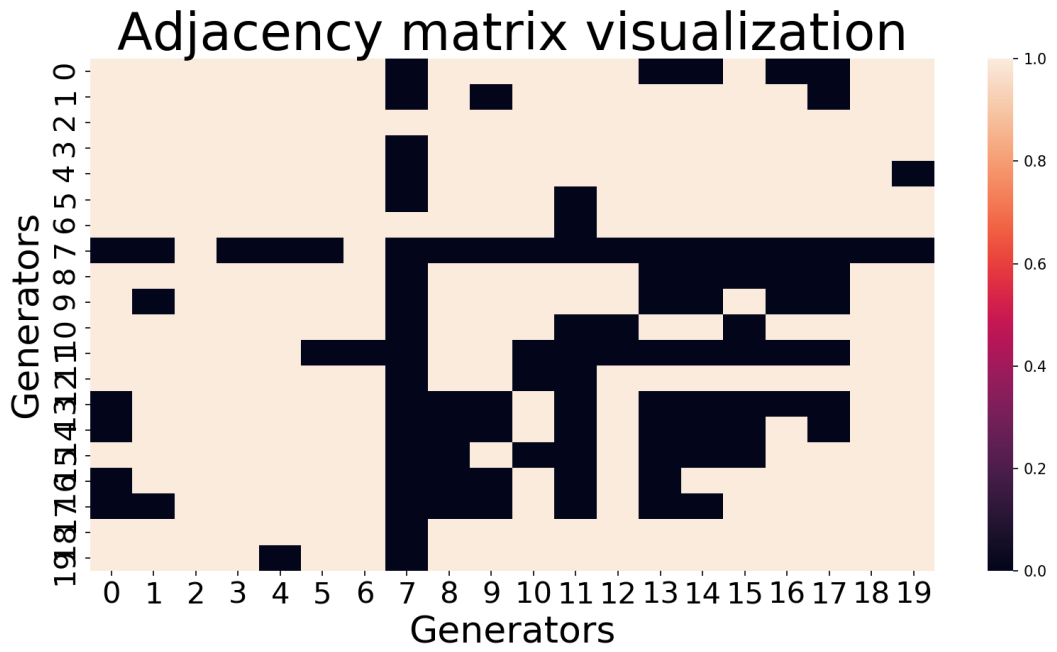


Figure 4.26: DFS Result in the Adjacency Matrix in Case 19

Checking Case 19 Clustering Results with PSS/E Simulation

Afterwards, plot the short-time PSS/E simulation in Figure.4.27 for the Case 19 clustering validation.

As what can be observed from Figure.4.27, that the trajectory of machine 8 has a dramatical slope compared with all other machines, which is the reason why it has been detected as the Critical Machine by the Clustering algorithm. While on the other hand, the rotor angles of machine 12 and 13 (Python index 11 and 12, respectively) also have different degrees of variations, but they are both not been detected as the Critical Machines. This can be explained in two ways, respectively. For the machine 12 (Python index 11), some of it relations has already been turned in Figure.4.26, however, these impaired relations are not enough to make the machine 12 a Critical Machine. And for machine 13 (Python index 12), though it has an apparent decrease in its trajectory, however, for a Critical Machine in the

transient stability analysis, the rotor angle shall face the harm of being increasing rather than decreasing. So the machine 13 should not be clustered as the Critical one even for its unusual variation. Furthermore, in this case, the correctness of the clustering result can also be double checked with the full-time domain simulation, along with the OMIB method implementation in the next chapter.

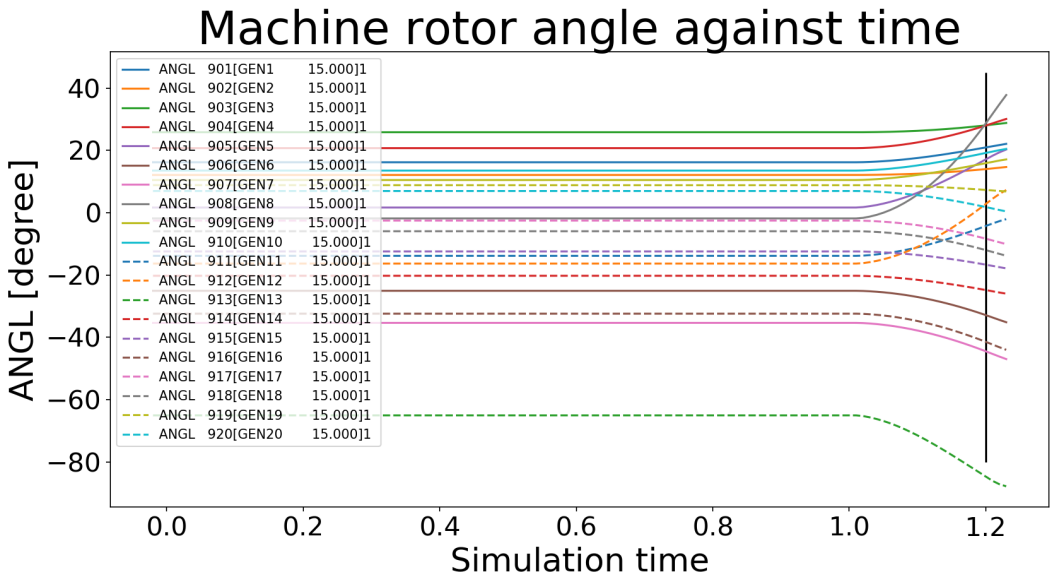


Figure 4.27: Machine rotor angle trajectory in Case 18

4.8 Chapter Summary

So far, in this chapter, the implementation of state of the art Clustering method has been elaborated and the results of three different cases have also be presented and discussed. In the next chapter, these three Cases (Case 1, 7 and 19) will keep going through the OMIB algorithm for the final transient stability analysis, by using the Clustering results achieved in this chapter. Besides, since the trajectory in Case 19 is not very clear, so the Clustering result of Case 19 will be further validated by its full-time domain simulation in the next chapter. But for the Case 1 and Case 7, the correctness of their Clustering results have been already validated basing on the existing data.

CHAPTER 5

OMIB Method Implementation

The basic theories and the mathematical expressions of the OMIB have been elaborated in the section 2.6. The implementation in this chapter will mainly focus on the programming level where to achieve all those targets mentioned in section 2.6.

5.1 OMIB Implementation

5.1.1 Angle Crossing Situation

The first problem need to be concerned is about the one machine equivalent (OME) rotor angle whose corresponding active power $P_{e_{OMIB}}$ exceeds its corresponding mechanical power $P_{m_{OMIB}}$. Although the major cases are those that $P_{e_{OMIB}}$ exceeding $P_{m_{OMIB}}$ happens just after the fault clearing time However, in some special cases, when the relay reaction time is quite short that the $P_{e_{OMIB}}$ in the post-fault has not 'grown' large enough to exceed the $P_{m_{OMIB}}$. The reason to refine this term, is for the calculation of the acceleration power, the A_{acc}, whose upper limit of its integral is this crossing angle.

To detect which crossing case it is in Python, a strategy shows in the flow chart in Figure.5.1 will need to be implemented. First of all, the index regarding to the fault clearing time in the entire time series will be recorded. Then compare the $P_{e_{OMIB}}$ and $P_{m_{OMIB}}$ corresponding to this 'index' and find if $P_{e_{OMIB}}$ is larger. And if the result is positive, then this is a normal crossing case that the corresponding active power will exceed its mechanical power just after the fault clearance. However, if the result is negative, then the crossing angle shall be calculated by the equilibrium between the $P_{e_{OMIB}}$ and $P_{m_{OMIB}}$, and then find out the corresponding 'crossing_index' by checking the δ_{OMIB} series. Finally, in the stability report in Python Console window, the case of 'angle crossing' will also need to be published for the convenience of the analysis.

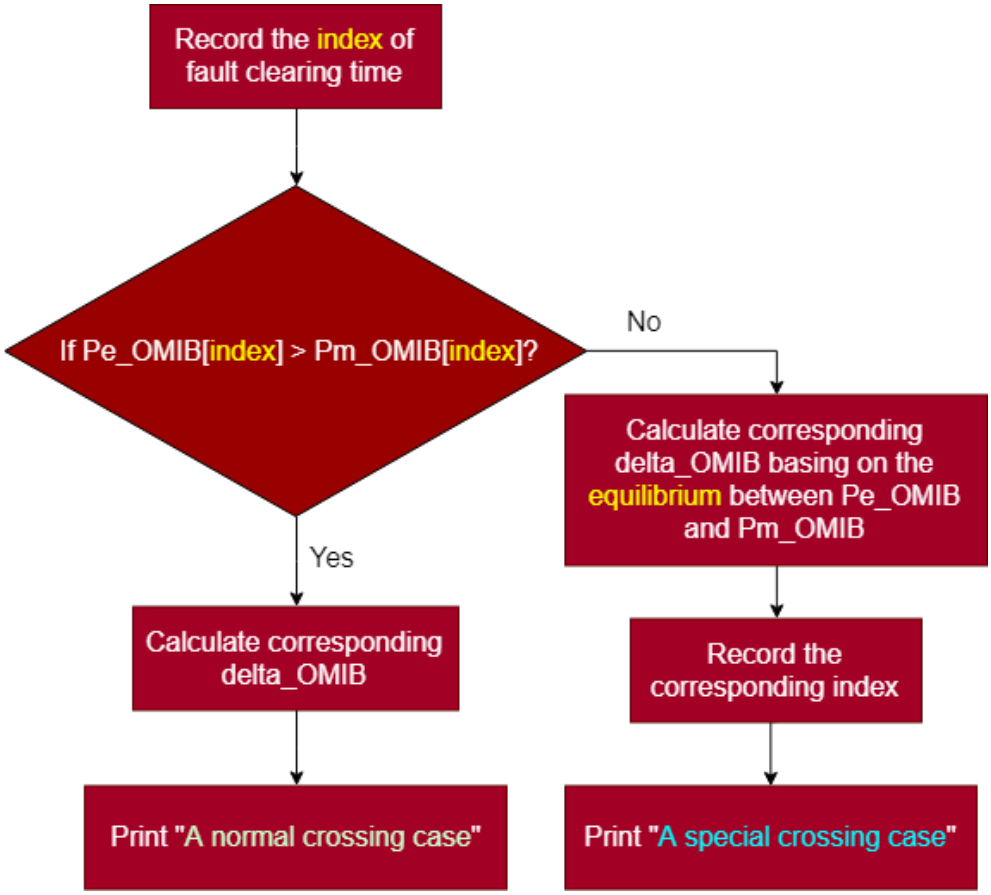


Figure 5.1: Flow chart of the crossing angle calculation

5.1.2 Returning Angle Calculation

Then the second problem is regarding to the returning angle calculation. The idea for calculating the returning angle is basing on the Repeated Root Iterative Method, which has been elaborated in section 2.6.3. The Repeated Root Iterative Method is achieved by an extra function called '**func_solver**' that can be imported into the '**OMIB**' function. The only thing here need to be mentioned is about the setting of the maximum iteration steps and calculation tolerance. Basing on several tests of the '**func_solver**', when the maximum iteration step is set to 100 times and the tolerance to be 10^{-5}, the calculating accuracy can be guaranteed and at the same time, the time spent for the iteration will not be too much.

Besides, since the arithmetic solution sometimes can return a result with nonsense, so the feasible boundary of the solution shall be limited to test if the arithmetic solution has a physical meaning or not. According to the theories mentioned in section 2.6.2, 2.6.3 and 5.1.1, the value of the calculating returning angle shall between the crossing angle δ_{cross} and the unstable critical angle δ_u. If the arithmetic solution given by the '**func_solver**' exceeds this boundary, then this solution can be treated as invalid, which means, in this case, there is no available returning angle.

The details of how to constructing the iteration function can be found in the appendix.{Transcendental Function Solver}.

5.1.3 Stability Situation Judgement

Afterwards, the next problem regarding to the OMIB implementation is about the stability judgement. The criterion of the first-swing transient stability judgement in this project is basing on if there is a valid returning angle in this OME, which in Python script is about '**if the maximum iteration step has been matched**' or '**if the arithmetic solution of returning angle exceeds its physical boundary**'. As long as there is one positive answer to these two questions, then the entire system will be judged as instability, otherwise, it shall be a stable case. Then the corresponding margin value can be calculated basing on the theories mentioned in section 2.6.4.

In addition, the last thing need to be explained here is about the reason why calculating the stability margin value after knowing the stability situation of the system. This seems like a detour, but however, as there a situation called 'Marginal Stability' existing in the judgement (section 2.6.4), so the calculation of the margin value is necessary since this situation can only be detected by the small calculated marginal values.

So far, the places that need to be paid attention to in the OMIB implementation has been elaborated. The entire OMIB algorithm is written in the function called '**OMIB**' and can be found in the appendix.{OMIB}.

5.2 OMIB Method Testing with PSS/E

After the orientations of the OMIB implementation, in this section this paper will keep going on the transient stability analysis basing on the Clustering results achieved in section 4.7 for Case 1, 7 and 19, respectively.

5.2.1 Stability Analysis in Case 1

Undertaking the section 4.7.1, running the entire program and then publish the stability report for Case 1 from the Python Console window as below:

```
######################
Branch tripped case 1
######################
##################################
#%%%%%%% DFS Method Report %%%%%%%#
##################################
Using the Numerical method
#########################################################################
#%%%%%%%%%%%%%%%%%%%%%%%$$$ Assessment Report $$$%%%%%%%%%%%%%%%%%%%%%%%#
#########################################################################
A normal crossing case
Cross angle =  0.298227848331
Critical unstable angle 2.84336480526
Returning angle, delta_r_OMIB =  0.891255284368021
Total iteration steps =  3
Aacc < Adec, their is a returning angle = 0.891
#%%%%%%%%%%%%%%%%%%%%%%%%#
This is a stable case :)
#%%%%%%%%%%%%%%%%%%%%%%%%#
Margin_stable =  16.0889409782574
Running time for assessment: 2.396408 sec
```

Figure 5.2: Transient stability analysis report for Case 1

As what can be observed from the Figure.5.2, by taking the Clustering result from section 4.7.1, the OMIB algorithm has been implemented and returned several data that help doing the analysis. First of all, since these is a valid arithmetic solution for the OME returning angle, so the case is judged as a stable case.

This case is detected as a 'normal crossing case' so its active power exceeds its mechanical power just after the fault clearing time. Then the returning angle is calculated as 0.891 rads, which is between the cross angle 0.29823 rads and the critical unstable angle 2.8434 rads. This means that the acceleration of this OME rotor angle has been stopped far away from its unstable critical angle. Then the rotor just rolled back to its synchronism, which left a very positive value for its margin, 16.0889. And to have a final confirmation, the PSS/E needs to run a full-time domain simulation for Case 1, which in this project, is the time series from the beginning to the 10 second.

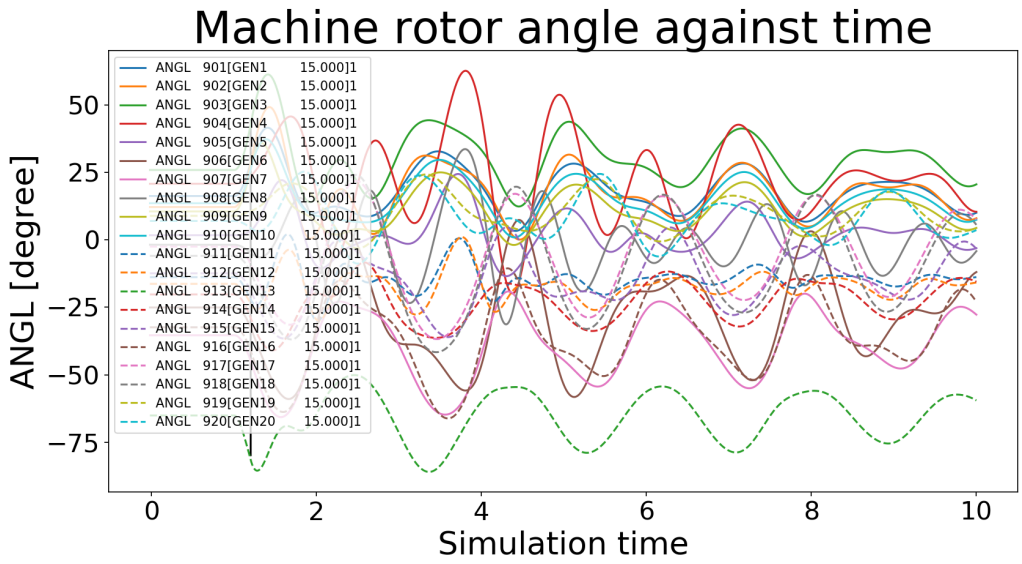


Figure 5.3: Full-time domain simulation for Case 1

As what can be observed from the full-time domain simulation for Case 1 in Figure.5.3, although there are somewhere that the rotor angles variate dramatically, however, the variations of the rotor angles finally dissipate and back to approach their starting points. Furthermore, the relatively dramatic variations among all the machines happen on those who have been clustered into the Critical group, the machine 1, 2, 3, 9 and 10, which validates the correctness of the Clustering result in section 4.7.1 from the side.

5.2.2 Stability Analysis in Case 7

Undertaking the section 4.7.2, run the entire program and then publish the stability report for Case 7 from the Python Console window as below:

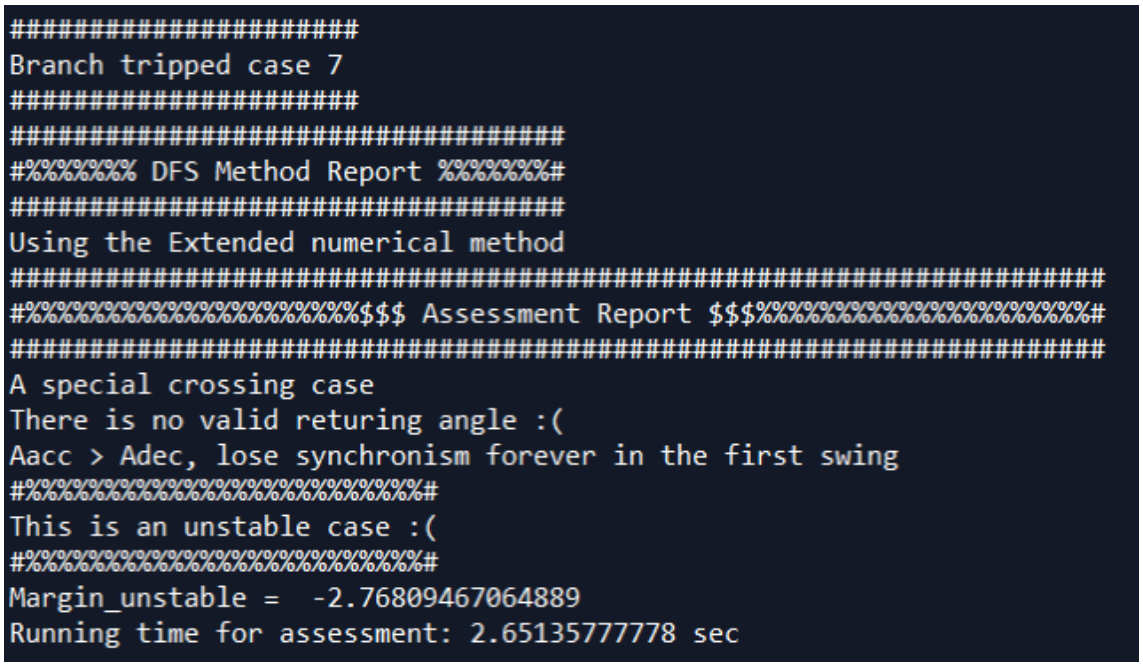


Figure 5.4: Transient stability analysis report for Case 7

As what can be observed from the Figure.5.4, by taking the Clustering result from section 4.7.2, the OMIB algorithm has been implemented and returned several data that help doing the analysis. Unfortunately, according to the stability analysis report from the Python Console window, there is no valid arithmetic solution for the returning angle, which means the rotor was accelerating for the whole time and directly lost its synchronism in the first swing. After judging the Case 7 as an unstable case, then calculate its stability margin value and get a negative solution, -2.7681, as expected. Besides, this case has been detected as a 'special crossing case', which means its active power will not exceed its mechanical power just after the fault clearing time.

Then, to have a final confirmation, the PSS/E needs to run a full-time domain simulation for Case 7, which in this project, is the time series from the beginning to the 10 second.

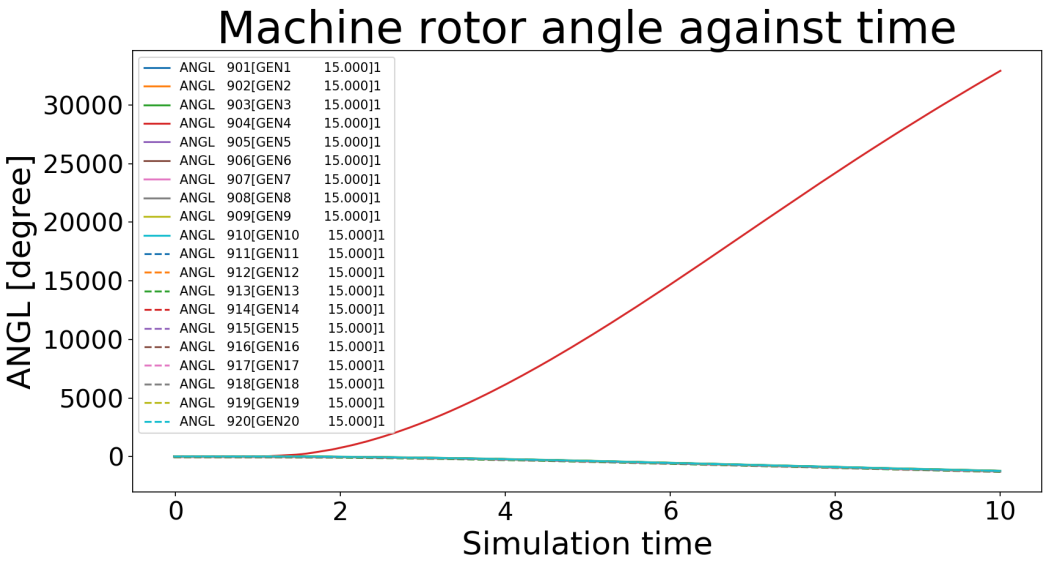


Figure 5.5: Full-time domain simulation for Case 7

The observation of Figure.5.5 for Case 7 can be easier compared with the Figure.5.3 for Case 1. First of all, the machine 4 is the one who lost its synchronism and has already been clustered in to the Critical-Machine group in section 4.7.2, which validates the correctness of its Clustering result directly. Secondly, since there is one machine that lost its synchronism forever, so the OME of this entire system under Case 7 can be treated as 'first swing instability', which also validates its transient stability analysis result from the OMIB algorithm at the same time.

5.2.3 Stability Analysis in Case 19

Undertaking the section 4.7.3, run the entire program and then publish the stability report for Case 19 from the Python Console window as below:

```
######################
Branch tripped case 19
######################
#################################
#%%%%%%% DFS Method Report %%%%%%%#
#################################
Using the Numerical method
#########################################################################
#%%%%%%%%%%%%%%%%%%%%%%%$$$ Assessment Report $$$%%%%%%%%%%%%%%%%%%%%%%%#
#########################################################################
A normal crossing case
Cross angle =  0.607302993652
Critical unstable angle 2.53428965994
Returning angle, delta_r_OMIB =  2.15820710797583
Total iteration steps =  4
Aacc < Adec, their is a returning angle = 2.158
#%%%%%%%%%%%%%%%%%%%%%%%%%%#
This is a stable case :)
#%%%%%%%%%%%%%%%%%%%%%%%%%%#
Margin_stable =  0.275685287143731
This is a marginal stability situation, so the full time domain simulation will be
needed.
Running time for assessment: 2.17307688889 sec
```

Figure 5.6: Transient stability analysis report for Case 19

As what can be observed from the Figure.5.6, by taking the Clustering result from section 4.7.3, the OMIB algorithm has been implemented and returned several data that help doing the analysis. Above all, since there is a valid arithmetic solution for the returning angle, 2.1582, so the Case 19 has been judged as a stable case at first. However, as what can be observed, the calculated returning angle is very closed to the critical unstable angle, 2.5343, which means the operating situation is very closed to the unstable boundary. This also explains the reason why the calculated margin value is so small. Furthermore, according to the statement in section 2.6.4, since the calculated margin value of Case 19 is lower than 2, therefore it will be treated as the 'marginal stable', because of its 'potential instability'.

Afterwards, since there is still a uncertain conclusion in section 4.7.3 about the Clustering result for Case 19, so the trajectories of Case 19 will be plot into two

versions. The first one will be in a small time scale just for the better observation, and the second one is the PSS/E full-time domain simulation trajectories, which in our project, is the time series from the beginning to the 10 second.

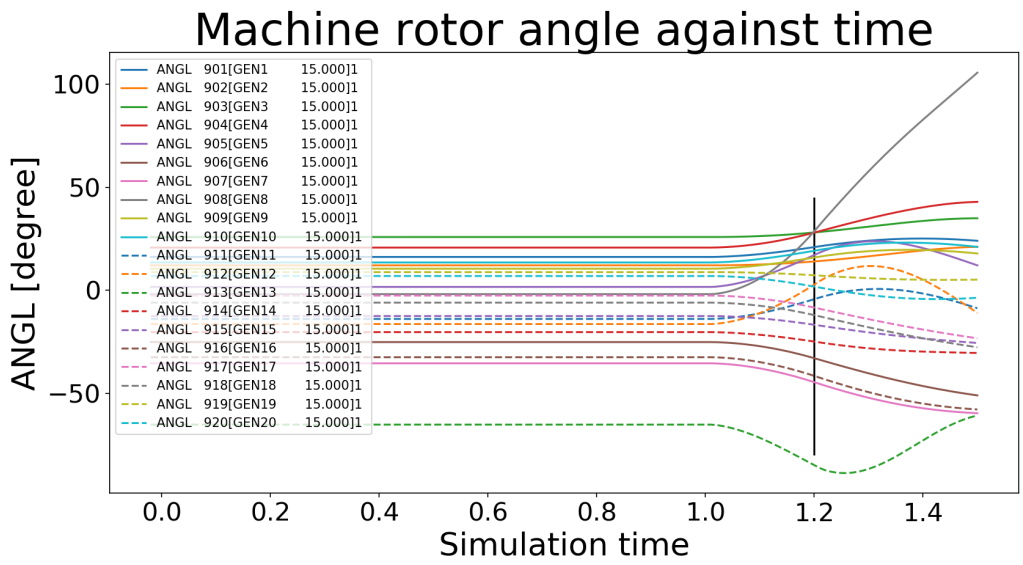


Figure 5.7: Part of the full-time domain simulation for Case 19

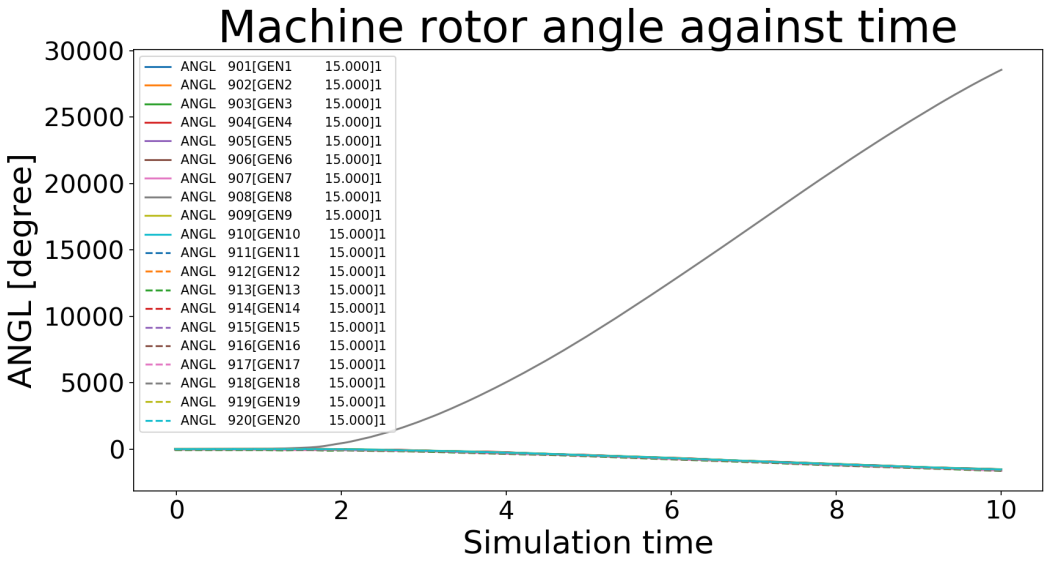


Figure 5.8: Full-time domain simulation for Case 19

First of all, as what can be observed from the Figure.5.7, though machine 12 and 13 both have unusual variations in the beginning, however, these variations dissipated quickly before 1.4 [sec]. While the machine 8 just kept being accelerated till

its unstable critical angle and then lost its synchronism forever. This visualization result can validate the Clustering result from section 4.7.3 thoroughly. But on the other hand, the 'losing synchronism' situation also proves the importance of the full-time domain simulation when the case has been detected as a 'marginal stable'. Furthermore, this result also proves that why the term 'Marginal Stable' should exist.

5.3 Chapter Summary

So far, in this chapter the implementation of the 'One Machine Infinite Bus' (**OMIB**) in the programming level has been elaborated. And the stability analyses of the Case 1, 7 and 19 have also been achieved basing on their Clustering results from the previous chapter. The purpose of taking the full-time domain simulation in PSS/E is to validate the OMIB algorithm implemented in this chapter, however, at the same time, it can also provide an irrefutable evidence when checking the Clustering results from the previous chapter. Further, both the Clustering result and stability analysis of each case discussed in this chapter and Chapter 4 have been proved correct, which means the entire algorithm can somehow guarantee its correctness. However, to find out how robustness this 'Clustering & Assessment' system is, more branch faults' tests will need to be implemented and discussed in the following chapter.

CHAPTER 6

Assessment Automation with Every One Fault in Each Branch

6.1 Automation Implementation

The purpose of the code automation is to build a smarter system that can implement all the possible branch faults automatically. This application can be beneficial for the offline preventive inspection or even the online instant check with the data collecting devices such like the PMUs and the SCADA. Since the entire program has already been divided into several 'main parts' during the programming, so the automation is actually not hard to be achieved and can be described as the flow chart showing in Figure.6.1 below:

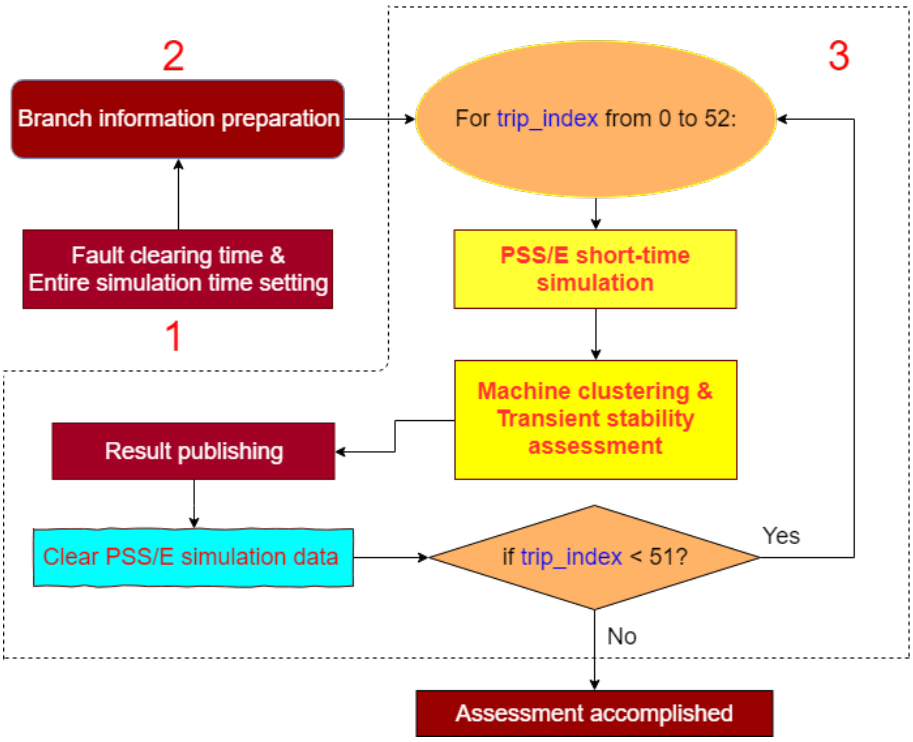


Figure 6.1: Flow chart of the code automation

As what can be observed from Figure.6.1, the first and second step are the time setting and the branch information loading, respectively. And the third step inside the dashed block is the automation part that combines the PSS/E short-time simulation with the Clustering & Assessment. Since step one and two are both the generalized setting, so they should be stated clearly in the beginning and be outside of the automation part. Furthermore, the automation part is constructing inside a 'for loop' that holds different 'trip_index' who corresponds to different branch fault cases. The loop variable, the 'trip_index' is from 0 to 51 and will be sent to the two yellow blocks, respectively.

The first yellow block is regarding to the PSS/E simulation. Its Python function is called '**PSSE_simulation.py**' and can be found in the appendix.{PSS/E Simulation}. The second yellow block is regarding to a highly integrated function that combines the 'Admittance matrix generation', 'Coupling-strength clustering' and the 'OMIB stability analysis'. However, since for some special cases, they are related to the multi-swing problem, which can result in the malfunction in the OMIB algorithm due to its feasible domain. So the code for the OMIB part has been 'commented' for now and the automation in this project is only about the Machine Clustering. Its function name is '**Assesment_function.py**' and can be found in the appendix.{Assessment}. Then after publishing the result from the images visualization and Python Console window, the simulation data in PSS/E will need to be cleared for making space for the next branch fault case. The main file that implements the code automation is called '**Automation.py**' and can be found in the appendix.{Automation}. In final, after all 52 cases have been simulated and clustered, the entire assessment is accomplished and the program will be shut down.

6.2 30 among 52 Cases Analysis

The entire running time for clustering the 52 cases in our system can be achieved from the Python Console window report:

Then pick up 30 random cases from the overall and implement the OMIB algorithm to them basing on their Clustering results. Afterwards, the results can be divided into two layers, the first one is regarding to Clustering Correctness, and the second

```
Running time for 52 cases' clustering: 157.195555556 sec
```

Figure 6.2: Automatic clustering running time

one is corresponding to Stability Analysis Correctness. Since the crucial part of this project is about Machine Clustering, so the priority of the first layer is much more important than the second one in this project. Then after accomplishing the analyses for the 30 random cases, plot their results' distribution in the 'tree' diagram in Figure.6.3.

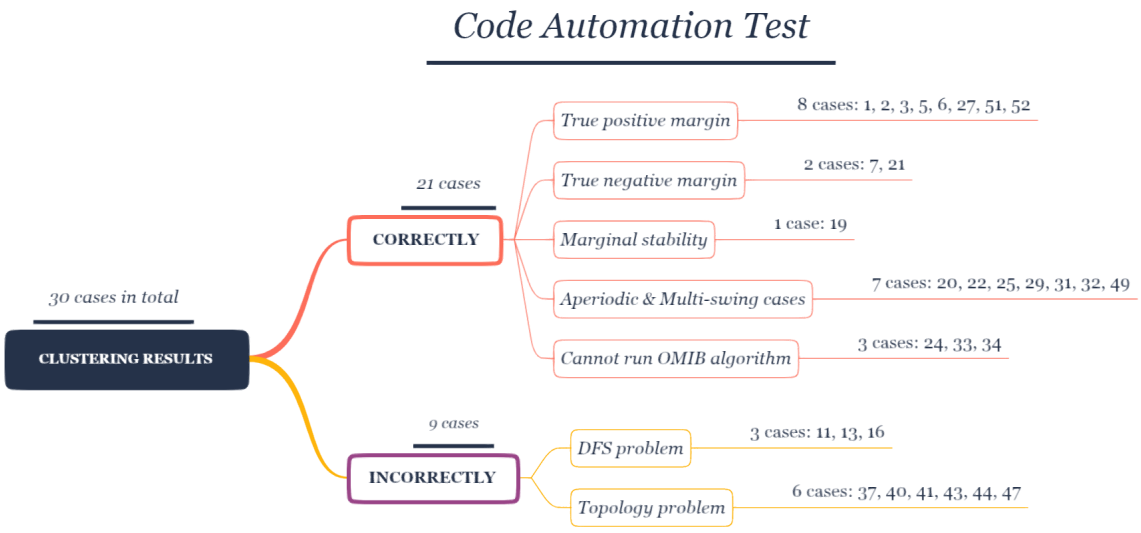


Figure 6.3: Results distribution for 30 random cases among the overall 52 cases

As what can be observed from the results' distribution showing in Figure.6.3, 30 cases' results have been separated into two groups basing on their clustering correctness in the first layer. And in the further second layer, for those who have been clustered correctly, the second layer will keep going on their transient stability analysis by using the OMIB algorithm and the Clustering results received before. On the other hand, for those who cannot be clustered correctly, the second layer will mainly focus on the discussion and speculation of the reasons behind them. Besides, basing on the testing results from the 30 cases, 21 cases can be detected correctly. So under this selected Nordic-20 Generators' system, the accuracy of this Clustering algorithm can be predicted roughly, which is $\frac{21}{30}$, 70%

6.2.1 When the Clustering is Correct

When the case has been clustered correctly through the Coupling-Strength algorithm, then its Clustering result can be applied for the OMIB algorithm to implement its transient stability assessment. Basing on the differences between results, they can be divided into 'True positive margin', 'True negative margin', 'Aperiodic & Multi-swing Cases' and 'Cannot Run OMIB Algorithm', respectively. Since for the 'TP margin', 'TN margin' and 'Marginal stability' have been presented in the Case 1, 7 and 19 in section 5.2, respectively, so in this section, it will mainly focus on the discussion of the rest two situations that cannot return a valid transient stability analysis report.

Aperiodic & Multi-swing Cases

Aperiodic & Multi-swing cases means those who have correct Clustering results but their transient stability situation cannot be analyzed correctly. The reason behind these cases can be explained by what has been discussed in the last paragraph in section 2.6.4, that the OMIB cannot cope with the multi-swing cases due to its limitation. This part can be described by taking the first one of the Multi-swing cases, the Case 20 as an example. So plot its coupling-strength coefficient difference matrix, adjacency matrix and stability analysis report at first:

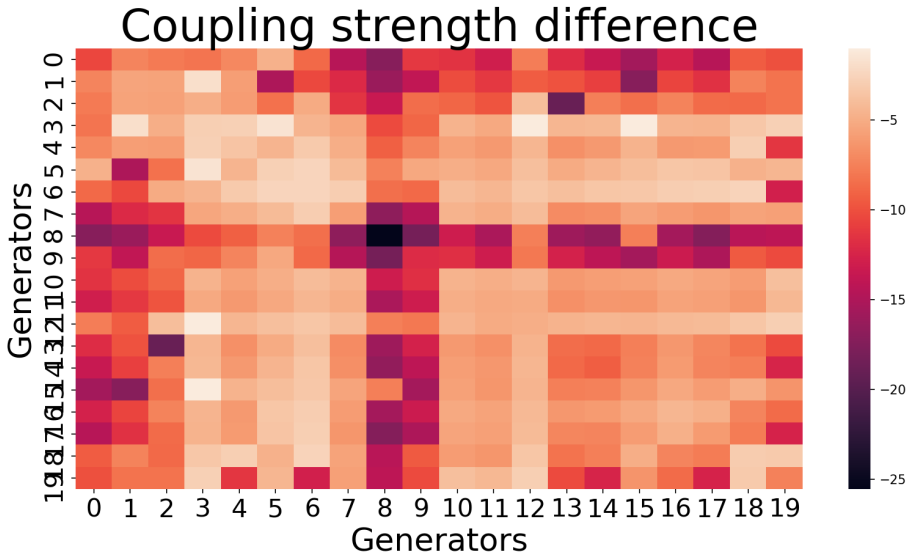


Figure 6.4: Coupling-strength difference for Case 20

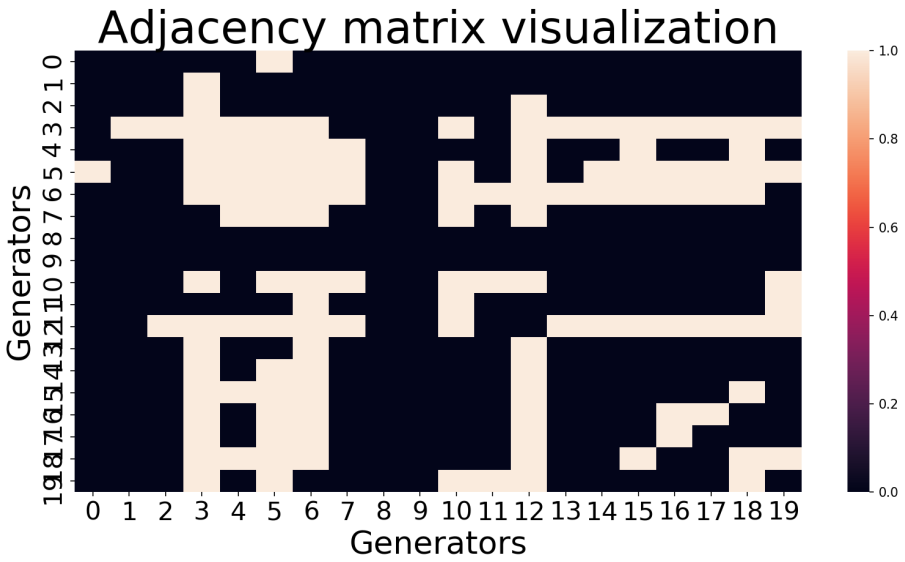


Figure 6.5: DFS results for Case 20

```
####################
Branch tripped case 20
####################
##################################
#%%%%%%% DFS Method Report %%%%%%%#
##################################
Using the Numerical method
Critical machine identification:
[1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0.]

#########################################################################
#%%%%%%%%%%%%%%%%%%%%%%%$$$ Assessment Report $$$%%%%%%%%%%%%%%%%%%%%%%%#
#########################################################################
A normal crossing case
Cross angle =  0.115188174497
Critical unstable angle 3.02640447909
Returning angle, delta_r_OMIB =  0.449823055008656
Total iteration steps =  4
Aacc < Adec, their is a returning angle = 0.450
#%%%%%%%%%%%%%%%%%%%%%%%%#
This is a stable case :)
#%%%%%%%%%%%%%%%%%%%%%%%%#
Margin_stable =  60.2888480522483
Running time for assessment: 2.16773288889 sec
```

Figure 6.6: Transient stability analysis report for Case 20

According to the results showing in the Figure.6.5 and 6.6, the machine 1, 2, 3, 9, 10, 12, 14 and 15 have been detected as the Critical Machines and the OMIB judges this case as a stable case. To validate that, the full-time domain simulation

will be necessary, and it will be plotted into different time scales to have a better observation:

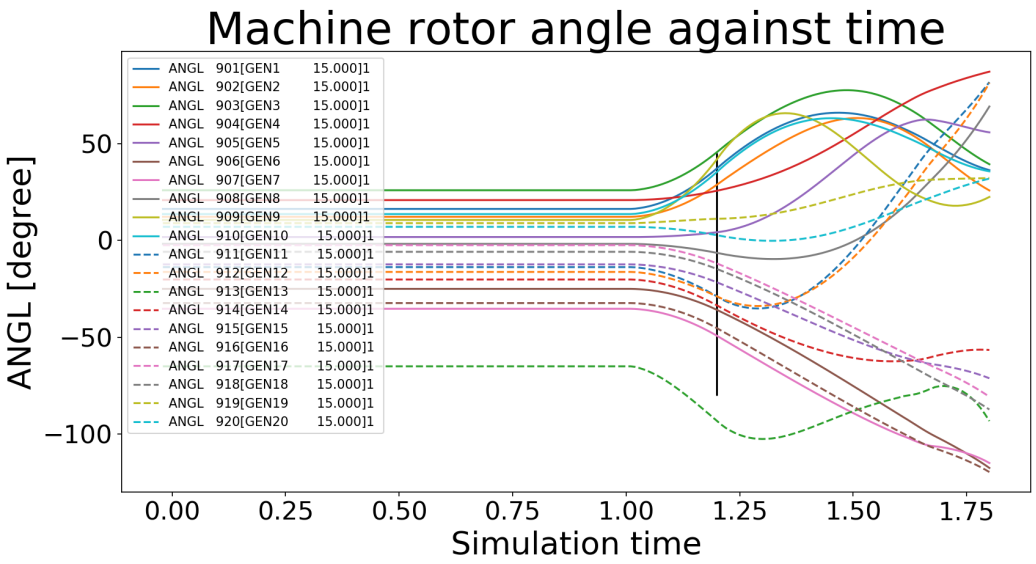


Figure 6.7: Part of full-time domain simulation for Case 20

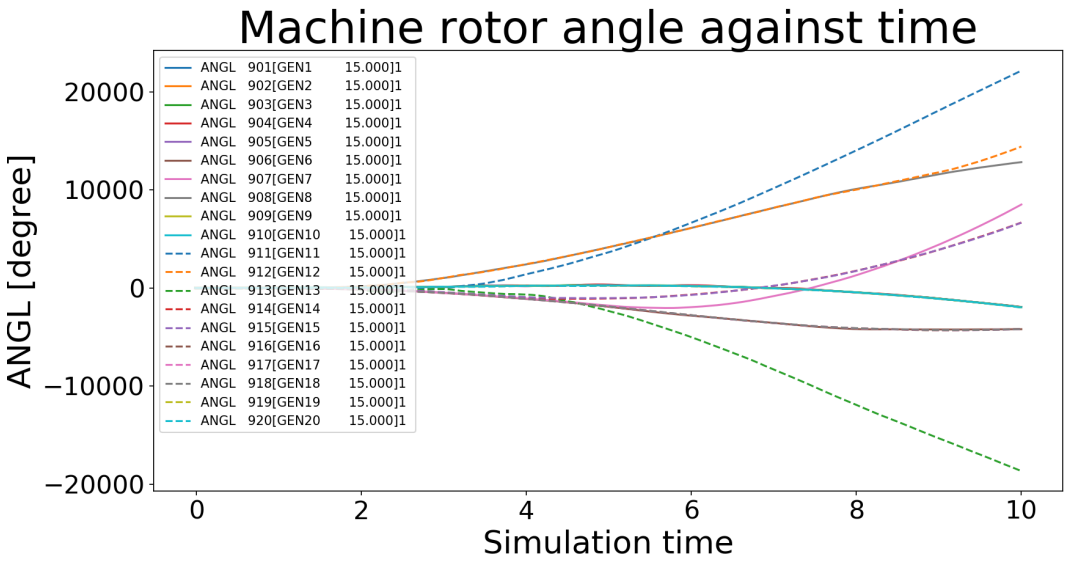


Figure 6.8: Full-time domain simulation for Case 20

As what can be observed from Figure.6.7, those who have an unusual increase (machine 1, 2, 3, 9, 10) have been detected by the Clustering method, and the entire system survives in the first-swing fluctuation. However, when enlarging

the simulating time scale, it can be observed in Figure.6.8 that parts of the system diverge and lose their synchronism forever. This result reveals that if the entire system is under a multi-swing problem, then the OMIB algorithm will probably be invalid and cannot return a correct transient stability analysis report even when the Clustering result is correct, . The same situation also happens in Case 22, 25, 29, 31, 32 and 49, which are both 'suffering' from the multi-swing problem.

Cannot Run OMIB Algorithm

In this case the machines can be clustered correctly, however, when aggregating them into the 'CMs & NMs' two machine equivalents and then the OME for the entire system in further, there might be an error in the mathematical domain that regards to Equation [2.59]. The reason is that the inside of the 'arcsin' function is larger than 1 due to some unknown reasons. This problem also reveals the limitation of the OMIB method that sometimes it cannot directly take the Clustering results and use them for the analysis even when the clustering method works well. To cope with these cases, a much more powerful 'OMIB' method will need to be proposed in the future to adapt all kinds of Clustering results getting from the upstream algorithm.

6.2.2 When the Clustering is Incorrect

When the Clustering method does not work perfectly to detect the correct machines, then there is no need to keep going on the further OMIB since its fundamental is not correct. Therefore, this section will pay attention to those cases who cannot have a correct Clustering result and try to find the reasons behind them. So far, basing on the test results from previous, the reasons of the failure can be divided into two parts, that the first one is the DFS problem and the second one is the problem regarding to the topology.

DFS problems

As the statement in section 2.5.2, the DFS is the technique that simulates the eye recognition of humanity by analyzing the graph theory. However, due to limitation of the three DFS strategies proposed in section 2.5.2, that some 'obvious' graphs of the 'ΔH' matrix cannot be identified. For example, in Case 11, show its PSS/E short-time simulation st first:

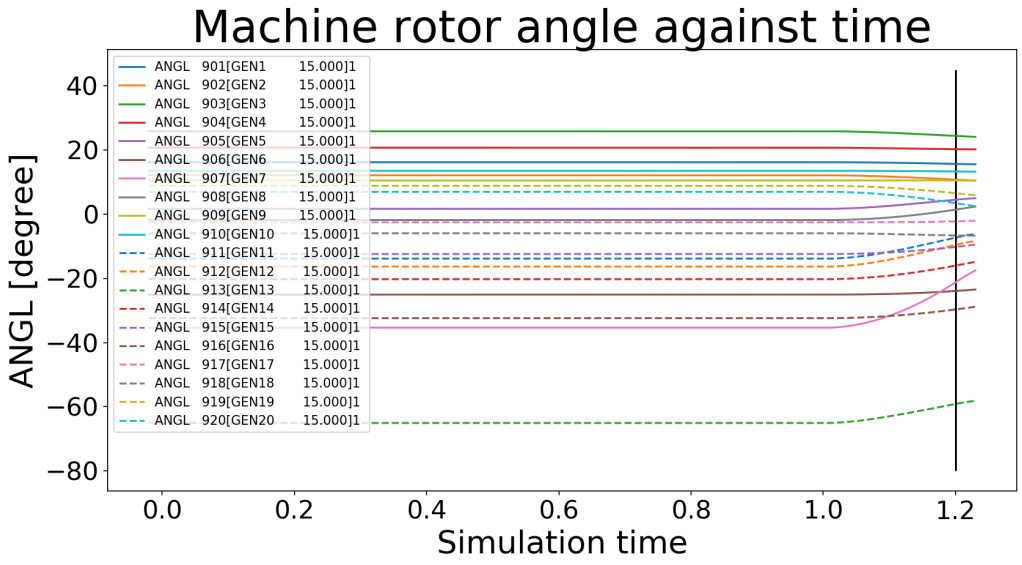


Figure 6.9: Part of full-time domain simulation for Case 11

As what shows in the Figure.6.9, that the machine 7 (Python index 6) is the one who has an unusual increase compared with all others. Then the next step is

to plot its ΔH matrix, adjacency matrix visualization and the Clustering report to see if the coupling-strength calculation works:

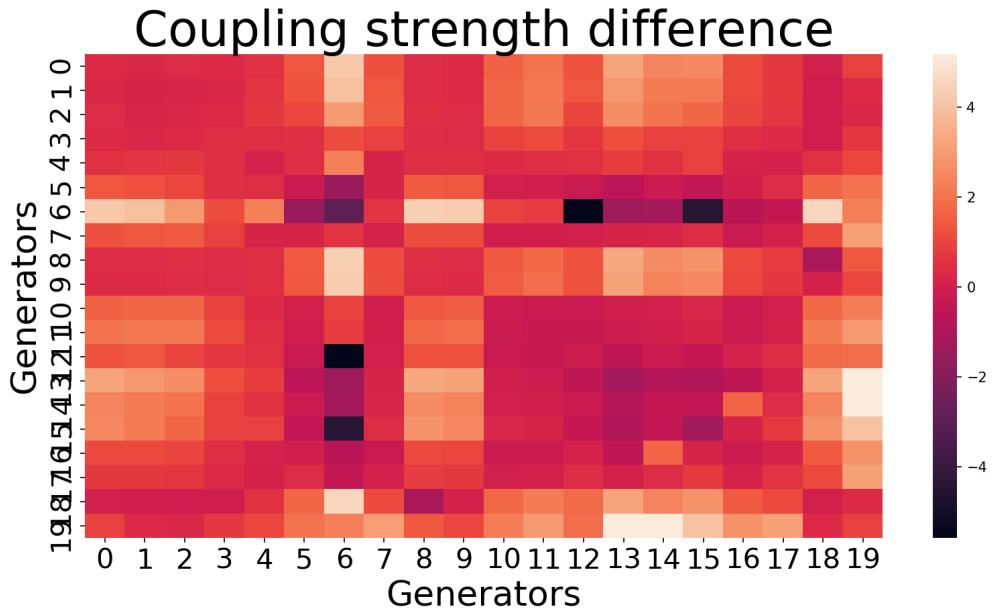


Figure 6.10: ΔH matrix for Case 11

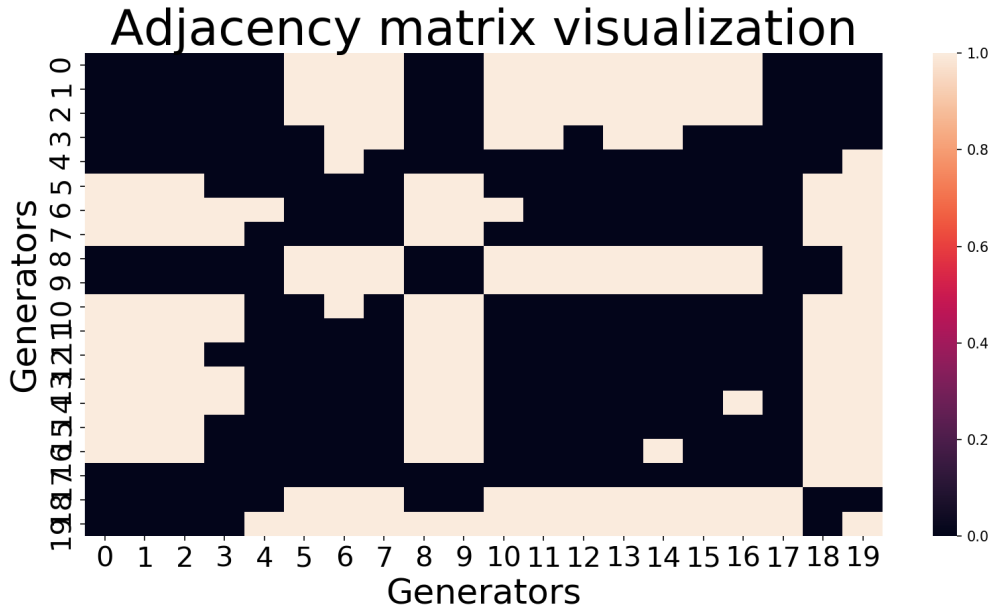


Figure 6.11: DFS results for Case 11

```
######################
Branch tripped case 11
######################
#####################################
#%%%%%%% DFS Method Report %%%%%%%#
#####################################
Using the Clock Method
Critical machine identification:
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

Figure 6.12: Clustering report for Case 11

So according to the Figure.6.10 above, the machine 7 (Python index 6) can be somehow detected as an obvious 'track', which can be observed in the ΔH matrix's column that regards to Python index 6. However, when the system went through the DFS algorithm and published the result in Figure.6.11 and 6.12, the machine 5 and 18 are the ones detected as the Critical ones. This can be attributed to the immature technique of DFS, that ranks and removes elements from Δh matrix mainly basing on the scale of the coupling-strength coefficient. On the other hand, Figure.6.10 also reveals a probability that if one machine has extremely weak relations with a group of machines, but at the same time also has very strong relations with others, then this extreme contrast might make it a Critical Machine. Further, from the perspective of mathematics, this also can be interpreted by using the standard deviation. So check the standard deviation of each column of the ΔH matrix at first:

```
In [10]: np.std(Delta_H, axis = 0)
Out[10]:
array([1.08994517, 1.03669228, 0.8474835 , 0.33994895, 0.46675782,
       0.87468373, 2.90326462, 0.76296267, 1.2009349 , 1.08452567,
       0.83383398, 1.0577265 , 1.52799806, 1.88796   , 1.52972555,
       1.83375639, 0.83954749, 0.68683234, 1.35312831, 1.46073942])
```

Figure 6.13: Standard deviation of ΔH matrix's column in Case 11

From the Figure.6.13, it can be observed that the 7_{th} column, which corresponds to the machine 7, has the largest standard deviation (2.90326462) among the others. This result also reveals another option to cluster the machines that the application

of 'standard deviation' can quantify the contradictory situation described in the red sentences above, and cluster the machines from an another perspective rather than the three existing strategies proposed in section 2.5.2. However, due to the rarity of this kind of cases, this method is not proposed as the fourth DFS strategy because of its limitation.

Besides, except the Case 11, Case 13 and 16 also have the same problem, which is the reason why they have been put into the same cluster with Case 11 in Figure.6.3.

Topology Problem

Except those who have a DFS problem in the clustering, the following cases behave a little bit 'stranger' since basing on the machine clustering results, it seems like that the Critical Machines 'have nothing to do' with their coupling-strength coefficients. For example, for the first case of the term 'Topology Problem' in Figure.6.3, plot its PSS/E short-time simulation, Clustering report, ΔH matrix visualization and the adjacency matrix at first:

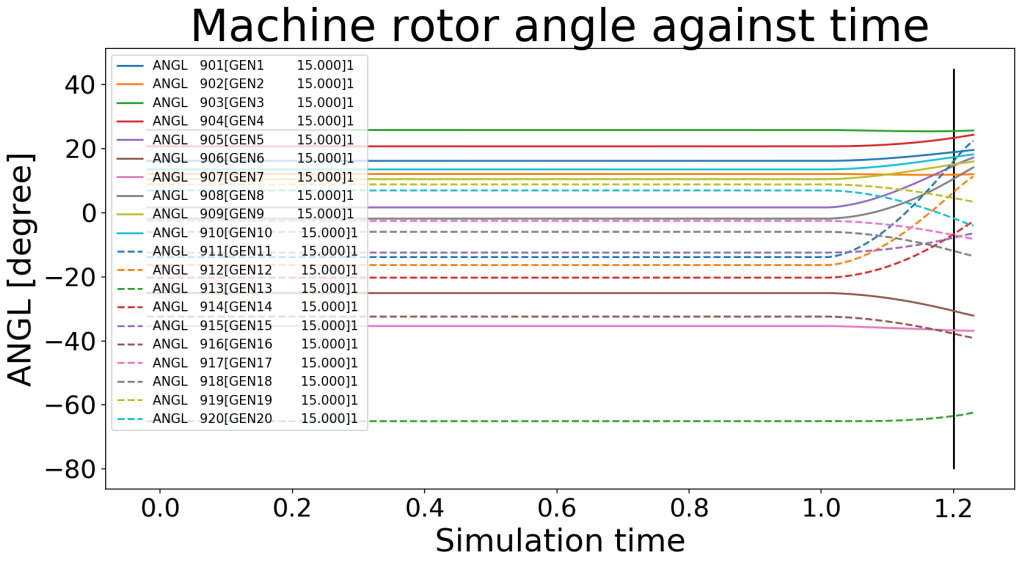


Figure 6.14: Part of full-time domain simulation for Case 37

```
########################
Branch tripped case 37
########################
#####################################
#%%%%%%% DFS Method Report %%%%%%%#
#####################################
Using the Extended numerical method
Critical machine identification:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

Figure 6.15: Clustering report for Case 37

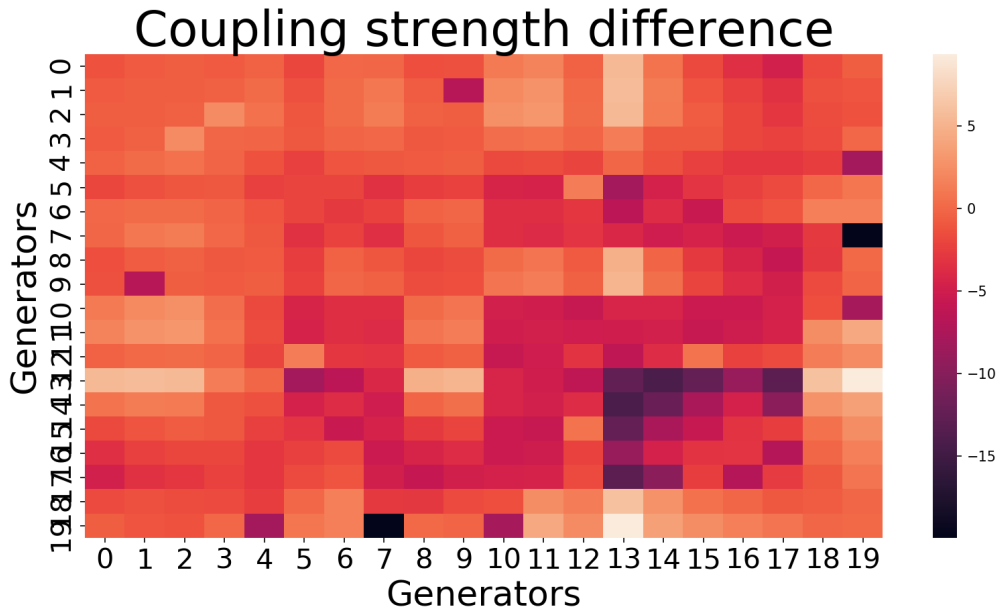


Figure 6.16: Coupling-strength difference for Case 37

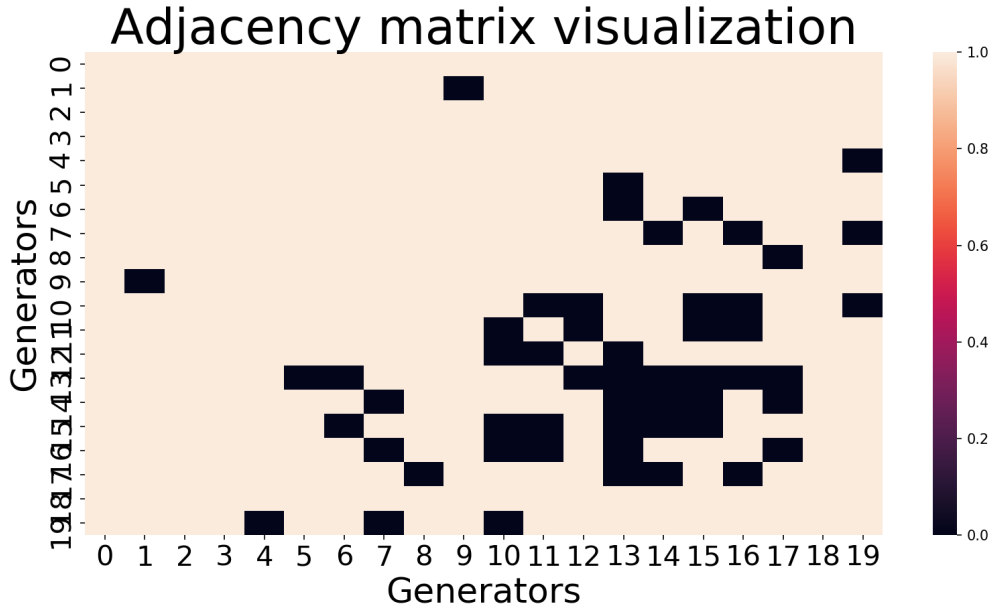


Figure 6.17: Adjacency matrix for Case 37

As what can be observed from Figure.6.14, it is obviously that machine 11, 12 and 14 shall be the Critical Machines, however, the Clustering method can only detect the last one at this time. Then by checking Figure.6.16 and 6.17, it can be observed that there are not too much apparent deep color occurs in the columns that correspond to machine 11 and 12, which makes them cannot be detected

by the DFS algorithm. So for cases like Case 37, the weakened relation between machines is not the main problem that leads to asynchronism. To explain this phenomenon, a deeper analysis basing on the topology comparison between the Case 37 and a successful clustering case, the Case 1, is going to be implemented as followed. Since the Case 37 is the branch between Bus 4042 and Bus 4043, and the Case 1 is the branch between Bus 1011 and 1013, so plot Case 37's topology at first:

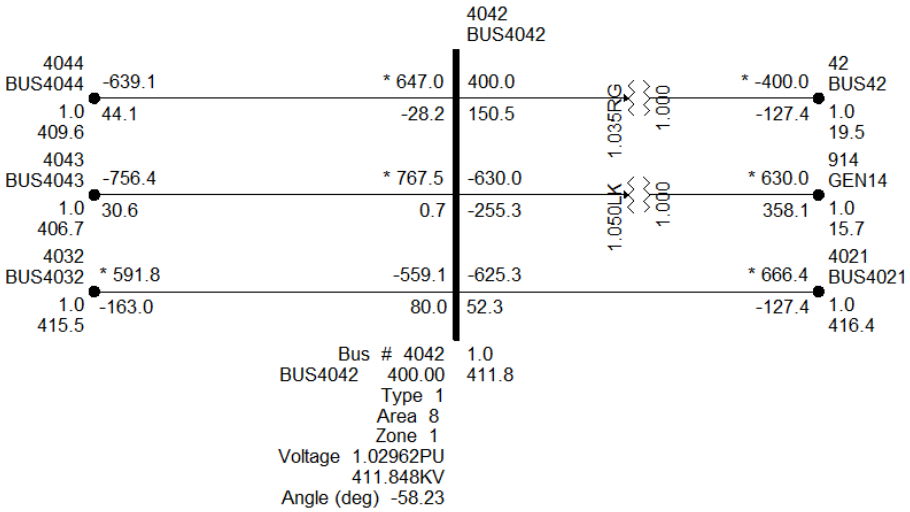


Figure 6.18: Bus 4042 topology

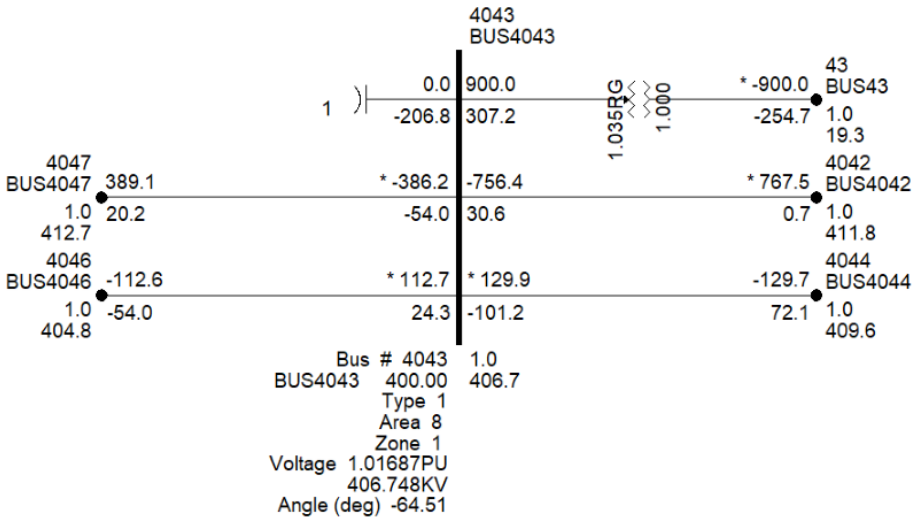


Figure 6.19: Bus 4043 topology

As what can be observed from Figure.6.18 and 6.19, both the Bus 4042 and Bus

4043 are surrounding by several buses and loads, and Bus 4042 also connects to the Generator 14. Furthermore, the power exchange between the Bus 4042 and Bus 4043 is the largest power flow for Bus 4042, and if this branch breaks, then there is no way for this power compensation.This makes the Bus 4042 and Bus 4043 can do nothing but drawing more active power from the generator or releasing more active power to the other buses that closed to Generator 11 and 12, which results in a severe power imbalance due to the unwise topology design.

In contrast, for the successful clustering cases, the Case 1, plot its topology between two sides of this case:

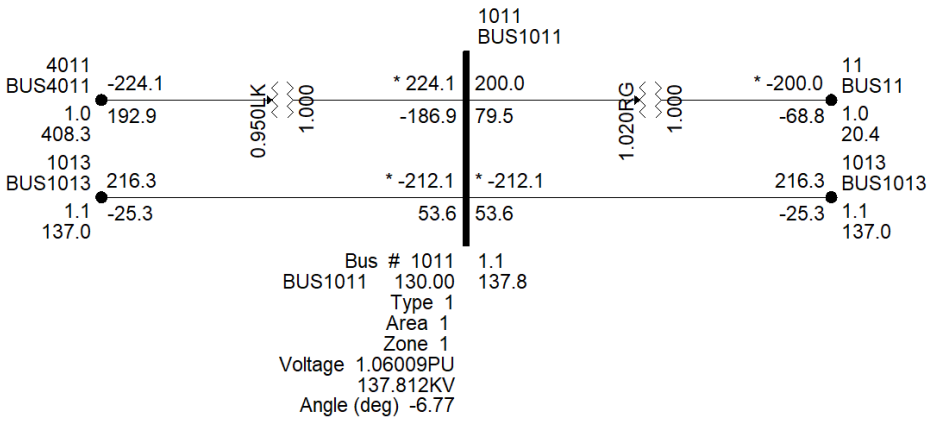


Figure 6.20: Bus 1011 topology

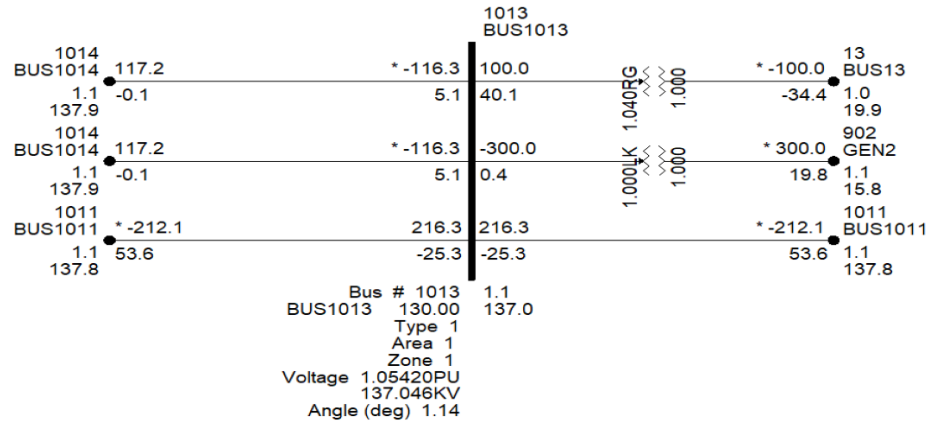


Figure 6.21: Bus 1013 topology

As what shows in the Figure.6.20 and 6.21, even though in Case 1 that one branch between Bus 1011 and Bus 1013 has been tripped, however, as the original system

has a two parallel lines between Bus 1011 and Bus 1013, so losing one line will not result in a severe power imbalance like what happened in Case 37. So basing on the comparison between Case 1 and Case 37, an inference can be proposed as the new limitation of this Coupling-Strength Clustering method, that when the system has huge power imbalance inside, then the Coupling-Strength Clustering method might malfunction due to the influence of the machine relations has been reduced by the dramatic power imbalance.

6.3 Chapter Summary

In this chapter the code automation has been achieved, and basing on the automatically running results the analyses have also been implemented. For those cases who can provide a correct Clustering results, their downstream results from the OMIB algorithm have been classified and discussed. While, for those cases who cannot provide a correct Clustering result, this paper has tried to find out and discuss the reasons behind them, and at the same time, propose a new limitation of the Coupling-Strength Clustering method applying.

CHAPTER 7

Discussion and Conclusion

7.1 Project Discussion

Basing on all the theories interpreted above, and the results' visualization from the implementation, the 'Coupling-Strength Coefficient' algorithm is able to accomplish the Machine Clustering with a relatively high accuracy. Although, there is still 'dead space' in this method that might make it malfunction, however, the research direction of the 'Coupling-Strength' algorithm is not a mistake and can be improved further to make it more generalized. Furthermore, the OMIB method also proves its reliability in the first-swing transient stability analysis. Besides, the convenience of the automation's implementation showing in the Chapter 6 also reveals its feasibility of industrialization.

7.2 Future Perspectives

If this project keep going on in the future, it can be improved in three aspects as followed.

7.2.1 The Evolution of the Clustering Method

As what has been discussed in section 6.2.2, there are still some deficiencies in the proposed Clustering method. So in the future, research direction can focus on extending the limitation of the Clustering method and make it much more generalized. For example, by having the consideration of combining the Coupling-Strength Coefficient with the huge power imbalance inside the system can be a new way to improve the entire algorithm.

7.2.2 Running Time Reduction

Basing on a Intel i5 core and 12 GB RAM, the running time for the Clustering & Transient Stability Analysis for one case is between 2 [sec] to 2.6 [sec]. This scale of time can be suitable for the offline preventive analysis, but on the other hand, not fast enough for the online testing. So in the future, the optimization of the program or the updating of the hard device can both be the direction to make it run faster and be more capable.

7.2.3 Interconnection with PMUs

Due to the limit of time, this project does not include the interconnection with the PMUs. Although for the same system, the upstream data from the PSS/E simulation or the snapshots from PMUs shall has the same influence in the final results. However, the interconnection with PMUs actually indicates the interconnection with the hard devices in the real world, which can be treated as the first step to industrialization.

7.3 Project Conclusion

As the final conclusion of the entire paper, list all the learning objectives at first:

(i) Understand and explain the instability mechanism regarding to first-swing transient stability in multi-machines system

(ii) Basing on the data coming from tiny time-domain simulation in PSS/E or the PMUs snapshots for the selected multi-machines system, develop the Python scripts for the 'Fast Contingencies Assessment' by implementing state of the art Machine Clustering method. Then double check the algorithm and the Clustering results with the PSS/E simulation to make sure it works well

(iii) Afterwards basing on the Clustering results from above, implement One Machine Infinite Bus (OMIB) method to evaluate the stability margin for selected multi-machines system

(iv) Finally achieve the automation of the assessment of all possible branches faults (all possible N-1 scenarios) in the system

So far, this paper has completely finished the objectives proposed in the beginning of this project. The theories discussed in both the physical and mathematical level have been achieved in Chapter 2, and the implementations of them have also been accomplished in the Chapter 3, 4 and 5, respectively. The source code of the entire Simulating & Clustering & Assessment system has been built in the Python script from the scratch, and the entire system is totally executable and reliable. From the beginning, by using the 'Generalized Laplacian Algorithm', the admittance matrix in pre-fault and post-fault can be generated automatically, and their results have been validated with the admittance matrix generated by PSS/E. After that, the result of the Clustering method, which is the crucial part of this project, has been presented, interpreted and also validated in the section 4.7 in three different cases. And basing on these Clustering results, the implementation of the OMIB algorithm also works well and provide a valid first-swing transient stability analysis report. Besides, this paper also developed a 'smarter' code to run every one fault in each branch automatically. And basing on the automation results, find the 'singularities' of the Clustering results and provide reasonable explanations of them.

Bibliography

[1] Stephen P.Leatherman. Pier Vellinga. Sea level rise, consequences and policies. *Climatic Change*, 15:175 - 189, October 1989.

[2] C. Fu and A. Bose. Contingency ranking based on severity indices in dynamic security analysis. *IEEE Trans. Power Syst*, 14(13):980 - 986, August 1999.

[3] Vittal V. Ejebe G. C. Irisarri G. D. Tong J. Pieper G. Chadalavada, V. and M. McMullen. An on-line contingency filtering scheme for dynamic security assessment. *IEEE Trans. Power Syst*, 12(1):153 - 161, February 1997.

[4] Cai N. Benidris, M. and J. Mitra. A fast transient stability screening and ranking tool. *Proceedings of the Power Systems Computation Conference (PSCCâ14)*, August 2014.

[5] Huang T. Xue, Y. and F. Xue. Effective and robust case screening for transient stability assessment. *IREP Symposium Bulk Power System Dynamics and Control - IX Optimization, Security and Control of the Emerging Power Grid*, pages 1 - 8, 2013.

[6] Mostafa H. E. Khatib A. R. Helal I. Hashiesh, F. and M. M. Mansour. An intelligent wide area synchrophasor based system for predicting and mitigating transient instabilities. *IEEE Trans. Smart Grid*, 3(2):645 - 652, 2012.

[7] Mevludin Glavic Tilman Weckesser, Hj0rtur Johannsson and Jacob Ãstergaard. An improved on-line contingency screening for power system transient stability assessment. *Electric Power Components and Systems*, 45(8):852 - 863.

[8] Hjortur; østergaard Jacob. Weckesser, Johannes Tilman Gabriel; Johannsson. Critical machine clustering identification using the equal area criterion. *IEEE Power and Energy Society General Meeting 2015*, 2015.

[9] Ernst D. Pavella, M. and D. Ruiz-Vega. Transient stability of power systems: A unified approach to assessment and control. *Kluwer Academic Publishers*.

[10] Kwon S.-H. Lee J. Nam H.-K. Choo J.-B. Lee, B. and D.-H. Jeon. Fast contingency screening for online transient stability monitoring and assessment of the kepco system. *IEEE Proc., Gener. Transm. Distrib.*, 150(4):399 - 404, July 2003.

Appendix

Branch Information

```
################################
###### Branch information ######
################################

def Branch_information():
    import numpy as np
    global B, direction

    #Create the Bus name dictionary basing on the raw file
    B = {1:0, 2:1, 3:2, 4:3, 5:4, 11:5, 12:6, 13:7, 22:8, 31:9,
       32:10, 41:11, 42:12, \
         43:13, 46:14, 47:15, 51:16, 61:17, 62:18, 63:19, 71:20,
             72:21, 901:22, 902:23, \
         903:24, 904:25, 905:26, 906:27, 907:28, 908:29, 909:30,
             910:31, 911:32, 912:33, \
         913:34, 914:35, 915:36, 916:37, 917:38, 918:39, 919:40,
             920:41, 1011:42, 1012:43, \
         1013:44, 1014:45, 1021:46, 1022:47, 1041:48, 1042:49,
            1043:50, 1044:51, 1045:52, 2031:53, \
         2032:54, 4011:55, 4012:56, 4021:57, 4022:58, 4031:59,
            4032:60, 4041:61, 4042:62, 4043:63, \
         4044:64, 4045:65, 4046:66, 4047:67, 4051:68, 4061:69,
            4062:70, 4063:71, 4071:72, 4072:73}

    #Branch & transformer direction
    #Branch direction firstly
    direction = np.zeros([52+50,2], dtype = int)
    direction[0,0] = B[1011]; direction[0,1] = B[1013]
    direction[1,0] = B[1011]; direction[1,1] = B[1013]
    direction[2,0] = B[1012]; direction[2,1] = B[1014]
    direction[3,0] = B[1012]; direction[3,1] = B[1014]
```

```
25      direction[4,0] = B[1013]; direction[4,1] = B[1014]
        direction[5,0] = B[1013]; direction[5,1] = B[1014]
27      direction[6,0] = B[1021]; direction[6,1] = B[1022]
        direction[7,0] = B[1021]; direction[7,1] = B[1022]
29      direction[8,0] = B[1041]; direction[8,1] = B[1043]
        direction[9,0] = B[1041]; direction[9,1] = B[1043]
31      direction[10,0] = B[1041]; direction[10,1] = B[1045]
        direction[11,0] = B[1041]; direction[11,1] = B[1045]
33      direction[12,0] = B[1042]; direction[12,1] = B[1044]
        direction[13,0] = B[1042]; direction[13,1] = B[1044]
35      direction[14,0] = B[1042]; direction[14,1] = B[1045]
        direction[15,0] = B[1043]; direction[15,1] = B[1044]
37      direction[16,0] = B[1043]; direction[16,1] = B[1044]
        direction[17,0] = B[2031]; direction[17,1] = B[2032]
39      direction[18,0] = B[2031]; direction[18,1] = B[2032]
        direction[19,0] = B[4011]; direction[19,1] = B[4012]
41      direction[20,0] = B[4011]; direction[20,1] = B[4021]
        direction[21,0] = B[4011]; direction[21,1] = B[4022]
43      direction[22,0] = B[4011]; direction[22,1] = B[4071]
        direction[23,0] = B[4012]; direction[23,1] = B[4022]
45      direction[24,0] = B[4012]; direction[24,1] = B[4071]
        direction[25,0] = B[4021]; direction[25,1] = B[4032]
47      direction[26,0] = B[4021]; direction[26,1] = B[4042]
        direction[27,0] = B[4022]; direction[27,1] = B[4031]
49      direction[28,0] = B[4022]; direction[28,1] = B[4031]
        direction[29,0] = B[4031]; direction[29,1] = B[4032]
51      direction[30,0] = B[4031]; direction[30,1] = B[4041]
        direction[31,0] = B[4031]; direction[31,1] = B[4041]
53      direction[32,0] = B[4032]; direction[32,1] = B[4042]
        direction[33,0] = B[4032]; direction[33,1] = B[4044]
55      direction[34,0] = B[4041]; direction[34,1] = B[4044]
        direction[35,0] = B[4041]; direction[35,1] = B[4061]
57      direction[36,0] = B[4042]; direction[36,1] = B[4043]
        direction[37,0] = B[4042]; direction[37,1] = B[4044]
59      direction[38,0] = B[4043]; direction[38,1] = B[4044]
        direction[39,0] = B[4043]; direction[39,1] = B[4046]
61      direction[40,0] = B[4043]; direction[40,1] = B[4047]
        direction[41,0] = B[4044]; direction[41,1] = B[4045]
63      direction[42,0] = B[4044]; direction[42,1] = B[4045]
        direction[43,0] = B[4045]; direction[43,1] = B[4051]
```

```
65      direction[44,0] = B[4045]; direction[44,1] = B[4051]
        direction[45,0] = B[4045]; direction[45,1] = B[4062]
67      direction[46,0] = B[4046]; direction[46,1] = B[4047]
        direction[47,0] = B[4061]; direction[47,1] = B[4062]
69      direction[48,0] = B[4062]; direction[48,1] = B[4063]
        direction[49,0] = B[4062]; direction[49,1] = B[4063]
71      direction[50,0] = B[4071]; direction[50,1] = B[4072]
        direction[51,0] = B[4071]; direction[51,1] = B[4072]
73
        #Transformer direction secondly
75      direction[52,0] = B[1]; direction[52,1] = B[1041]
        direction[53,0] = B[2]; direction[53,1] = B[1042]
77      direction[54,0] = B[3]; direction[54,1] = B[1043]
        direction[55,0] = B[4]; direction[55,1] = B[1044]
79      direction[56,0] = B[5]; direction[56,1] = B[1045]
        direction[57,0] = B[11]; direction[57,1] = B[1011]
81      direction[58,0] = B[12]; direction[58,1] = B[1012]
        direction[59,0] = B[13]; direction[59,1] = B[1013]
83      direction[60,0] = B[22]; direction[60,1] = B[1022]
        direction[61,0] = B[31]; direction[61,1] = B[2031]
85      direction[62,0] = B[32]; direction[62,1] = B[2032]
        direction[63,0] = B[41]; direction[63,1] = B[4041]
87      direction[64,0] = B[42]; direction[64,1] = B[4042]
        direction[65,0] = B[43]; direction[65,1] = B[4043]
89      direction[66,0] = B[46]; direction[66,1] = B[4046]
        direction[67,0] = B[47]; direction[67,1] = B[4047]
91      direction[68,0] = B[51]; direction[68,1] = B[4051]
        direction[69,0] = B[61]; direction[69,1] = B[4061]
93      direction[70,0] = B[62]; direction[70,1] = B[4062]
        direction[71,0] = B[63]; direction[71,1] = B[4063]
95      direction[72,0] = B[71]; direction[72,1] = B[4071]
        direction[73,0] = B[72]; direction[73,1] = B[4072]
97      direction[74,0] = B[901]; direction[74,1] = B[1012]
        direction[75,0] = B[902]; direction[75,1] = B[1013]
99      direction[76,0] = B[903]; direction[76,1] = B[1014]
        direction[77,0] = B[904]; direction[77,1] = B[1021]
101     direction[78,0] = B[905]; direction[78,1] = B[1022]
        direction[79,0] = B[906]; direction[79,1] = B[1042]
103     direction[80,0] = B[907]; direction[80,1] = B[1043]
        direction[81,0] = B[908]; direction[81,1] = B[2032]
```

```
105     direction[82,0] = B[909]; direction[82,1] = B[4011]
        direction[83,0] = B[910]; direction[83,1] = B[4012]
107     direction[84,0] = B[911]; direction[84,1] = B[4021]
        direction[85,0] = B[912]; direction[85,1] = B[4031]
109     direction[86,0] = B[913]; direction[86,1] = B[4041]
        direction[87,0] = B[914]; direction[87,1] = B[4042]
111     direction[88,0] = B[915]; direction[88,1] = B[4047]
        direction[89,0] = B[916]; direction[89,1] = B[4051]
113     direction[90,0] = B[917]; direction[90,1] = B[4062]
        direction[91,0] = B[918]; direction[91,1] = B[4063]
115     direction[92,0] = B[919]; direction[92,1] = B[4071]
        direction[93,0] = B[920]; direction[93,1] = B[4072]
117     direction[94,0] = B[1011]; direction[94,1] = B[4011]
        direction[95,0] = B[1012]; direction[95,1] = B[4012]
119     direction[96,0] = B[1022]; direction[96,1] = B[4022]
        direction[97,0] = B[1044]; direction[97,1] = B[4044]
121     direction[98,0] = B[1044]; direction[98,1] = B[4044]
        direction[99,0] = B[1045]; direction[99,1] = B[4045]
123     direction[100,0] = B[1045]; direction[100,1] = B[4045]
        direction[101,0] = B[2031]; direction[101,1] = B[4031]
125

127     return B, direction
```

PSS/E Simulation

```
1 ##################################
  ###### PSS/E simulation file ######
3 ##################################
  def PSSE_simulation(t_clear, t_final, B, direction, trip_index):
5     #Example script for running PSSE from python
      #, which is developed for PSSE33.
7
      # Import modules from python standard library
9     import read_psse_outfile as rpo
      import matplotlib.pyplot as plt
11    import os, sys
      # Verify that your python version is 2.7.xx at 32bit
13    print(sys.version)


15
      # Locate the psspy.pyc file among your PSSE installation
          files.
17    # Add the path of the file to your python search path and
          the windows PATH variable
      sys.path.append(r"C:\Program␣Files␣(x86)\PTI\PSSE33\PSSBIN")
19    os.environ['PATH'] = (r"C:\Program␣Files␣(x86)\PTI\PSSE33\
          PSSBIN;" + os.environ['PATH'])

21    # Import the python module interface to PSSE
      import psspy
23    # Redirect will alow you to pipe messages from the solver to
          e.g. a log file.
      # This will not be used in this example
25    import redirect
      redirect.psse2py()
27



29
      # Define integer and float formats used by psse.
31    _i = psspy._i
      _f = psspy._f
33
      # Initialize PSSE with enough memory allocated for a number
          of nodes (2000 by example)
```

```
35  psspy.psseinit(2000)

37

39  # Inspect the file 'nordic_system.raw' which contains the
        power flow data of your model.
    # Now read load flow data into python
41  psspy.read(0,r"""nordic_system.raw""")

43

45  # Now we set solution parameters for the power flow
        soulution.
    # _i and _f represent the default integer values for the
        respective parameters.
47  # Find solutions_parameters_4 in the PSSE API (in DOCS) and
        check that the parameters
    # below should set the parameter for a Newton-Raphson Power
        flow at an error of at most
49  # 0.00001 times the system power base.
    #*****************************************[[Find in book
        API, page_766.]]
51  psspy.solution_parameters_4([_i,20,_i,_i,10],[_f,_f,_f,_f,_f
        , 0.1,_f,_f,_f,_f,\
                                    _f,_f,_f,_f,_f,_f,_f,_f,_f])
53

55

    # Now we are ready to run the power flow solution.
57  # You can refer to the Program Operations Manual and the
        PSSE API for information of
    #the function call
59  #****************************************** [[Find in book
         API, page_214.]]
    psspy.fnsl([0,0,0,1,1,1,-1,0])
61  #############
    ###-Notes-###
63  #############
    # The function here is the trigger of different adjustment
        flag.
```

```
65


67

    ########################################
69  # Run power flow and inspect results
    # Try and change power injection of some of
71  # the generators in the raw file and
    # see power injection at swing bus changes
73  # when you re-run the script
    ########################################
75


77

    ######################################################
79  # Now we are ready to look into dynamic simulations.
    # Inspect the .dyr file of the example in a text editor.
81  # All the models are documented in DOCS/MODELS.book
    ######################################################
83
    #*******************************************[[Find in book
       API, page_1028.]]
85  # load the dyr file in python
    psspy.dyre_new([1,1,1,1],r"""nordic_system.dyr""","","","")
87  #############
    ###-Notes-###
89  #############
    # Function here is for dynamic model. Including the pre-
       allocating, dynamic data
91  # read, and put the model choosed by the data into the
       working memory.


93


95

    #*******************************************[[Find in book
       API, page_1024.]]
97  # set appropriate parameters for a dynamic simulation
    # In this example we keep everything at default values, but
       set the simulation
99  # time step to 10ms
```

```
    psspy.dynamics_solution_param_2([_i,_i,_i,_i,_i,_i,_i,_i],[
        _f,_f, 0.0100,_f,_f,_f,_f,_f])
101 #############
    ###-Notes-###
103 #############
    # The form here is implemented by "dynamics_solution_param_2
        (intgar, realar)"
105 # The 'intgar' and 'realar' here are both array of 8
        elements
    # 'Intgar' here is associated with variables, available
        location;
107 # 'Realar' here is associated with different factors like
        for acceleration,
    # convergence tolerance, time step. From 6-8 is about island
         frequency simulation.
109 # But here, dont know why using the '_i' and '_f'.

111


113 # Specify a simulation output file name and set-up output
        channels
    # i.e. what data streams should be written to the file.
115 out_file = '""␣'+ 'example' + '.out"'"
    #*****************************************[[Find in book
        API, page_1003.]]
117 psspy.change_channel_out_file(out_file)#############
    ###-Notes-###
119 #############
    # Here the function is to set the file name for the output
        file.
121


123


125 #*****************************[[Find in book API, page_1073
        .]]
    # Rotor angle and power
127 psspy.chsb(0,1,[-1,-1,-1,1,1,0]) # Machine rotor angle
    psspy.chsb(0,1,[-1,-1,-1,1,2,0]) # Machine active power
129 psspy.chsb(0,1,[-1,-1,-1,1,3,0]) # Machine reactive power
```

```
    psspy.chsb(0,1,[-1,-1,-1,1,6,0]) # Machine mechanical power
131 psspy.chsb(0,1,[-1,-1,-1,1,7,0]) # Machine speed deviation
    # Voltage magnitude and angle
133 psspy.chsb(0,1,[-1,-1,-1,1,4,0]) # Machine terminal voltage
    psspy.chsb(0,1,[-1,-1,-1,1,14,0])# Voltage and angle of all
        buses
135

137 #############
    ###-Notes-###
139 #############
    # Function here is to monitor the dynamic simulation in the
        subsystem.
141 # First index here is the 'valid subsystem identifier range
        from 0 to 11'
    # Second index here is the trigger of buses selected.(1 for
        all and 0 for buses
143 # in subsystem)
    # The third one is an array of six elements. From 1-3 is the
         different starting
145 # index; while the 4 is the type of what will be placed into
         the output channels.
    # and depending on what in 4, the 5 can be different(e.g.
        machine angle, P, Q and so on.)
147 # 6 is the out of service equipment option.

149


151
    # Refer all angles to a system-wide average
153 # (Carefull! May not be the right choice for your simulation
        )
    #*****************************[[Find in book API, page_1057
        .]]
155 psspy.set_relang(1,-1,"")
    #############
157 ###-Notes-###
    #############
159 # Function here is to enable of disable the expression
        approach that basing on a
```

```
# designated reference angle.
# The first is the trigger to open relative angle calculation or not.
# The second is the type of relative angle calculation.(-1 means system weighted
# average angle.)
# The third is the machine identifier.



# Dynamic simulations with constant power loads will give nummeric problems.
# Loads are therfore typically converted to the constant currents or constant impedances.
# Below We convert all loads such that active power is represented by constant currents
# and reactive power becomes constant impedances:
#*****************************[[Find in book API, page_176.]]
psspy.conl(0,1,1,[0,0],[ 100.0,0.0,0.0, 100.0])
psspy.conl(0,1,2,[0,0],[ 100.0,0.0,0.0, 100.0])
psspy.conl(0,1,3,[0,0],[ 100.0,0.0,0.0, 100.0])
##############
###-Notes-###
##############
# Function here is for the transformation of the constant MVA load to a mixture of
# constant MVA, constant current and constant admittance load characteristics since
# when using the constant power load for the dynamic simulation the nummeric problems
# probably happens.
# The first one inside is the a valid subsystem identifier, range from 0 to 11, default
# by 0;
# Second here is the all buses or specified buses flag, 1 for all and 0 for subsystem;
# Third here is the mode of operation in API;
# ***
```

```
189     # Fourth here is an array of two elements, the first one
            inside is the conversion type,
        # 0 means convert all constant MVA load; 1 and 2 apply
            standardization firstly and then
191     # convert(This one here used only when the third index above
             is 1 !!!!).And the second
        # one inside the fourth stands for the choice of buses(used
            only when the third index
193     # above is 2 !!!). Both these two here using default: 0;
        # ***
195     # The last one inside this function is an array of 4
            elements, which holds the percent
        # of different
197     # kind of loads transfer to different characteristics (API,
            Page_177)


199


201

        # Convert generators, factorize admittance matrix and solve
            for switching studies
203     #*****************************[[Find in book API, page_175,
             page_205 and
        # page_652, respectively]]
205     #############
        ###-Notes-###
207     #############
        psspy.cong(0) # Function here is to transfer the power flow
            G for switching studies
209     # and dynamic simulations;
        psspy.fact()  # Function here is to factorize the Ybus
            matrix for the same above proposes;
211     psspy.tysl(0) # Function here is to activate the switching
            study network solutions.
        # (0 for default start and
213                   # 1 for flat start).


215


217     ###################
```

```
    # Initialize state-space model
219 #*****************************[[Find in book API, page_1069
        .]]
    psspy.strt(0,out_file)
221 #############
    ###-Notes-###
223 #############
    # Function here is to initialize a PSSE dynamic simulation
        for state-space simulations
225 # The first part is the network convengence monitor option,
        0 for only if
    # CM interrupt activated for the convergence monitor.
227 # And the second part is the Channel Output File


229


231
    # Run simulation until t=1 sec.
233 #*****************************[[Find in book API, page_1046
        .]]
    psspy.run(0, 1.0,0,1,0)
235 #############
    ###-Notes-###
237 #############
    # Function here is for running the PSSE state-space dynamic
        simulations
239 # The first element inside the function is the same with the
         first part in '.strt'
    # The second element insdie is the value of the whole
        simulation time
241 # The third element inside is the number of time steps
        between printing of output
    # channel values
243 # The fourth element inside is the time steps between the
        writing of output channel
    # values to the current channel output file.
245 # ***
    # The last element is the number of time steps between the
        plotting of these output
247 # channel values
```

```
249
    # Branch tripping information
251 trip_branch_from = list(B.keys())[list(B.values()).index(
        direction[trip_index,0])]
    trip_branch_to   = list(B.keys())[list(B.values()).index(
        direction[trip_index,1])]
253

255 # Apply a branch fault on circuit 1 between buses 2031 and
        2032 then run for another 100ms
    #*****************************[[Find in book API, page_1355
        .]]
257 psspy.dist_branch_fault(trip_branch_from, trip_branch_to,r
        """1""",1, 400.0,[0.0,-0.2E+10])
    #############
259 ###-Notes-###
    #############
261 # Function here is to apply a fault between buses.
    # The first and second element is the fault location between
         two buses
263 # The third is the circuit identifier, 1 by default
    # The fourth is the fault admittance of impedence unit, 1
        for MVA, 2 for mhos, and 3 for ohms.
265 # The fifth is the base voltage in kV used for per unit only
         if the fourth is 2 or 3 !!!
    # The last one here is an array of two elements, the first
        and second part of it stand for
267 # the real and imaginary part of the complex fault
        admittance or impedence, respectively.

269 #*****************************[[Find in book API, page_1046
        .]]
    psspy.run(0, t_clear, 0, 1, 0)  # Function here has the same
         comment with python code line_228.
271

273
    # Trip the branch and run until t-final
275 #*****************************************[[Find in book
```

```
    API, page_1357.]]
psspy.dist_branch_trip(trip_branch_from, trip_branch_to,r"""
    1""")
#############
###-Notes-###
#############
# Function here is to work as the fault clearance during the
    dynamic simulation.
# The first and second is still the fault location
# The third is still the circuit identifier

#*****************************[[Find in book API, page_1046
    .]]
psspy.run(0, t_final, 0, 1, 0) # Function here has the same
    comment with python code line_228.



##################
# To inspect the results please see "read_psse_outfile.py"


# Import modules for reading out file and plotting
# There is an alternative to the homemade out-file reader
    called 'dyntools'
# - also to be found on the psse path. But here we will use
    a home-made module.


# specify outout file name and perform the read
file_name = "example.out"
data = rpo.read_psse_outfile(file_name)          # Function
    can read the '.out' file


# Check that the imported data is a dict and print all field
    names of that dict
print(type(data))
```

```
307     print(data.keys())

309
        # Specify a set of data series to plot - here 5 rotor angles
311
        dataseries = ['ANGL    901[GEN1        15.000]1 ',
313                   'ANGL    902[GEN2        15.000]1 ',
                      'ANGL    903[GEN3        15.000]1 ',
315                   'ANGL    904[GEN4        15.000]1 ',
                      'ANGL    905[GEN5        15.000]1 ',
317                   'ANGL    906[GEN6        15.000]1 ',
                      'ANGL    907[GEN7        15.000]1 ',
319                   'ANGL    908[GEN8        15.000]1 ',
                      'ANGL    909[GEN9        15.000]1 ',
321                   'ANGL    910[GEN10       15.000]1 ',
                      'ANGL    911[GEN11       15.000]1 ',
323                   'ANGL    912[GEN12       15.000]1 ',
                      'ANGL    913[GEN13       15.000]1 ',
325                   'ANGL    914[GEN14       15.000]1 ',
                      'ANGL    915[GEN15       15.000]1 ',
327                   'ANGL    916[GEN16       15.000]1 ',
                      'ANGL    917[GEN17       15.000]1 ',
329                   'ANGL    918[GEN18       15.000]1 ',
                      'ANGL    919[GEN19       15.000]1 ',
331                   'ANGL    920[GEN20       15.000]1 ']

333
        # plot the data.
335     t = data['time']

337     #Plot the angle of machines
        plt.figure(dpi = 150)
339     for i in range(len(dataseries)):
            x = data[dataseries[i]]
341         if i <= 9:
                plt.plot(t,x,label=dataseries[i])
343         else:
                plt.plot(t,x,'--',label=dataseries[i])
345     plt.vlines(t_clear, -80, 45)
        plt.xlabel('Simulation time',fontsize = 25)
```

```
347     plt.ylabel('ANGL [degree]',fontsize = 25)
        plt.title('Machine rotor angle against time',fontsize = 35)
349     plt.tick_params(labelsize = 20)
        plt.legend(loc = 'upper left')
351     plt.show()
        print''
353
        return
```

Results Reading

```
1 ######################################################
  ########### Output file from PSS/E Reading ############
3 ######################################################

5 import numpy as np
  import read_psse_outfile as rpo
7
  def results_Reading(nr_G):
9     global delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale,
          t, VLoad
      ## Read the simulation results from the PSS/E
11    file_name = "example.out"
      data = rpo.read_psse_outfile(file_name)       # Function can
         read the '.out' file
13
      ## Split the data to their position.
15    delta_series = ['ANGL    901[GEN1         15.000]1 ',
                      'ANGL    902[GEN2         15.000]1 ',
17                    'ANGL    903[GEN3         15.000]1 ',
                      'ANGL    904[GEN4         15.000]1 ',
19                    'ANGL    905[GEN5         15.000]1 ',
                      'ANGL    906[GEN6         15.000]1 ',
21                    'ANGL    907[GEN7         15.000]1 ',
                      'ANGL    908[GEN8         15.000]1 ',
23                    'ANGL    909[GEN9         15.000]1 ',
                      'ANGL    910[GEN10        15.000]1 ',
25                    'ANGL    911[GEN11        15.000]1 ',
                      'ANGL    912[GEN12        15.000]1 ',
27                    'ANGL    913[GEN13        15.000]1 ',
                      'ANGL    914[GEN14        15.000]1 ',
29                    'ANGL    915[GEN15        15.000]1 ',
                      'ANGL    916[GEN16        15.000]1 ',
31                    'ANGL    917[GEN17        15.000]1 ',
                      'ANGL    918[GEN18        15.000]1 ',
33                    'ANGL    919[GEN19        15.000]1 ',
                      'ANGL    920[GEN20        15.000]1 ']
35


37
```

```
    Pe_series = ['POWR   901[GEN1         15.000]1 ',
                 'POWR   902[GEN2         15.000]1 ',
                 'POWR   903[GEN3         15.000]1 ',
                 'POWR   904[GEN4         15.000]1 ',
                 'POWR   905[GEN5         15.000]1 ',
                 'POWR   906[GEN6         15.000]1 ',
                 'POWR   907[GEN7         15.000]1 ',
                 'POWR   908[GEN8         15.000]1 ',
                 'POWR   909[GEN9         15.000]1 ',
                 'POWR   910[GEN10        15.000]1 ',
                 'POWR   911[GEN11        15.000]1 ',
                 'POWR   912[GEN12        15.000]1 ',
                 'POWR   913[GEN13        15.000]1 ',
                 'POWR   914[GEN14        15.000]1 ',
                 'POWR   915[GEN15        15.000]1 ',
                 'POWR   916[GEN16        15.000]1 ',
                 'POWR   917[GEN17        15.000]1 ',
                 'POWR   918[GEN18        15.000]1 ',
                 'POWR   919[GEN19        15.000]1 ',
                 'POWR   920[GEN20        15.000]1 ']


    Pm_series = ['PMEC   901[GEN1         15.000]1 ',
                 'PMEC   902[GEN2         15.000]1 ',
                 'PMEC   903[GEN3         15.000]1 ',
                 'PMEC   904[GEN4         15.000]1 ',
                 'PMEC   905[GEN5         15.000]1 ',
                 'PMEC   906[GEN6         15.000]1 ',
                 'PMEC   907[GEN7         15.000]1 ',
                 'PMEC   908[GEN8         15.000]1 ',
                 'PMEC   909[GEN9         15.000]1 ',
                 'PMEC   910[GEN10        15.000]1 ',
                 'PMEC   911[GEN11        15.000]1 ',
                 'PMEC   912[GEN12        15.000]1 ',
                 'PMEC   913[GEN13        15.000]1 ',
                 'PMEC   914[GEN14        15.000]1 ',
                 'PMEC   915[GEN15        15.000]1 ',
                 'PMEC   916[GEN16        15.000]1 ',
                 'PMEC   917[GEN17        15.000]1 ',
```

```
                'PMEC␣␣␣918[GEN18␣␣␣␣␣␣␣15.000]1␣',
79              'PMEC␣␣␣919[GEN19␣␣␣␣␣␣␣15.000]1␣',
                'PMEC␣␣␣920[GEN20␣␣␣␣␣␣␣15.000]1␣']
81

83
    omega_series = ['SPD␣␣␣␣901[GEN1␣␣␣␣␣␣␣␣15.000]1␣',
85                  'SPD␣␣␣␣902[GEN2␣␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣903[GEN3␣␣␣␣␣␣␣␣15.000]1␣',
87                  'SPD␣␣␣␣904[GEN4␣␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣905[GEN5␣␣␣␣␣␣␣␣15.000]1␣',
89                  'SPD␣␣␣␣906[GEN6␣␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣907[GEN7␣␣␣␣␣␣␣␣15.000]1␣',
91                  'SPD␣␣␣␣908[GEN8␣␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣909[GEN9␣␣␣␣␣␣␣␣15.000]1␣',
93                  'SPD␣␣␣␣910[GEN10␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣911[GEN11␣␣␣␣␣␣␣15.000]1␣',
95                  'SPD␣␣␣␣912[GEN12␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣913[GEN13␣␣␣␣␣␣␣15.000]1␣',
97                  'SPD␣␣␣␣914[GEN14␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣915[GEN15␣␣␣␣␣␣␣15.000]1␣',
99                  'SPD␣␣␣␣916[GEN16␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣917[GEN17␣␣␣␣␣␣␣15.000]1␣',
101                 'SPD␣␣␣␣918[GEN18␣␣␣␣␣␣␣15.000]1␣',
                    'SPD␣␣␣␣919[GEN19␣␣␣␣␣␣␣15.000]1␣',
103                 'SPD␣␣␣␣920[GEN20␣␣␣␣␣␣␣15.000]1␣']

105
    #Voltage for generator buses
107 V_series = ['ETRM␣␣␣901[GEN1␣␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣902[GEN2␣␣␣␣␣␣␣␣15.000]1␣',
109             'ETRM␣␣␣903[GEN3␣␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣904[GEN4␣␣␣␣␣␣␣␣15.000]1␣',
111             'ETRM␣␣␣905[GEN5␣␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣906[GEN6␣␣␣␣␣␣␣␣15.000]1␣',
113             'ETRM␣␣␣907[GEN7␣␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣908[GEN8␣␣␣␣␣␣␣␣15.000]1␣',
115             'ETRM␣␣␣909[GEN9␣␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣910[GEN10␣␣␣␣␣␣␣15.000]1␣',
117             'ETRM␣␣␣911[GEN11␣␣␣␣␣␣␣15.000]1␣',
```

```
                'ETRM␣␣␣912[GEN12␣␣␣␣␣␣␣15.000]1␣',
119             'ETRM␣␣␣913[GEN13␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣914[GEN14␣␣␣␣␣␣␣15.000]1␣',
121             'ETRM␣␣␣915[GEN15␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣916[GEN16␣␣␣␣␣␣␣15.000]1␣',
123             'ETRM␣␣␣917[GEN17␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣918[GEN18␣␣␣␣␣␣␣15.000]1␣',
125             'ETRM␣␣␣919[GEN19␣␣␣␣␣␣␣15.000]1␣',
                'ETRM␣␣␣920[GEN20␣␣␣␣␣␣␣15.000]1␣']
127
129
    Q_series = ['VARS␣␣␣901[GEN1␣␣␣␣␣␣␣15.000]1␣',
131             'VARS␣␣␣902[GEN2␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣903[GEN3␣␣␣␣␣␣␣15.000]1␣',
133             'VARS␣␣␣904[GEN4␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣905[GEN5␣␣␣␣␣␣␣15.000]1␣',
135             'VARS␣␣␣906[GEN6␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣907[GEN7␣␣␣␣␣␣␣15.000]1␣',
137             'VARS␣␣␣908[GEN8␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣909[GEN9␣␣␣␣␣␣␣15.000]1␣',
139             'VARS␣␣␣910[GEN10␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣911[GEN11␣␣␣␣␣␣␣15.000]1␣',
141             'VARS␣␣␣912[GEN12␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣913[GEN13␣␣␣␣␣␣␣15.000]1␣',
143             'VARS␣␣␣914[GEN14␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣915[GEN15␣␣␣␣␣␣␣15.000]1␣',
145             'VARS␣␣␣916[GEN16␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣917[GEN17␣␣␣␣␣␣␣15.000]1␣',
147             'VARS␣␣␣918[GEN18␣␣␣␣␣␣␣15.000]1␣',
                'VARS␣␣␣919[GEN19␣␣␣␣␣␣␣15.000]1␣',
149             'VARS␣␣␣920[GEN20␣␣␣␣␣␣␣15.000]1␣']
151
153 #Voltage angle for the generator buses
    V_delta_series = ['ANGL␣␣␣␣901␣[GEN1␣␣␣␣␣␣␣15.000]',
155                   'ANGL␣␣␣␣902␣[GEN2␣␣␣␣␣␣␣15.000]',
                      'ANGL␣␣␣␣903␣[GEN3␣␣␣␣␣␣␣15.000]',
157                   'ANGL␣␣␣␣904␣[GEN4␣␣␣␣␣␣␣15.000]',
```

```
                    'ANGL    905 [GEN5         15.000]',
159                 'ANGL    906 [GEN6         15.000]',
                    'ANGL    907 [GEN7         15.000]',
161                 'ANGL    908 [GEN8         15.000]',
                    'ANGL    909 [GEN9         15.000]',
163                 'ANGL    910 [GEN10        15.000]',
                    'ANGL    911 [GEN11        15.000]',
165                 'ANGL    912 [GEN12        15.000]',
                    'ANGL    913 [GEN13        15.000]',
167                 'ANGL    914 [GEN14        15.000]',
                    'ANGL    915 [GEN15        15.000]',
169                 'ANGL    916 [GEN16        15.000]',
                    'ANGL    917 [GEN17        15.000]',
171                 'ANGL    918 [GEN18        15.000]',
                    'ANGL    919 [GEN19        15.000]',
173                 'ANGL    920 [GEN20        15.000]']

175     Vbus_series = ['VOLT      1 [BUS1         20.000]',
                       'VOLT      2 [BUS2         20.000]',
177                    'VOLT      3 [BUS3         20.000]',
                       'VOLT      4 [BUS4         20.000]',
179                    'VOLT      5 [BUS5         20.000]',
                       'VOLT     11 [BUS11        20.000]',
181                    'VOLT     12 [BUS12        20.000]',
                       'VOLT     13 [BUS13        20.000]',
183                    'VOLT     22 [BUS22        20.000]',
                       'VOLT     31 [BUS31        20.000]',
185                    'VOLT     32 [BUS32        20.000]',
                       'VOLT     41 [BUS41        20.000]',
187                    'VOLT     42 [BUS42        20.000]',
                       'VOLT     43 [BUS43        20.000]',
189                    'VOLT     46 [BUS46        20.000]',
                       'VOLT     47 [BUS47        20.000]',
191                    'VOLT     51 [BUS51        20.000]',
                       'VOLT     61 [BUS61        20.000]',
193                    'VOLT     62 [BUS62        20.000]',
                       'VOLT     63 [BUS63        20.000]',
195                    'VOLT     71 [BUS71        20.000]',
                       'VOLT     72 [BUS72        20.000]',
197                    'VOLT    901 [GEN1         15.000]',
```

```
                'VOLT␣␣␣␣902␣[GEN2␣␣␣␣␣␣␣␣15.000]',
199             'VOLT␣␣␣␣903␣[GEN3␣␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣904␣[GEN4␣␣␣␣␣␣␣␣15.000]',
201             'VOLT␣␣␣␣905␣[GEN5␣␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣906␣[GEN6␣␣␣␣␣␣␣␣15.000]',
203             'VOLT␣␣␣␣907␣[GEN7␣␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣908␣[GEN8␣␣␣␣␣␣␣␣15.000]',
205             'VOLT␣␣␣␣909␣[GEN9␣␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣910␣[GEN10␣␣␣␣␣␣␣15.000]',
207             'VOLT␣␣␣␣911␣[GEN11␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣912␣[GEN12␣␣␣␣␣␣␣15.000]',
209             'VOLT␣␣␣␣913␣[GEN13␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣914␣[GEN14␣␣␣␣␣␣␣15.000]',
211             'VOLT␣␣␣␣915␣[GEN15␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣916␣[GEN16␣␣␣␣␣␣␣15.000]',
213             'VOLT␣␣␣␣917␣[GEN17␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣918␣[GEN18␣␣␣␣␣␣␣15.000]',
215             'VOLT␣␣␣␣919␣[GEN19␣␣␣␣␣␣␣15.000]',
                'VOLT␣␣␣␣920␣[GEN20␣␣␣␣␣␣␣15.000]',
217             'VOLT␣␣␣1011␣[BUS1011␣␣␣␣␣130.00]',
                'VOLT␣␣␣1012␣[BUS1012␣␣␣␣␣130.00]',
219             'VOLT␣␣␣1013␣[BUS1013␣␣␣␣␣130.00]',
                'VOLT␣␣␣1014␣[BUS1014␣␣␣␣␣130.00]',
221             'VOLT␣␣␣1021␣[BUS1021␣␣␣␣␣130.00]',
                'VOLT␣␣␣1022␣[BUS102␣2␣␣␣␣130.00]',
223             'VOLT␣␣␣1041␣[BUS1041␣␣␣␣␣130.00]',
                'VOLT␣␣␣1042␣[BUS104␣2␣␣␣␣130.00]',
225             'VOLT␣␣␣1043␣[BUS1043␣␣␣␣␣130.00]',
                'VOLT␣␣␣1044␣[BUS1044␣␣␣␣␣130.00]',
227             'VOLT␣␣␣1045␣[BUS1045␣␣␣␣␣130.00]',
                'VOLT␣␣␣2031␣[BUS2031␣␣␣␣␣220.00]',
229             'VOLT␣␣␣2032␣[BUS2032␣␣␣␣␣220.00]',
                'VOLT␣␣␣4011␣[BUS4011␣␣␣␣␣400.00]',
231             'VOLT␣␣␣4012␣[BUS4012␣␣␣␣␣400.00]',
                'VOLT␣␣␣4021␣[BUS4021␣␣␣␣␣400.00]',
233             'VOLT␣␣␣4022␣[BUS4022␣␣␣␣␣400.00]',
                'VOLT␣␣␣4031␣[BUS4031␣␣␣␣␣400.00]',
235             'VOLT␣␣␣4032␣[BUS4032␣␣␣␣␣400.00]',
                'VOLT␣␣␣4041␣[BUS4041␣␣␣␣␣400.00]',
237             'VOLT␣␣␣4042␣[BUS4042␣␣␣␣␣400.00]',
```

```
                    'VOLT    4043 [BUS4043      400.00]',
239                 'VOLT    4044 [BUS4044      400.00]',
                    'VOLT    4045 [BUS4045      400.00]',
241                 'VOLT    4046 [BUS4046      400.00]',
                    'VOLT    4047 [BUS4047      400.00]',
243                 'VOLT    4051 [BUS4051      400.00]',
                    'VOLT    4061 [BUS4061      400.00]',
245                 'VOLT    4062 [BUS4062      400.00]',
                    'VOLT    4063 [BUS4063      400.00]',
247                 'VOLT    4071 [BUS4071      400.00]',
                    'VOLT    4072 [BUS4072      400.00]']
249

251     Vbus_delta_series =    ['ANGL      1 [BUS1        20.000]',
                                'ANGL      2 [BUS2        20.000]',
253                             'ANGL      3 [BUS3        20.000]',
                                'ANGL      4 [BUS4        20.000]',
255                             'ANGL      5 [BUS5        20.000]',
                                'ANGL     11 [BUS11       20.000]',
257                             'ANGL     12 [BUS12       20.000]',
                                'ANGL     13 [BUS13       20.000]',
259                             'ANGL     22 [BUS22       20.000]',
                                'ANGL     31 [BUS31       20.000]',
261                             'ANGL     32 [BUS32       20.000]',
                                'ANGL     41 [BUS41       20.000]',
263                             'ANGL     42 [BUS42       20.000]',
                                'ANGL     43 [BUS43       20.000]',
265                             'ANGL     46 [BUS46       20.000]',
                                'ANGL     47 [BUS47       20.000]',
267                             'ANGL     51 [BUS51       20.000]',
                                'ANGL     61 [BUS61       20.000]',
269                             'ANGL     62 [BUS62       20.000]',
                                'ANGL     63 [BUS63       20.000]',
271                             'ANGL     71 [BUS71       20.000]',
                                'ANGL     72 [BUS72       20.000]',
273                             'ANGL    901 [GEN1        15.000]',
                                'ANGL    902 [GEN2        15.000]',
275                             'ANGL    903 [GEN3        15.000]',
                                'ANGL    904 [GEN4        15.000]',
277                             'ANGL    905 [GEN5        15.000]',
```

```
                'ANGL    906 [GEN6         15.000]',
279             'ANGL    907 [GEN7         15.000]',
                'ANGL    908 [GEN8         15.000]',
281             'ANGL    909 [GEN9         15.000]',
                'ANGL    910 [GEN10        15.000]',
283             'ANGL    911 [GEN11        15.000]',
                'ANGL    912 [GEN12        15.000]',
285             'ANGL    913 [GEN13        15.000]',
                'ANGL    914 [GEN14        15.000]',
287             'ANGL    915 [GEN15        15.000]',
                'ANGL    916 [GEN16        15.000]',
289             'ANGL    917 [GEN17        15.000]',
                'ANGL    918 [GEN18        15.000]',
291             'ANGL    919 [GEN19        15.000]',
                'ANGL    920 [GEN20        15.000]',
293             'ANGL   1011 [BUS1011      130.00]',
                'ANGL   1012 [BUS1012      130.00]',
295             'ANGL   1013 [BUS1013      130.00]',
                'ANGL   1014 [BUS1014      130.00]',
297             'ANGL   1021 [BUS1021      130.00]',
                'ANGL   1022 [BUS102 2     130.00]',
299             'ANGL   1041 [BUS1041      130.00]',
                'ANGL   1042 [BUS104 2     130.00]',
301             'ANGL   1043 [BUS1043      130.00]',
                'ANGL   1044 [BUS1044      130.00]',
303             'ANGL   1045 [BUS1045      130.00]',
                'ANGL   2031 [BUS2031      220.00]',
305             'ANGL   2032 [BUS2032      220.00]',
                'ANGL   4011 [BUS4011      400.00]',
307             'ANGL   4012 [BUS4012      400.00]',
                'ANGL   4021 [BUS4021      400.00]',
309             'ANGL   4022 [BUS4022      400.00]',
                'ANGL   4031 [BUS4031      400.00]',
311             'ANGL   4032 [BUS4032      400.00]',
                'ANGL   4041 [BUS4041      400.00]',
313             'ANGL   4042 [BUS4042      400.00]',
                'ANGL   4043 [BUS4043      400.00]',
315             'ANGL   4044 [BUS4044      400.00]',
                'ANGL   4045 [BUS4045      400.00]',
317             'ANGL   4046 [BUS4046      400.00]',
```

```
                                'ANGL    4047 [BUS4047      400.00]',
319                             'ANGL    4051 [BUS4051      400.00]',
                                'ANGL    4061 [BUS4061      400.00]',
321                             'ANGL    4062 [BUS4062      400.00]',
                                'ANGL    4063 [BUS4063      400.00]',
323                             'ANGL    4071 [BUS4071      400.00]',
                                'ANGL    4072 [BUS4072      400.00]']
325
    #
        ######################################################################

327 #
        ######################################################################


329 #Time recording
    t = data['time']
331
    #Pre-allocating
333 matrix_scale = len(t)
    delta = np.zeros([nr_G, matrix_scale])
335 omega = np.zeros([nr_G, matrix_scale])
    Pe = np.zeros([nr_G, matrix_scale])
337 Pm = np.zeros([nr_G, matrix_scale])
    V = np.zeros([nr_G, matrix_scale])
339 V_delta = np.zeros([nr_G, matrix_scale])
    Q = np.zeros([nr_G, matrix_scale])
341 Vbus = np.zeros([len(Vbus_series), matrix_scale])
    Vbus_delta = np.zeros([len(Vbus_delta_series), matrix_scale
        ])
343


345
    for i in range(nr_G):
347     delta[i,:] = data[delta_series[i]]
        omega[i,:] = data[omega_series[i]]
349     Pe[i,:] = data[Pe_series[i]]
        Pm[i,:] = data[Pm_series[i]]
351     V[i,:] = data[V_series[i]]
        V_delta[i,:] = data[V_delta_series[i]]
```

```
353         Q[i,:] = data[Q_series[i]]

355     for j in range(len(Vbus_series)):
            Vbus[j,:] = data[Vbus_series[j]]
357         Vbus_delta[j,:] = data[Vbus_delta_series[j]]


359


361
        delta = delta*np.pi/180                 # Rotor angle in rads
363     V_delta = V_delta*np.pi/180             # Voltage angle in rads
        Vbus_delta = Vbus_delta*np.pi/180       # Voltage angle in rads
365     omega = omega + 1                       # Absolute omega

367     #Make voltage into complex values
        V = V*np.exp(1j*V_delta)
369     Vbus = Vbus*np.exp(1j*Vbus_delta)

371     #Replace the base for Pm with Sbase
        Sbase = 100                 #Unit in MVA
373     Mbase = np.zeros(nr_G)
        Mbase[0] = 800.00           #Unit in MVA
375     Mbase[1] = 600.00
        Mbase[2] = 700.00
377     Mbase[3] = 600.00
        Mbase[4] = 250.00
379     Mbase[5] = 400.00
        Mbase[6] = 200.00
381     Mbase[7] = 850.00
        Mbase[8] = 1000.00
383     Mbase[9] = 800.00
        Mbase[10] = 300.00
385     Mbase[11] = 350.00
        Mbase[12] = 300.00
387     Mbase[13] = 700.00
        Mbase[14] = 1200.00
389     Mbase[15] = 700.00
        Mbase[16] = 600.00
391     Mbase[17] = 1200.00
        Mbase[18] = 500.00
```

```
393     Mbase[19] = 4500.00

395     for k in range(nr_G):
            Pm[k,:] = Pm[k,:]*Mbase[k]/Sbase      #Unite base to Sbase
397


399
        #Calculate the Moment of inertia
401     M = 2*np.array([3, 3, 3, 3, 3, 6, 6, 3, 3, 3, \
                        3, 3, 2, 6, 6, 6, 6, 6, 3, 3])
403     M = M*Mbase/Sbase

405     #Calculate I_tr
        S = Pe + 1j*Q
407     I_tr = np.conj(S/V)

409
        return delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale,
            t
```

Admittance Matrix in Pre- & Post-fault

```
1 ##########################################
  ###### Ybus construction (pre-fault) ######
3 ##########################################
  #Code here is the function for constructing the Ybus, extended
     Ybus matrix basing
5 #the .raw file by using the (J^T)*(Yprim)*J

7 def red_Ybus_pre(B, direction, nr_G, Vbus):
      import numpy as np
9     global J, Y_prim, B_B, B_f, YLoad, Xd_prime, Ybus_pre,
         Yaug_pre, Ybg, Ygg, Y_red_pre

11    #Prepare the vector of transformer ratio
      ratio = np.zeros(50)
13    ratio[0] = 0.9979
      ratio[1] = 0.9964
15    ratio[2] = 1.0054
      ratio[3] = 0.9829
17    ratio[4] = 0.9904
      ratio[5] = 1.0202
19    ratio[6] = 1.0298
      ratio[7] = 1.0400
21    ratio[8] = 0.9826
      ratio[9] = 0.9615
23    ratio[10] = 1.0003
      ratio[11] = 1.0429
25    ratio[12] = 1.0354
      ratio[13] = 1.0354
27    ratio[14] = 1.0279
      ratio[15] = 1.0129
29    ratio[16] = 1.0354
      ratio[17] = 1.0504
31    ratio[18] = 1.0504
      ratio[19] = 1.0504
33    ratio[20] = 1.0300
      ratio[21] = 1.0431
35    ratio[22] = 1
      ratio[23] = 1
37    ratio[24] = 1
```

```
    ratio[25] = 1
    ratio[26] = 1.05
    ratio[27] = 1.05
    ratio[28] = 1.05
    ratio[29] = 1.05
    ratio[30] = 1.05
    ratio[31] = 1.05
    ratio[32] = 1.05
    ratio[33] = 1.05
    ratio[34] = 1.05
    ratio[35] = 1.05
    ratio[36] = 1.05
    ratio[37] = 1.05
    ratio[38] = 1.05
    ratio[39] = 1.05
    ratio[40] = 1.05
    ratio[41] = 1.05
    ratio[42] = 0.95
    ratio[43] = 0.95
    ratio[44] = 0.93
    ratio[45] = 1.03
    ratio[46] = 1.03
    ratio[47] = 1.04
    ratio[48] = 1.04
    ratio[49] = 1.00


    #Basing on the data of the .raw file, pre-allocating the J
        matrix
    J = np.zeros([len(direction),len(B)])
    #Construct the J matirx
    for i in range(0, len(direction)):
        if i <= 51:
            J[i,direction[i,0]] = 1; J[i,direction[i,1]] = -1
        else:
            J[i,direction[i,0]] = 1; J[i,direction[i,1]] = -1/
                ratio[i-52]


```

```
    #Construct the Y_prim
77  #Y_prim is a diagonal matrix, who holds the admittances
        regarding to the
    #corresponding branch.
79  admittance = np.zeros(len(direction), dtype = complex)
    #Branch admittance
81  admittance[0] = 1/(1e-2 + 1j*(7e-2))
    admittance[1] = 1/(1e-2 + 1j*(7e-2))
83  admittance[2] = 1/(1.4e-2 + 1j*(9e-2))
    admittance[3] = 1/(1.4e-2 + 1j*(9e-2))
85  admittance[4] = 1/(7e-3 + 1j*(5e-2))
    admittance[5] = 1/(7e-3 + 1j*(5e-2))
87  admittance[6] = 1/(3e-2 + 1j*(2e-1))
    admittance[7] = 1/(3e-2 + 1j*(2e-1))
89  admittance[8] = 1/(1e-2 + 1j*(6e-2))
    admittance[9] = 1/(1e-2 + 1j*(6e-2))
91  admittance[10] = 1/(1.5e-2 + 1j*(1.2e-1))
    admittance[11] = 1/(1.5e-2 + 1j*(1.2e-1))
93  admittance[12] = 1/(3.8e-2 + 1j*(2.8e-1))
    admittance[13] = 1/(3.8e-2 + 1j*(2.8e-1))
95  admittance[14] = 1/(5e-2 + 1j*(3e-1))
    admittance[15] = 1/(1e-2 + 1j*(8e-2))
97  admittance[16] = 1/(1e-2 + 1j*(8e-2))
    admittance[17] = 1/(1.2e-2 + 1j*(9e-2))
99  admittance[18] = 1/(1.2e-2 + 1j*(9e-2))
    admittance[19] = 1/(1e-3 + 1j*(8e-3))
101 admittance[20] = 1/(6e-3 + 1j*(6e-2))
    admittance[21] = 1/(4e-3 + 1j*(4e-2))
103 admittance[22] = 1/(5e-3 + 1j*(4.5e-2))
    admittance[23] = 1/(4e-3 + 1j*(3.5e-2))
105 admittance[24] = 1/(5e-3 + 1j*(5e-2))
    admittance[25] = 1/(4e-3 + 1j*(4e-2))
107 admittance[26] = 1/(1e-2 + 1j*(6e-2))
    admittance[27] = 1/(4e-3 + 1j*(4e-2))
109 admittance[28] = 1/(4e-3 + 1j*(4e-2))
    admittance[29] = 1/(1e-3 + 1j*(1e-2))
111 admittance[30] = 1/(6e-3 + 1j*(4e-2))
    admittance[31] = 1/(6e-3 + 1j*(4e-2))
113 admittance[32] = 1/(1e-2 + 1j*(4e-2))
    admittance[33] = 1/(6e-3 + 1j*(5e-2))
```

```
115     admittance[34] = 1/(3e-3 + 1j*(3e-2))
        admittance[35] = 1/(6e-3 + 1j*(4.5e-2))
117     admittance[36] = 1/(2e-3 + 1j*(1.5e-2))
        admittance[37] = 1/(2e-3 + 1j*(2e-2))
119     admittance[38] = 1/(1e-3 + 1j*(1e-2))
        admittance[39] = 1/(1e-3 + 1j*(1e-2))
121     admittance[40] = 1/(2e-3 + 1j*(2e-2))
        admittance[41] = 1/(2e-3 + 1j*(2e-2))
123     admittance[42] = 1/(2e-3 + 1j*(2e-2))
        admittance[43] = 1/(4e-3 + 1j*(4e-2))
125     admittance[44] = 1/(4e-3 + 1j*(4e-2))
        admittance[45] = 1/(1.1e-2 + 1j*(8e-2))
127     admittance[46] = 1/(1e-3 + 1j*(1.5e-2))
        admittance[47] = 1/(2e-3 + 1j*(2e-2))
129     admittance[48] = 1/(3e-3 + 1j*(3e-2))
        admittance[49] = 1/(3e-3 + 1j*(3e-2))
131     admittance[50] = 1/(3e-3 + 1j*(3e-2))
        admittance[51] = 1/(3e-3 + 1j*(3e-2))
133
        #Transformer admittance
135     admittance[52] = 1/(1j*0.008300)
        admittance[53] = 1/(1j*0.016700)
137     admittance[54] = 1/(1j*0.021700)
        admittance[55] = 1/(1j*0.006300)
139     admittance[56] = 1/(1j*0.007100)
        admittance[57] = 1/(1j*0.025000)
141     admittance[58] = 1/(1j*0.016700)
        admittance[59] = 1/(1j*0.050000)
143     admittance[60] = 1/(1j*0.017900)
        admittance[61] = 1/(1j*0.050000)
145     admittance[62] = 1/(1j*0.025000)
        admittance[63] = 1/(1j*0.009300)
147     admittance[64] = 1/(1j*0.012500)
        admittance[65] = 1/(1j*0.005600)
149     admittance[66] = 1/(1j*0.007100)
        admittance[67] = 1/(1j*0.050000)
151     admittance[68] = 1/(1j*0.006300)
        admittance[69] = 1/(1j*0.010000)
153     admittance[70] = 1/(1j*0.016700)
        admittance[71] = 1/(1j*0.008500)
```

```
155     admittance[72] = 1/(1j*0.016700)
        admittance[73] = 1/(1j*0.002500)
157     admittance[74] = 1/(1j*0.018800)
        admittance[75] = 1/(1j*0.025000)
159     admittance[76] = 1/(1j*0.021400)
        admittance[77] = 1/(1j*0.025000)
161     admittance[78] = 1/(1j*0.060000)
        admittance[79] = 1/(1j*0.037500)
163     admittance[80] = 1/(1j*0.075000)
        admittance[81] = 1/(1j*0.017600)
165     admittance[82] = 1/(1j*0.015000)
        admittance[83] = 1/(1j*0.018800)
167     admittance[84] = 1/(1j*0.050000)
        admittance[85] = 1/(1j*0.042900)
169     admittance[86] = 1/(1j*0.033300)
        admittance[87] = 1/(1j*0.021400)
171     admittance[88] = 1/(1j*0.021400)
        admittance[89] = 1/(1j*0.021400)
173     admittance[90] = 1/(1j*0.025000)
        admittance[91] = 1/(1j*0.012500)
175     admittance[92] = 1/(1j*0.030000)
        admittance[93] = 1/(1j*0.003300)
177     admittance[94] = 1/(1j*0.008000)
        admittance[95] = 1/(1j*0.008000)
179     admittance[96] = 1/(1j*0.012001)
        admittance[97] = 1/(1j*0.010000)
181     admittance[98] = 1/(1j*0.010000)
        admittance[99] = 1/(1j*0.010000)
183     admittance[100] = 1/(1j*0.010000)
        admittance[101] = 1/(1j*0.012001)
185

187     #Then construct the Y_prim
        Y_prim = np.diag(admittance)
189
        #Assemble the Ybus matrix (without the branch shunt,
            compensation shunt and the load)
191     Ybus_pre = ((J.T).dot(Y_prim)).dot(J)

193
```

```
        #Branch shunts admittance
195     B_B = np.zeros(52)          #B for branches
        B_B[0] = 0.014000           #Unit in p.u.
197     B_B[1] = 0.014000
        B_B[2] = 0.018000
199     B_B[3] = 0.018000
        B_B[4] = 0.010000
201     B_B[5] = 0.010000
        B_B[6] = 0.030000
203     B_B[7] = 0.030000
        B_B[8] = 0.012000
205     B_B[9] = 0.012000
        B_B[10] = 0.025000
207     B_B[11] = 0.025000
        B_B[12] = 0.060000
209     B_B[13] = 0.060000
        B_B[14] = 0.060000
211     B_B[15] = 0.016000
        B_B[16] = 0.016000
213     B_B[17] = 0.017000
        B_B[18] = 0.017000
215     B_B[19] = 0.201000
        B_B[20] = 1.799000
217     B_B[21] = 1.201000
        B_B[21] = 1.402000
219     B_B[23] = 1.051000
        B_B[24] = 1.498000
221     B_B[25] = 1.201000
        B_B[26] = 3.001000
223     B_B[27] = 1.201000
        B_B[28] = 1.201000
225     B_B[29] = 0.302000
        B_B[30] = 2.398000
227     B_B[31] = 2.398000
        B_B[32] = 2.001000
229     B_B[33] = 2.398000
        B_B[34] = 0.900000
231     B_B[35] = 1.302000
        B_B[36] = 0.498000
233     B_B[37] = 0.598000
```

```
B_B[38] = 0.302000
B_B[39] = 0.302000
B_B[40] = 0.598000
B_B[41] = 0.598000
B_B[42] = 0.598000
B_B[43] = 1.201000
B_B[44] = 1.201000
B_B[45] = 2.398000
B_B[46] = 0.498000
B_B[47] = 0.598000
B_B[48] = 0.900000
B_B[49] = 0.900000
B_B[50] = 3.001000
B_B[51] = 3.001000
B_B = B_B*1j

#Add the branch shunt to the Ybus diagonal elements
for j in range(0, len(B_B)):
    Ybus_pre[direction[j,0], direction[j,0]] += B_B[j]/2
    Ybus_pre[direction[j,1], direction[j,1]] += B_B[j]/2


#Fixed shunt admittance
B_f = np.zeros(len(B))
B_f[B[1022]] = 50.00        #Unit in Mvar
B_f[B[1041]] = 250.00
B_f[B[1043]] = 200.00
B_f[B[1044]] = 200.00
B_f[B[1045]] = 200.00
B_f[B[4012]] = -100.00
B_f[B[4041]] = 200.00
B_f[B[4043]] = 200.00
B_f[B[4046]] = 100.00
B_f[B[4051]] = 100.00
B_f[B[4071]] = -400.00
B_f = B_f*1j              #Unit now in p.u.
#Translate to p.u.
Sbase = 100               #Unit in MVA
```

```
    for k in range(0, len(B)):
275         B_f[k] = B_f[k]/Sbase/(Vbus[k]*np.conj(Vbus[k]))


277
    #Add fixed branch to the Ybus_pre
279 for ii in range(0, len(B)):
        Ybus_pre[ii,ii] += B_f[ii]
281
    #Now start to generate the reduced Ybus matrix
283 #Prepare the transient Xd data (read data from dyr file)
    Xd_prime = np.zeros(nr_G)
285 Xd_prime[0] = 0.25          #Xd_prime
    Xd_prime[1] = 0.25          ##Unit in machine base, [pu]
287 Xd_prime[2] = 0.25
    Xd_prime[3] = 0.25
289 Xd_prime[4] = 0.25
    Xd_prime[5] = 0.3
291 Xd_prime[6] = 0.3
    Xd_prime[7] = 0.25
293 Xd_prime[8] = 0.25
    Xd_prime[9] = 0.25
295 Xd_prime[10] = 0.25
    Xd_prime[11] = 0.25
297 Xd_prime[12] = 0.3
    Xd_prime[13] = 0.3
299 Xd_prime[14] = 0.3
    Xd_prime[15] = 0.3
301 Xd_prime[16] = 0.3
    Xd_prime[17] = 0.3
303 Xd_prime[18] = 0.25
    Xd_prime[19] = 0.25
305 Xd_prime = Xd_prime*1j

307 #Machine base
    Mbase = np.zeros(nr_G)
309 Mbase[0] = 800.00           #Unit in MVA
    Mbase[1] = 600.00
311 Mbase[2] = 700.00
    Mbase[3] = 600.00
313 Mbase[4] = 250.00
```

```
        Mbase[5] = 400.00
315     Mbase[6] = 200.00
        Mbase[7] = 850.00
317     Mbase[8] = 1000.00
        Mbase[9] = 800.00
319     Mbase[10] = 300.00
        Mbase[11] = 350.00
321     Mbase[12] = 300.00
        Mbase[13] = 700.00
323     Mbase[14] = 1200.00
        Mbase[15] = 700.00
325     Mbase[16] = 600.00
        Mbase[17] = 1200.00
327     Mbase[18] = 500.00
        Mbase[19] = 4500.00
329     Xd_prime = Xd_prime*Sbase/Mbase       #Unite the machine base
            to system base


331
        #Calculate the load admittance
333     I_inj = Ybus_pre.dot(Vbus)
        YLoad = -I_inj/Vbus              # should add a minus???
335
        for jj in range(0, B[901]):
337         Ybus_pre[jj,jj] += YLoad[jj]

339     for pp in range(B[920] + 1, B[4072] + 1):
            Ybus_pre[pp,pp] += YLoad[pp]
341


343     Yaug_pre = Ybus_pre
        #Ybg matrix
345     Ybg = np.zeros([len(B), nr_G], dtype = complex)
        for kk in range(0, nr_G):
347         Ybg[B[901 + kk], kk] = -1/Xd_prime[kk]

349         Yaug_pre[B[901 + kk], B[901 + kk]] += 1/Xd_prime[kk]

351     Ygg = np.diag(1/Xd_prime)
```

```
353
        #Finally, assemble the reduced Ybus matrix
355     Y_red_pre = Ygg - ((Ybg.T).dot(np.linalg.inv(Yaug_pre))).dot
            (Ybg)

357
        return J, Y_prim, B_B, B_f, YLoad, Xd_prime, Ybus_pre,
            Yaug_pre, Ybg, Ygg, Y_red_pre
```

```
1 ##########################################
  ###### Ybus construction (post-fault) ######
3 ##########################################
  #Code here is the function for constructing the Ybus, extended
      Ybus matrix basing
5 #the .raw file by using the (J^T)*(Yprim)*J

7 def red_Ybus_post(B, direction, J, Y_prim, B_B, B_f, Vbus,
      Xd_prime, nr_G, Ybg, Ygg, trip_index):
      import numpy as np
9     global YLoad_post, Ybus_post, Yaug_post, Y_red_post

11    #Since when one line is triped, its corresponding impedence
          will be infinite,
      #which means the admittance will be 0, as well as the branch
          shunt admittance.
13    Y_prim[trip_index, trip_index] = 0
      B_B[trip_index] = 0
15
      #Then assemble the Ybus matrix again
17    Ybus_post = ((J.T).dot(Y_prim)).dot(J)


19
      #Add the branch shunt
21    for i in range(0, len(B_B)):
          Ybus_post[direction[i,0], direction[i,0]] += B_B[i]/2
23        Ybus_post[direction[i,1], direction[i,1]] += B_B[i]/2

25    #Add the Fixed shunt
      for j in range(0, len(B)):
27        Ybus_post[j,j] += B_f[j]
```

```
29      #Add the load admittance
        I_inj = Ybus_post.dot(Vbus)
31      YLoad_post = -I_inj/Vbus

33      for k in range(0, B[901]):
            Ybus_post[k,k] += YLoad_post[k]
35
        for p in range(B[920] + 1, B[4072] + 1):
37          Ybus_post[p,p] += YLoad_post[p]

39      Yaug_post = Ybus_post
        for ii in range(0, nr_G):
41          Yaug_post[B[901 + ii], B[901 + ii]] += 1/Xd_prime[ii]


43
        #Now start to generate the reduced Ybus matrix
45      Y_red_post = Ygg - ((Ybg.T).dot(np.linalg.inv(Yaug_post))).
           dot(Ybg)

47
        return  YLoad_post, Ybus_post, Yaug_post, Y_red_post
```

Coupling-Strength Calculation for Pre- & Post-fault

```
########################################
########## Coupling_Strength ##########
########################################
# The Grouping function here serves the whole system to provide a group
# classification before the implementation of OMIB, the most input will be the
# results from the simulatiion in PSS/E, however, since for this method, the
# internal voltage E' will be essencial, which makes the extended and reduced
# matrix something mandatory. With the help of these matrices, the internal voltage
# can be available so the elements inside Eq.[19] and Eq.[20] can be both achieved
# then the coupling and difference cupling matrix can be both available. In final,
# the clusters of the machines can be easily distinguished due to separation of the
# graph.
import numpy as np
import math


def coupling_strength_pre(Y_red, I_tr, delta, omega, M, Pm, Pe, nr_G):
    global H_pre, Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot

    ## Pre-allocating
    phi = np.zeros([nr_G, nr_G])        # Generator rotor angle difference
    d_phi = np.zeros([nr_G, nr_G])      # Generator rotor speed difference
    M_ij = np.zeros([nr_G, nr_G])       # Inertia coefficient
    Pm_ij = np.zeros([nr_G, nr_G])      # Generator Pm difference
    Pe_ij = np.zeros([nr_G, nr_G])      # Generator Pe difference
    Pa_ij = np.zeros([nr_G, nr_G])
    MT = np.zeros([nr_G, nr_G])
    Pc_ij = np.zeros([nr_G, nr_G])
```

```
    C_ij = np.zeros([nr_G, nr_G])
30  D_ij = np.zeros([nr_G, nr_G])
    Pmax_ij = np.zeros([nr_G, nr_G])
32  rho_ij  = np.zeros([nr_G, nr_G])
    Ekin_ij = np.zeros([nr_G, nr_G])
34  Edis_ij = np.zeros([nr_G, nr_G])
    H_pre   = np.zeros([nr_G, nr_G])
36  #For testing
    Pee_ij  = np.zeros([nr_G, nr_G])
38
    ## Calculate the internal voltage
40  E_dot = np.linalg.inv(Y_red).dot(I_tr)


42
    ## Elements construction
44  #  Construct the relative angle, speed, inertia coefficient,
        Pm and Pe matrix
    for i in range(0, nr_G):
46      for j in range(0, nr_G):
            #phi[i,j] = delta[i] - delta[j]
48          phi[i,j] = np.angle(E_dot[i]) - np.angle(E_dot[j])
                  #Not rotor angle here
            d_phi[i,j] = omega[i] - omega[j]
50          MT[i,j] = M[i] + M[j]
            M_ij[i,j] = M[i]*M[j]/MT[i,j]
52          Pm_ij[i,j] = (M[j]*Pm[i] - M[i]*Pm[j])/MT[i,j]
            Pee_ij[i,j] = (M[j]*Pe[i] - M[i]*Pe[j])/MT[i,j]
54          Pc_ij[i,j] = (M[j]*(np.abs(E_dot[i])**2)*np.abs(
                Y_red[i,i])*\
                  np.cos(np.angle(Y_red[i,i])) - M[i]*(np.abs(
                      E_dot[j])**2)*\
56                np.abs(Y_red[j,j])*np.cos(np.angle(Y_red[j,j]))
                      )/MT[i,j]

58          # Calculate the elements for Pmax_ij
            A = np.zeros(nr_G)
60          B = np.zeros(nr_G)
            C = np.zeros(nr_G)
62          D = np.zeros(nr_G)
```

```
64          for k in range(0, nr_G):
                if k == i:                  #Notes in the ancient
                    paper
66                  continue

68              A[k] = np.abs(E_dot[i])*np.abs(E_dot[k])*np.abs(
                    Y_red[i,k])*\
                np.cos(np.angle(E_dot[j]) - np.angle(E_dot[k]) -
                     np.angle(Y_red[i,k]))
70
                C[k] = np.abs(E_dot[i])*np.abs(E_dot[k])*np.abs(
                    Y_red[i,k])*\
72              np.sin(np.angle(E_dot[j]) - np.angle(E_dot[k]) -
                     np.angle(Y_red[i,k]))


74
            for kk in range(0, nr_G):
76              if kk == j:                 #Notes in the ancient
                    paper
                    continue
78
                B[kk] = np.abs(E_dot[j])*np.abs(E_dot[kk])*np.
                    abs(Y_red[j,kk])*\
80              np.cos(np.angle(E_dot[i]) - np.angle(E_dot[kk])
                    - np.angle(Y_red[j,kk]))

82              D[kk] = np.abs(E_dot[j])*np.abs(E_dot[kk])*np.
                    abs(Y_red[j,kk])*\
                np.sin(np.angle(E_dot[i]) - np.angle(E_dot[kk])
                    - np.angle(Y_red[j,kk]))
84


86          C_ij[i,j] =  (M[j]*np.sum(A) - M[i]*np.sum(B))/MT[i,
                j]
            D_ij[i,j] = -(M[j]*np.sum(C) + M[i]*np.sum(D))/MT[i,
                j]
88          Pmax_ij[i,j] = np.sqrt((C_ij[i,j])**2 + (D_ij[i,j])
                **2)
            rho_ij[i,j]  =  -math.atan(C_ij[i,j]/D_ij[i,j])
90
```

```
            # Then assemble the Pe_ij
92          # Pe_ij[i,j] = Pc_ij[i,j] + Pmax_ij[i,j]*np.sin(phi[
                i,j] - rho_ij[i,j])
            Pe_ij[i,j] = Pc_ij[i,j] + Pmax_ij[i,j]*np.sin(phi[i,
                j] - rho_ij[i,j])
94
            Pa_ij[i,j] = Pm_ij[i,j] - Pe_ij[i,j]
96
            # Calculate the kinetic energy
98          Ekin_ij[i,j] = 0.5*M_ij[i,j]*(d_phi[i,j])**2


100
            # Calcualte the dissipation energy
102         # Firstly, calculate the corresponding euqilibrium
                point and critical unstable point.
            phi_0_ij = phi[i,j]  #Since in pre-fault, it is
                already equilibrium
104
            # Then basing on the different situation find the
                critical angle.
106         if phi_0_ij >=0:
                phi_u_ij = np.pi - phi_0_ij
108         else:
                phi_u_ij = -np.pi - phi_0_ij
110

112         # Afterwards, basing on the integral euqation of
                Edis,ij
            Edis_ij[i,j] = (Pc_ij[i,j] - Pm_ij[i,j])*(phi_u_ij -
                phi_0_ij) - \
114         Pmax_ij[i,j]*(np.cos(phi_u_ij - rho_ij[i,j]) - np.
                cos(phi_0_ij - rho_ij[i,j]))


116
            # In final, calculate the coupling strength
118         H_pre[i,j] = Edis_ij[i,j] - Ekin_ij[i,j]


120
    return H_pre, Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot

########################################
```

```
########## Coupling_Strength ##########
########################################
# The Grouping function here serves the whole system to provide a group
# classification before the implementation of OMIB, the most input will be the
# results from the simulatiion in PSS/E, however, since for this method, the
# internal voltage E' will be essencial, which makes the extended and reduced
# matrix something mandatory. With the help of these matrices, the internal voltage
# can be available so the elements inside Eq.[19] and Eq.[20] can be both achieved
# then the coupling and difference cupling matrix can be both available. In final,
# the clusters of the machines can be easily distinguished due to separation of the
# graph.
import numpy as np
import math


def coupling_strength_post(Y_red, I_tr, delta, omega, M, Pm, Pe, nr_G):
    global H_post, E_dot_post

    ## Pre-allocating
    phi = np.zeros([nr_G, nr_G])        # Generator rotor angle difference
    d_phi = np.zeros([nr_G, nr_G])      # Generator rotor speed difference
    M_ij = np.zeros([nr_G, nr_G])       # Inertia coefficient
    Pm_ij = np.zeros([nr_G, nr_G])      # Generator Pm difference
    Pe_ij = np.zeros([nr_G, nr_G])      # Generator Pe difference
    Pa_ij = np.zeros([nr_G, nr_G])
    MT = np.zeros([nr_G, nr_G])
    Pc_ij = np.zeros([nr_G, nr_G])
    C_ij = np.zeros([nr_G, nr_G])
    D_ij = np.zeros([nr_G, nr_G])
```

```
    Pmax_ij = np.zeros([nr_G, nr_G])
32  rho_ij  = np.zeros([nr_G, nr_G])
    Ekin_ij = np.zeros([nr_G, nr_G])
34  Edis_ij = np.zeros([nr_G, nr_G])
    H_post   = np.zeros([nr_G, nr_G])
36

38  ## Calculate the internal voltage
    E_dot_post = np.linalg.inv(Y_red).dot(I_tr)
40
    ## Elements construction
42  #  Construct the relative angle, speed, inertia coefficient,
        Pm and Pe matrix
    for i in range(0, nr_G):
44      for j in range(0, nr_G):
            phi[i,j] = np.angle(E_dot_post[i]) - np.angle(
               E_dot_post[j])     #Not rotor angle here
46          d_phi[i,j] = omega[i] - omega[j]
            MT[i,j] = M[i] + M[j]
48          M_ij[i,j] = M[i]*M[j]/MT[i,j]
            Pm_ij[i,j] = (M[j]*Pm[i] - M[i]*Pm[j])/(M[i] + M[j])
50          Pc_ij[i,j] = (M[j]*(np.abs(E_dot_post[i])**2)*np.abs
               (Y_red[i,i])*\
                np.cos(np.angle(Y_red[i,i])) - M[i]*(np.abs(
                   E_dot_post[j])**2)*\
52              np.abs(Y_red[j,j])*np.cos(np.angle(Y_red[j,j]))
                   )/MT[i,j]

54          # Calculate the elements for Pmax_ij
            A = np.zeros(nr_G)
56          B = np.zeros(nr_G)
            C = np.zeros(nr_G)
58          D = np.zeros(nr_G)

60          for k in range(0, nr_G):
                if k == i:                      #Notes in the
                   ancient paper
62                  continue
                A[k] = np.abs(E_dot_post[i])*np.abs(E_dot_post[k
                   ])*np.abs(Y_red[i,k])*\
```

```
64              np.cos(np.angle(E_dot_post[j]) - np.angle(
                    E_dot_post[k]) - np.angle(Y_red[i,k]))

66              C[k] = np.abs(E_dot_post[i])*np.abs(E_dot_post[k
                    ])*np.abs(Y_red[i,k])*\
                np.sin(np.angle(E_dot_post[j]) - np.angle(
                    E_dot_post[k]) - np.angle(Y_red[i,k]))
68

70          for kk in range(0, nr_G):
                if kk == j:                     #Notes in the
                    ancient paper
72                  continue
                B[kk] = np.abs(E_dot_post[j])*np.abs(E_dot_post[
                    kk])*np.abs(Y_red[j,kk])*\
74              np.cos(np.angle(E_dot_post[i]) - np.angle(
                    E_dot_post[kk]) - np.angle(Y_red[j,kk]))

76              D[kk] = np.abs(E_dot_post[j])*np.abs(E_dot_post[
                    kk])*np.abs(Y_red[j,kk])*\
                np.sin(np.angle(E_dot_post[i]) - np.angle(
                    E_dot_post[kk]) - np.angle(Y_red[j,kk]))
78

80          C_ij[i,j] =  (M[j]*np.sum(A) - M[i]*np.sum(B))/MT[i,
                j]
            D_ij[i,j] = -(M[j]*np.sum(C) + M[i]*np.sum(D))/MT[i,
                j]
82          Pmax_ij[i,j] = np.sqrt((C_ij[i,j]**2) + (D_ij[i,j])
                **2)
            rho_ij[i,j]  =  -math.atan(C_ij[i,j]/D_ij[i,j])
84
            # Then assemble the Pe_ij
86          Pe_ij[i,j] = Pc_ij[i,j] + Pmax_ij[i,j]*np.sin(phi[i,
                j] - rho_ij[i,j])
            Pa_ij[i,j] = Pm_ij[i,j] - Pe_ij[i,j]
88
            # Calculate the kinetic energy
90          Ekin_ij[i,j] = 0.5*M_ij[i,j]*(d_phi[i,j])**2
```

```
92
                # Calcualte the dissipation energy
94              # Firstly, calculate the corresponding euqilibrium
                    point and critical unstable point.
                phi_0_ij = math.asin((Pm_ij[i,j] - Pc_ij[i,j])/
                    Pmax_ij[i,j]) + rho_ij[i,j]
96
                # Then basing on the different situation find the
                    critical angle.
98              if phi_0_ij >=0:
                    phi_u_ij = np.pi - phi_0_ij
100             else:
                    phi_u_ij = -np.pi - phi_0_ij
102

104             # Afterwards, basing on the integral euqation of
                    Edis,ij
                Edis_ij[i,j] = (Pc_ij[i,j] - Pm_ij[i,j])*(phi_u_ij -
                    phi[i,j]) - \
106             Pmax_ij[i,j]*(np.cos(phi_u_ij - rho_ij[i,j]) - np.
                    cos(phi[i,j] - rho_ij[i,j]))

108
                # In final, calculate the coupling strength
110             H_post[i,j] = Edis_ij[i,j] - Ekin_ij[i,j]

112
        return H_post, E_dot_post
```

Clustering & Grouping

```
##########################################
######### The Clustering function #########
##########################################
import numpy as np


def Clustering(nr_G, H_pre, H_post):
    global critical_index, Delta_H, Delta_h, Adj

    # Calculate the difference matrix in coupling strength
       between post- and pre-fault.
    Delta_H = H_post - H_pre

    # Then construct the Delta_h vector
    # But firstly, decompose the Delta_H matrix and make it a
       vector which holds
    # each column of Delta_H connecting together.
    Delta_h = Delta_H[:,0]
    for i in range(1, nr_G):
        Delta_h = np.block([[Delta_h, Delta_H[:,i]]])
    Delta_h = Delta_h[0,:]

    # Afterwards order this vector into a ascending order.
    Delta_h = Delta_h.tolist()
    Delta_h = sorted(Delta_h)

    # Then record the index for each element inside the Delta_h
       vector.
    index = np.zeros([nr_G*nr_G, 2], dtype = int)
    for j in range(0, nr_G*nr_G):
        location = np.where(Delta_H == Delta_h[j])
        index[j,0] = location[0][0]
        index[j,1] = location[1][0]


    ###################################
    ## DFS method basing on Graph theory
    ###################################
    # Pre-allocating the adjacency matrix
```

```
    # Basing on graph theory, the adjacency matrix shows the
      potential
38  # relationship between the generators, so in the beginning
      just assuming
    # there is a relation, which just sets 1 in the beginning.
40
    #'Numerical method'
42  Adj = np.ones([nr_G, nr_G])
    critical_index = np.zeros(nr_G)   # index for recording the
      CMs
44  for k in range(0, nr_G*nr_G):
        # Here the Adj follows the index of Delta_H
46      Adj[index[k,0], index[k,1]] = 0
        Adj[index[k,1], index[k,0]] = 0
48
        # Check if there is a separation already happened.
50      for ii in range(0, nr_G):
            if np.sum(Adj[:,ii]) <= 4:
52              critical_index[ii] = 1

54      if Delta_h[k] >= -5:
            break
56

58  #'Extended numerical method'
    key = 0          #The trigger to activate the next algorithm
      , if necessary.
60  if len(filter(None, critical_index)) == 0:
        beacon = np.zeros(nr_G)
62      for jj in range(0, nr_G):
            for kk in range(0, nr_G):
64              if Delta_H[jj,kk] <= -5:
                    beacon[jj] += 1
66                  Adj[jj,kk] = 0
                    Adj[kk,jj] = 0
68
            if beacon[jj] >= 8:
70              critical_index[jj] = 1
                key = 1
72
```

```
74      #If nothing detected in the numerical method, then implement
            the
        #'Clock Method'
76      trigger = False          # Trigger of collapse happened
        if len(filter(None, critical_index)) == 0 and key == 0:
78          #'Clock' method
            Adj = np.ones([nr_G, nr_G])
80          clock    = 0                          # Clcok for extra
                loops
            critical_index = np.zeros(nr_G)    # index for recording
                the CMs
82          for iii in range(0, nr_G*nr_G):
                # Here the Adj follows the index of Delta_H
84              Adj[index[iii,0], index[iii,1]] = 0
                Adj[index[iii,1], index[iii,0]] = 0
86
                # Check if there is a separation already happened.
88              for jjj in range(0, nr_G):
                    if not trigger:
90                      if np.sum(Adj[:,jjj]) <= 4:
                            # Then separation (the first one)
                                already happened.
92                          trigger = True
                            critical_index[jjj] = 1
94                  else:
                        if np.sum(Adj[:,jjj]) <= 4:
96                          critical_index[jjj] = 1

98              # After the first separation happened, just checking
                    for 20 extra loop
                # to see if there is another collapse.(This is
                    basing on an assumption
100             # that CMs can be clustered in short time interval.)
                if trigger:
102                 clock += 1
                # And after 20 extra loop, break the outer loop
104             if clock == 20:
                    break
106
```

```
    print '####################################'
    print '#%%%%%%% DFS Method Report %%%%%%%#'
    print '####################################'
    if key == 0 and trigger == False:
        print 'Using the Numerical method'
    elif key == 1:
        print 'Using the Extended numerical method'
    else:
        print 'Using the Clock Method'

    print 'Critical machine identification:'
    print critical_index
    print ''

    return critical_index, Delta_H, Delta_h, Adj
```

```
##########################################
########## The Grouping function ##########
##########################################

import numpy as np

def Grouping(critical_index, delta, omega, M, Pm, Pe,
    matrix_scale):
    global MK, Mj, delta_K, delta_j, omega_K, omega_j, Pm_K,
        Pm_j, Pe_K, Pe_j

    # Use some trick to have the rest of elements as the non-
        critical group.
    noncritical_index = (critical_index - 1)*(-1)  # Then the
        '1' in the original
    # array now is zero,
    # however, the 0 in original corresponding to the non-
        critical now is -1.

    delta_K = np.zeros([len(filter(None, critical_index)),
        matrix_scale])
    omega_K = np.zeros([len(filter(None, critical_index)),
        matrix_scale])
    Pm_K    = np.zeros([len(filter(None, critical_index)),
        matrix_scale])
```

```
18      Pe_K     = np.zeros([len(filter(None, critical_index)),
            matrix_scale])

20      delta_j = np.zeros([len(filter(None, noncritical_index)),
            matrix_scale])
        omega_j = np.zeros([len(filter(None, noncritical_index)),
            matrix_scale])
22      Pm_j     = np.zeros([len(filter(None, noncritical_index)),
            matrix_scale])
        Pe_j     = np.zeros([len(filter(None, noncritical_index)),
            matrix_scale])
24

26      for i in range(0, matrix_scale):
            # Use the filter function to eliminate the non-critical
                stuffs firstly, so
28          # what is left will belong to the critical group.
            delta_K[:,i] = np.array(filter(None, (critical_index*
                delta[:,i])))
30          omega_K[:,i] = np.array(filter(None, (critical_index*
                omega[:,i])))
            Pm_K[:,i]    = np.array(filter(None, (critical_index*Pm
                [:,i])))
32          Pe_K[:,i]    = np.array(filter(None, (critical_index*Pe
                [:,i])))

34          # And the rest will be the Non-Critical group.
            delta_j[:,i] = np.array(filter(None, (noncritical_index*
                delta[:,i])))
36          omega_j[:,i] = np.array(filter(None, (noncritical_index*
                omega[:,i])))
            Pm_j[:,i]    = np.array(filter(None, (noncritical_index*
                Pm[:,i])))
38          Pe_j[:,i]    = np.array(filter(None, (noncritical_index*
                Pe[:,i])))
            # Now so far, the grouping method is done and the
                critical and non-critical
40          # machines has been separated perfectly.

42      MK = np.array(filter(None, (critical_index*M)))
```

```
    Mj = np.array(filter(None, (noncritical_index*M)))
44
    return MK, Mj, delta_K, delta_j, omega_K, omega_j, Pm_K,
        Pm_j, Pe_K, Pe_j
```

Transcendental Function Solver

```
import numpy as np
import sympy
from sympy import *

def func_solver(A, B, C):
    global delta_r_OMIB, step_count
    var('x')
    func = A*x + B*cos(x) - C

    ## Here we are using the second order 'repeated root
       iteration'
    derivative_1st = A - B*sin(x)
    derivative_2rd = -B*cos(x)

    CHF = x - (func*derivative_1st)/(derivative_1st**2 \
              - func*derivative_2rd)

    MAXSTEP = 100
    step_count = 0
    tolerance = 1e-5
    x0 = float(pi/2)
    temp = CHF.subs(x, x0)
    while step_count < MAXSTEP and abs(temp - x0) > tolerance:
        x0 = temp
        temp = CHF.subs(x, x0)
        step_count +=1

    delta_r_OMIB = x0


    return delta_r_OMIB, step_count
```

OMIB

```
1 ####################################
  ######### The OMIB function #########
3 ####################################
  # Here the function is a prototype, basing on the research on
      the orientation.
5 # And the function itself will be imported by the 'SIME'
      function, and the output of
  # this function will be the trajectory of the One Machine
      Infinite Bus. Then the
7 # stability of the whole system can be observed by the OMIB
      trajectory instead.

9 import numpy as np
  import math
11 import matplotlib.pyplot as plt
  import func_solver as fs
13 from sympy import *

15 def OMIB(t, MK, Mj, delta_K, delta_j, omega_K, omega_j, Pm_K,
      Pm_j, Pe_K, Pe_j, t_clear):
       global delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB, index_cross,
          index, eta
17
       #############
19     ## OMIB setup
       #############
21     MC = np.sum(MK, axis = 0)     # Critical machine inertia
       MN = np.sum(Mj, axis = 0)     # Non-critical machine inertia
23     # Pre-allocating
       delta_OMIB = np.zeros(len(t))
25     delta_C = np.zeros(len(t))
       delta_N = np.zeros(len(t))
27     omega_C = np.zeros(len(t))
       omega_N = np.zeros(len(t))
29     M_OMIB = np.zeros(len(t))
       Pm_OMIB = np.zeros(len(t))
31     Pe_OMIB = np.zeros(len(t))
       Pa_OMIB = np.zeros(len(t))
33     eta = np.zeros(len(t))
```

```
35
    for i in range(0, len(t)):
37      delta_C[i] = np.sum((MK*delta_K[:,i]), axis = 0)/MC
        delta_N[i] = np.sum((Mj*delta_j[:,i]), axis = 0)/MN
39      omega_C[i] = np.sum((MK*omega_K[:,i]), axis = 0)/MC
        omega_N[i] = np.sum((Mj*omega_j[:,i]), axis = 0)/MN
41

43  delta_OMIB = delta_C - delta_N
    omega_OMIB = omega_C - omega_N
45  M_OMIB = MC*MN/(MC + MN)

47  key = 0   #Boolean
    for j in range(0, len(t)):
49      Pm_OMIB[j] = M_OMIB*((np.sum(Pm_K[:,j], axis = 0))/MC -
            \
                (np.sum(Pm_j[:,j], axis = 0))/MN)
51      Pe_OMIB[j] = M_OMIB*((np.sum(Pe_K[:,j], axis = 0))/MC -
            \
                (np.sum(Pe_j[:,j], axis = 0))/MN)
53      Pa_OMIB[j] = Pm_OMIB[j] - Pe_OMIB[j]

55      # Find the index corresponding to the t_cross
        if  key == 0:
57          if t[j] >= t_clear:
                index = j
59              key = 1


61
    #
       ##########################################################################

63  ## Returning angle delta_r and critical unstable angle
       calculation
    #############
65  coefficient = np.zeros(len(t) - index)
    for k in range(0, len(t) - index):
67      coefficient[k] = Pe_OMIB[k + index]/np.sin(delta_OMIB[k
           + index])
```

```
69      coefficient = np.mean(coefficient)

71
        # Find the critical unstable angle delta_u_OMIB
73      delta_u_OMIB = np.pi - math.asin(Pm_OMIB[index]/coefficient)

75      # Start constructing the report
        print'
           ##################################################################
           '
77      print'#%%%%%%%%%%%%%%%%%%%%%%%$$$ Assessment Report $$$
           %%%%%%%%%%%%%%%%%%%%%%%#'
        print'
           ##################################################################
           '
79      # Find the angle that making Pe_OMIB cross Pm_OMIB in the
           first time
        if (Pe_OMIB[index] >= Pm_OMIB[index]):
81          # For normal situation
            delta_cross_OMIB = math.asin(Pm_OMIB[index]/coefficient)
83          index_cross = index
            print'A normal crossing case'
85      else:
            # For special situation that cross happening after the
               fault clearance
87          delta_cross_OMIB = math.asin(Pm_OMIB[index]/coefficient)
            for u in range(index+1, len(t)):
89              if (delta_OMIB[u] > delta_cross_OMIB):
                    index_cross = u
91                  print'A special crossing case'
                    break
93          if (delta_OMIB[len(t) - 1] < delta_cross_OMIB):
                index_cross = len(t) - 1
95              print'A special crossing case'

97
        # Then calculating the returning angle basing on EAC
99      # Calculate the Aacc firstly:
        ii = 1
```

```
101     Aacc = 0
        while ii <= index_cross:
103         d_delta_OMIB = delta_OMIB[ii] - delta_OMIB[ii-1]
            Aacc += (Pa_OMIB[ii] + Pa_OMIB[ii-1])*d_delta_OMIB/2
105         ii += 1
        ## The output here will be the Aacc
107


109     # Then solve the returning angle basing on Aacc = Adec
        # For convinence, the equation below is after the
            simpification of EAC
111     # ' A*delta_r_OMIB + B*np.cos(delta_r_OMIB) = C
        # However, it is still a transcendental function :(
113     # So the function here can only be solved basing on the
            iteration.
        # First of all, calculate the right side and constant
            coefficients of the
115     # equation, respectively.
        C = coefficient*np.cos(delta_cross_OMIB) + delta_cross_OMIB*
            Pm_OMIB[index] - Aacc
117     A = Pm_OMIB[index]
        B = coefficient
119     fs.func_solver(A, B, C)  #The returning angle can be
            achieved in here.


121
        flag = 0                                        #Feasible
            domain
123     if (fs.delta_r_OMIB > delta_u_OMIB or fs.delta_r_OMIB <=
            delta_cross_OMIB):
            flag = 1
125         print'There is no valid returing angle :('
        else:
127         print'Cross angle = ', delta_cross_OMIB
            print'Critical unstable angle', delta_u_OMIB
129         print'Returning angle, delta_r_OMIB = ', fs.delta_r_OMIB
            print'Total iteration steps = ', fs.step_count
131


133     #
```

```
    ########################################################################

## Margin calculation and judgement.
#############
# First of all, if there is no equilibrium between the Aacc
    and Adec, then
# the maximum iteration number will be reached.

# So the total iteration number can be a standard to judge
    if it is unstable:
if (fs.step_count == 100 or flag == 1):
    print 'Aacc > Adec, lose synchronism forever in the first
        swing'
    print '#%%%%%%%%%%%%%%%%%%%%%%%%%#'
    print 'This is an unstable case :('
    print '#%%%%%%%%%%%%%%%%%%%%%%%%%#'
    # So margin here will be negative, equal to Adec - Aacc
    # Then calculate the Adec firsstly, basing on the
        integral from
    # delta_cross_OMIB to delta_u_OMIB
    limit_down = delta_cross_OMIB
    limit_up   = delta_u_OMIB
    var('delta')
    f = -(Pm_OMIB[index] - coefficient*sin(delta))       #
        Adec - Aacc
    Adec = integrate(f, (delta, limit_down,limit_up))

    # Then margin can be expressed as Adec - Aacc
    eta_unstable = Adec - Aacc
    print 'Margin_unstable = ', eta_unstable


else:
    print 'Aacc < Adec, their is a returning angle = %.3f' %
        (fs.delta_r_OMIB)
    print '#%%%%%%%%%%%%%%%%%%%%%%#'
    print 'This is a stable case :)'
    print '#%%%%%%%%%%%%%%%%%%%%%%#'
    # So the margin here will be a positive value, which can
        be understood
```

```
165         # as how far the extra power is away from the returning
                angle to the
            # critical unstable angle.
167         limit_down = fs.delta_r_OMIB
            limit_up   = delta_u_OMIB
169         var('delta')
            f = -(Pm_OMIB[index] - coefficient*sin(delta))      #
                Adec - Aacc
171         eta_stable = integrate(f, (delta, limit_down, limit_up))
            print 'Margin_stable = ', eta_stable
173         # Here is the threshold to the marginal stability.
            if (eta_stable <= 2 and eta_stable >= 0):
175             print 'This is a marginal stability situation, so the
                    full time \
    domain simulation will be needed.'
177

179     return  delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB, index_cross,
            index, eta
```

Assessment

```
1 ########################################################
  ########### Mainfunction for the Assessment ############
3 ########################################################

5 def Assesment_function(nr_G, t_clear, B, direction, trip_index):
      global delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale,
          t, Ybg, Ygg, YLoad, \
7     Ybus_pre, Yaug_pre, Y_red_pre, YLoad_post, Ybus_post,
         Yaug_post, Y_red_post, H_pre, \
      Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot, H_post, E_dot_post,
         critical_index, Delta_H, \
9     Delta_h, Adj, MK, Mj, delta_K, delta_j, omega_K, omega_j,
         Pm_K, Pm_j, Pe_K, Pe_j\
      #, delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB, index_cross, index,
          eta
11
      ## Packages and functions imported
13    import numpy as np
      import seaborn as sns
15    import matplotlib.pyplot as plt
      import results_Reading as rR
17    import red_Ybus_pre as rYp
      import red_Ybus_post as rYpt
19    import coupling_strength_pre as csp
      import coupling_strength_post as cspt
21    import Clustering as Clr
      import Grouping as Gp
23    import OMIB as OMIB


25
      ##############################
27    ###### Data Preparation ######
      ##############################
29    #Reading the simulation results and put them in the proper
         position.
      delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale, t = rR
         .results_Reading(nr_G)
31
```

```
33      #Generate the Ybus, Y_red in the pre fault.
        J, Y_prim, B_B, B_f, YLoad, Xd_prime, Ybus_pre, Yaug_pre,
           Ybg, Ygg, Y_red_pre = \
35      rYp.red_Ybus_pre(B, direction, nr_G, Vbus[:,100])


37
        #Generate the Ybus, Y_red in the post fault.
39      YLoad_post, Ybus_post, Yaug_post, \
        Y_red_post = rYpt.red_Ybus_post(B, direction, J,  Y_prim,
           B_B, B_f, Vbus[:,-1], \
41                                      Xd_prime, nr_G, Ybg, Ygg, \
                                        trip_index)
43
        ################################################
45      ####### Clustering method implementation #######
        ################################################
47      #The coupling strength coefficient in the pre-fault
        H_pre, Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot = \
49      csp.coupling_strength_pre(Y_red_pre, I_tr[:,100], delta
           [:,100], omega[:,100], \
                                  M, Pm[:,100], Pe[:,100], nr_G)
51


53      #The coupling strengrh coefficient in the post-fault
        H_post, E_dot_post = \
55      cspt.coupling_strength_post(Y_red_post, I_tr[:,-3], \
                                    delta[:,-3], omega[:,-3], \
57                                  M, Pm[:,-3], Pe[:,-3], nr_G)


59
        #Report title
61      print'######################'
        print'Branch tripped case %d' % (trip_index + 1)
63      print'######################'
        #Clustering the machines into CMs and NMs
65      critical_index, Delta_H, Delta_h, Adj = Clr.Clustering(nr_G,
            H_pre, H_post)


67
        #Grouping the machines
```

```
69      MK, Mj, delta_K, delta_j, omega_K, omega_j, Pm_K, Pm_j, Pe_K
            , Pe_j = \
        Gp.Grouping(critical_index, delta, omega, M, Pm, Pe,
            matrix_scale)
71
73      ##############################################
        ########### Grouping results plot ###########
75      ##############################################
        #H_pre matrix heatmap plot
77      plt.figure(dpi = 150)
        figure_1 = sns.heatmap(H_pre)
79      plt.xlabel('Generators',fontsize = 25)
        plt.ylabel('Generators',fontsize = 25)
81      plt.title('Coupling␣strength␣in␣pre-fault',fontsize = 35)
        plt.tick_params(labelsize = 20)
83
85      #H_post matrix heatmap plot
        plt.figure(dpi = 150)
87      figure_2 = sns.heatmap(H_post)
        plt.xlabel('Generators',fontsize = 25)
89      plt.ylabel('Generators',fontsize = 25)
        plt.title('Coupling␣strength␣in␣post-fault',fontsize = 35)
91      plt.tick_params(labelsize = 20)

93
        #Delta_H matrix heatmap plot
95      plt.figure(dpi = 150)
        figure_3 = sns.heatmap(Delta_H)
97      plt.xlabel('Generators',fontsize = 25)
        plt.ylabel('Generators',fontsize = 25)
99      plt.title('Coupling␣strength␣difference',fontsize = 35)
        plt.tick_params(labelsize = 20)
101

103     #Adjacency matrix heatmap plot
        plt.figure(dpi = 150)
105     figure_4 = sns.heatmap(Adj)
        plt.xlabel('Generators',fontsize = 25)
```

```
107     plt.ylabel('Generators',fontsize = 25)
        plt.title('Adjacency matrix visualization',fontsize = 35)
109     plt.tick_params(labelsize = 20)


111
        '''
113     ########################################################
        ### OMIB Margin calculatiion and Stability Criteria ###
115     ########################################################
        #Call the OMIB function
117     delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB, index_cross, index,
            eta = \
        OMIB.OMIB(t, MK, Mj, delta_K, delta_j, omega_K, omega_j,
            Pm_K, Pm_j, Pe_K, Pe_j, t_clear)
119     '''


121
        return delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale,
            t, Ybg, Ygg, YLoad, \
123             Ybus_pre, Yaug_pre, Y_red_pre, YLoad_post, Ybus_post
                    , Yaug_post, Y_red_post, \
                H_pre, Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot, H_post,
                    E_dot_post, critical_index, \
125             Delta_H, Delta_h, Adj, MK, Mj, delta_K, delta_j,
                    omega_K, omega_j, Pm_K, Pm_j, \
                Pe_K, Pe_j #, delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB,
                    index_cross, index, eta
```

Automation

```
1 ##############################################
  #### Simulation & Assesment Ultimate File ####
3 ##############################################
  import numpy as np
5 import os
  import time
7 import Branch_information as Bi

9 #Parameters
  nr_G = 20
11 t_clear = 1.2
  t_final = t_clear + 0.01*3      ## Final simulation time.
13 #t_final = 1.5                 ## Final simulation time.


15
  #Apply the Buses and Branches information
17 B, direction = Bi.Branch_information()

19 #Counting the time
  start = time.clock()
21
  #Create branch tripping plan
23 for trip_index in range(0, 52):
      ################################################
25    ###### PSS/E real time-domain simulation ######
      ################################################
27
      import PSSE_simulation as PSSE_s
29    PSSE_s.PSSE_simulation(t_clear, t_final, B, direction,
          trip_index)

31    ########################################################
      #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
33    #####Boundary between Simulation and Assessment ######
      #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
35    ########################################################

37    #Case Title
      print'########'
```

```
39      print 'Case %d' % (trip_index + 1)
        print '########'
41
        ########################################
43      ###### Activate the assessment part ######
        ########################################
45      import Assesment_function as Asf
        delta, omega, Pe, Pm, M, I_tr, V, Vbus, matrix_scale, t, Ybg
            , Ygg, YLoad, Ybus_pre, Yaug_pre, \
47      Y_red_pre, YLoad_post, Ybus_post, Yaug_post, Y_red_post,
            H_pre, Pee_ij, Pe_ij, Pm_ij, Pc_ij, E_dot, H_post, \
        E_dot_post, critical_index, Delta_H, Delta_h, Adj, MK, Mj,
            delta_K, delta_j, omega_K, omega_j, Pm_K, \
49      Pm_j, Pe_K, Pe_j  = Asf.Assesment_function(nr_G, t_clear, B,
             direction, trip_index)
        #, delta_OMIB, M_OMIB, Pm_OMIB, Pe_OMIB, index_cross, index,
             eta \
51      print '%%%%%%%%%%%%%%%%%%%%This is the boundary between cases
            %%%%%%%%%%%%%%%%%%%%'
        print ''
53      print ''

55      #Clear the previous file
        os.remove('C:\Users\syh11\Desktop\Master_thesis\Code\
            Code_automation\example.out')
57
  #Counting the running time
59 end = time.clock()
  print 'Running time for %d cases\' clustering:' % (trip_index +
      1), (end - start), 'sec'
61 print ''
```

Department of Electrical Engineering
Center for Electric Power and Energy (CEE)
Technical University of Denmark
Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark

www.elektro.dtu.dk/cee
Tel: (+45) 45 25 35 00
Fax: (+45) 45 88 61 11
E-mail: cee@elektro.dtu.dk