Towards Three-Phase Dynamic Analysis of Large Electric Power Systems

Abhineet Himanshu Parchure

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical Engineering

06/11/2015
Blacksburg, Virginia

Keywords: Three-Phase Dynamic Analysis, Electromechanical Transients, Distributed Generation, Combined Transmission & Distribution (T&D) Modeling and Simulation, Unbalanced System Analysis, Power System Stability

Towards Three-Phase Dynamic Analysis of Large Electric Power Systems

Abhineet Himanshu Parchure

ABSTRACT

This thesis primarily focuses on studying the impact of Distributed Generation (DG) on the electromechanical transients in the electric grid (distribution, transmission or combined transmission and distribution (T&D) systems) using a Three Phase Dynamics Analyzer (hereafter referred to as TPDA). TPDA includes dynamic models for electric machines, their controllers, and a three-phase model of the electric grid, and performs three-phase dynamic simulations without assuming a positive sequence network model. As a result, TPDA can be used for more accurate investigation of electromechanical transients in the electric grid in the presence of imbalances.

At present, the Electromagnetic Transient Program (EMTP) software can be used to perform three-phase dynamic simulations. This software models the differential equations of the entire electric network along with those of the machines. This calls for solving differential equations with time constants in the order of milliseconds (representing the fast electric network) in tandem with differential equations with time constants in the order of seconds (representing the slower electromechanical machines). This results in a stiff set of differential equations, making such an analysis extremely time consuming. For the purpose of electromechanical transient analysis, TPDA exploits the difference in the order of time constants and adopts phasor analysis of the electric network, solving differential equations only for the equipment whose dynamics are much slower than those of the electric network. Power Flow equations are solved using a graph trace analysis based approach which, along with the explicit partitioned method adopted in TPDA, can eventually lead to the use of distributed computing that will further enhance the speed of TPDA and perhaps enable it to perform dynamic simulation in real time .

In the work presented here, first an overview of the methodology behind TPDA is provided. A description of the object oriented implementation of TPDA in C++/C# is included. Subsequently, TPDA is shown to accurately simulate power system dynamics of balanced networks by comparing its results against those obtained using GE-PSLF®. This is followed by an analysis that demonstrates the advantages of using TPDA by highlighting the differences in results when the same problem is analyzed using a three-phase network model with unbalances and the positive sequence network model as used in GE-PSLF®. Finally, the impact of rapidly varying DG generation is analyzed, and it is shown that as the penetration level of DG increases, the current and voltage oscillations throughout the transmission network increase as well. Further, rotor speed deviations are shown to grow proportionally with increasing DG penetration.

To My Parents: Himanshu Parchure and Deepa Parchure

Acknowledgements

I wish to extend my sincerest gratitude to Dr. Robert Broadwater for his guidance and support throughout this thesis work. I have had countless discussions and brain storming sessions with him which have filled me with inspiration and broadened the scope of my thinking. I am also very grateful to Dr. Jaime De La Ree who has constantly supported me. I have learnt a lot from him, especially through a summer research project. I am extremely thankful to Dr. Anne Ryan who extended her support whenever I approached her, during coursework and otherwise.

I would also like to thank Himanshu Jain who I have learnt from immensely, especially on the technical front. He has been the first 'go-to' person for all my queries and he has always responded by helping me out. I would also like to extend my gratitude to the rest of my lab mates and friends at Virginia Tech who have been by my side on countless occasions.

Finally, I would like to thank my parents and sister for their perennial love and motivation. Their encouragement, moral and financial support have been essential for the successful completion of my degree requirements.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In analyzing system stability it is necessary to solve for dynamic responses using numerical integration. One approach for performing a three-phase dynamic analysis would be to model the entire system with all of its machines together with the three-phase electric network through differential equations. However, this will lead to a system of differential equations with time constants that vary by several orders of magnitude. This results in a stiff set of differential equations which requires very small integration step sizes and thus a long computation time. To avoid this time consuming process, traditional, simplifying assumptions have included the system is balanced, the disturbance is balanced, and that the disturbed system remains symmetrical. This allows for a positive sequence representation of the system.

Several researchers have identified the need for three phase dynamic analysis. In [1] the authors argue that in a traditional power system line impedance imbalance caused by un-transposed lines and feeders, along with single phase loads, impose unbalanced conditions on the entire power system. Unbalanced voltages at machine terminals can result in large negative sequence currents in the rotor which result in increased losses, affecting the dynamic behavior, and more significantly, raising alarms due to excessive heating. Designers and protection engineers have been concerned with the heating aspect and have paid attention to limiting the negative sequence currents in the stator to limit the negative sequence currents in the rotor [2]. However, power system dynamic studies, especially for large T&D systems, have mostly ignored the imbalance effects on the dynamic response of the rotor speed of each machine. Now, with the rapid proliferation of fast varying Distributed Generation (DG) in the distribution system, that is inherently unbalanced, there is a need-more than ever to consider the impact of imbalance on generation sources connected to the transmission network.

Reference [3] summarizes some reasons that highlight the need for three phase analysis. These are summarized as:

1) There are unbalanced impedances in 3-phase transmission due to non-transposition of the lines
2) There are unbalanced three phase loads

3) There are one-phase or two-phase lines in distribution networks.
4) There are one-phase and two-phase loads
5) There are also individual phase controls, resulting in unbalances.

In addition to these, many DG generators are single phase which may increase the imbalance in the system.

In [4] authors perform a three phase power flow analysis in an attempt to capture the imbalance among the three phases. However, they go on to convert the network into a positive sequence equivalent circuit and neglect the effects of negative sequence and zero sequence components. Further, they present rotor angle results that exactly match those obtained using the traditional balanced network approach for unbalanced faults and fail to capture the expected 120Hz oscillations in the rotor speed.

Research efforts based on the assumption of a balanced transmission network have shown that the implementation of DG at the distribution level may influence the technical aspects of the distribution grids [5]-[8] but not significantly impact power system transient stability (when connected in small amounts) [9]. At the same time, it has also been shown that as the penetration of DG increases, its impact is no longer restricted to the distribution system, and it begins to influence the entire grid [10], [11]. Several studies have been conducted to model and observe the impacts of considerable DG penetration on the dynamic behavior of the power system. In [12]-[14], the impact of different types of DG have been analyzed. The general impacts of DG on power system stability are presented in [9] by investigating 5 different DG technologies. Reference [15] investigates the transient stability of transmission systems by observing the behavior of individual bulk generators when the system is subjected to a particular fault. However, all of these works are based on traditional positive sequence model dynamics and 3-phase dynamic simulations are not considered.

In order to overcome the above limitations and accurately simulate the impact of network imbalance on the study of electromechanical transients, a new three-phase dynamic analysis technique is considered. The software implementation of this technique is called the Three-Phase Dynamics Analyzer (TPDA). The algorithm underlying TPDA was being developed by Himanshu

Jain, as a part of his doctoral research under the guidance of Dr. Robert Broadwater. As a part of this thesis, I contributed to the development of this algorithm by incorporating certain modifications that helped improve TPDA's performance by successfully distinguishing between balanced and unbalanced system conditions. Reference [16] provides the mathematical basis for the algorithm behind TPDA and also discusses its calibration against GE-PSLF® and the Alternative Transients Program (ATP). TPDA was then added as an application within a multi-phase graph trace analysis based power flow simulator, Distributed Engineering Workstation (DEW) [17]. Since TPDA models all three-phases of the electric network, it avoids the inaccuracies that may arise due to positive sequence model assumption. At the same time, since TPDA is aimed at studying electromechanical oscillations only, it avoids differential equation modeling of the electric network, which enhances the computational speed of the dynamic simulations. Thus, TPDA has some advantages over existing dynamic simulation software for analyzing networks with imbalance. The increasing penetration of Distributed Generation (DG) further pushes for such a technique.

This thesis begins by briefly discussing the algorithm underlying TPDA, providing a methodology overview and an explanation of the individual contributions towards the algorithm. In addition, this thesis also includes a complete object oriented implementation of TPDA and a Forward Euler based numerical integration routine in C++ which is added as an alternative to the existing MATLAB interface for solving differential equations. Further, TPDA is shown to accurately simulate power system dynamics for balanced electric networks by comparing its results against an industry standard power system dynamic analysis tool based on the positive sequence model assumption (GE-PSLF®). The work here seeks to investigate errors that result due to traditional simplifying assumptions like positive network model assumption, and yet produces an analysis that can be solved rapidly while at the same time maintaining accuracy. The advantages of using TPDA over software that use the positive sequence model of the electric network are highlighted through a comparative study of the dynamic analysis performed on an unbalanced network with TPDA and GE-PSLF®. Finally, the last chapter of this thesis focusses on studying the impact of solar PV based DG on the electromechanical oscillations in the electric grid using TPDA.

Chapter 2: Three-Phase Dynamics Analyzer (TPDA)

2.1 Assumptions and Overview

In the previous chapter the traditional approach to dynamic analysis was addressed along with the need to look beyond the traditional approach. Published work evaluating the impact of DG on the electric grid was also reviewed with the intent of establishing the uniqueness and goal of this thesis work. This chapter is dedicated to understanding the concepts behind TPDA [16]. Further, a brief description is provided on the object oriented implementation of TPDA in C++/C#. TPDA is based on two key assumptions:

- The electric network is assumed to be in steady state and modeled as a set of algebraic equations instead of differential equations, allowing fast phasor analysis of the network.
- The system does not undergo a large frequency deviation from its nominal value of 60 Hz.

The first assumption noted above is premised on the argument that electromagnetic transients associated with the electric network decay much faster than do the electromechanical transients associated with its machines. This argument is supported by the fact that the electromagnetic transients decay with time constants in the order of milliseconds, while electromechanical transients have time constants in the order of seconds. So it is assumed that by the time large electro-mechanical machines react to any change in the system, the electric network has already reached its steady state. To better understand this assumption consider Fig. 2.1. Assume that the machine states and power flow results at time t_1 are all known. At this time instant the system is exposed to a disturbance. Now the machines in the system start undergoing dynamics as soon as they are subjected to this sudden change. These dynamics are modeled in the form of differential equations for every machine. Assume that it takes time Δt (based on the time constants of these differential equations representing the machines) for the machine states to undergo any significant change. So, assumption 1 states that this Δt is large with respect to the δt time that the electric network would take in order to undergo its transients and reach steady state.

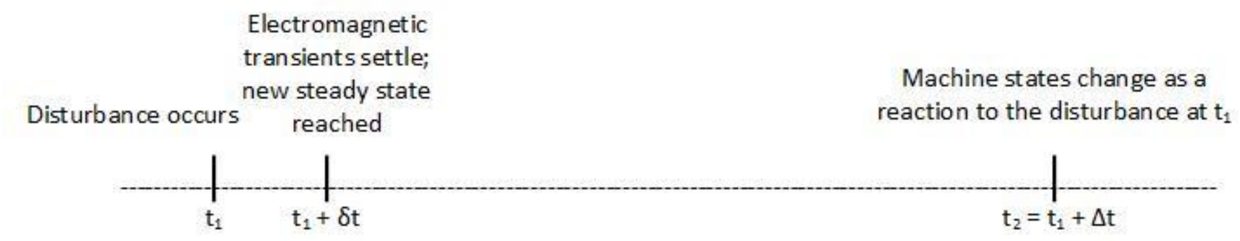
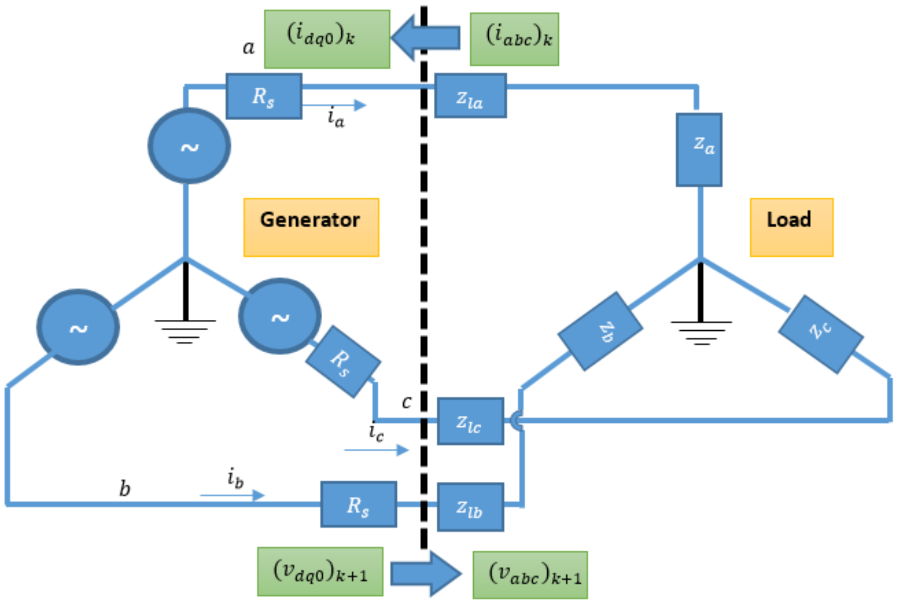

Electromagnetic transients settle; new steady state reached
Disturbance occurs
Machine states change as a reaction to the disturbance at t_1
t_1
$t_1 + \delta t$
$t_2 = t_1 + \Delta t$

Figure 2.1: Assumption 1

The second assumption listed above allows writing equations for voltages at a single frequency, which is 60 Hz. This assumption is considered to be valid for most dynamic simulations [18].

Thus, for a disturbance occurring at t1 machine states are assumed to be the same from time t1 to t1+ δt. Since the scope of the work presented here is electro-mechanical dynamic analysis, we ignore the electro-magnetic transients that the system goes through between t1 and t1+ δt. This allows modeling power flow equations algebraically and adopting phasor analysis of the electric network. So, as soon as a disturbance occurs in the system at time t1, power flow is solved, capturing the steady state network response to this disturbance. Essentially power flow results at the machine terminals have changed instantly at t1. Next, machine differential equations are solved from time t1 to t2 = t1+Δt, representing the machine's response to the disturbance. Subsequently these results serve as an input to power flow analysis, which calculates new results instantly at time t2. Again, the machines react to these changed results from t2 to t3 and the back-and-forth process continues. Such a method that involves separately solving differential and algebraic equations has been referred to as a "Partitioned Explicit" method [19].

Figure 2.2 explains this process for a single machine system. TPDA begins by solving power flow and calculating currents at the generator terminals. So, if there was a disturbance in the system, this disturbance was captured by results for generator terminal currents. Since TPDA aims at providing a multi-phase analysis, it employs a multi-phase power flow technique based on graph trace analysis [17]. These ABC current values are then transformed to dq0 values. Park's transformation into dq0 frame simplifies dynamic machine modeling. Using these dq0 currents, next step voltages at the machine terminals are determined by numerically integrating machine differential equations. Subsequently, the inverse Park transformation is used to convert back to the phasor domain so that power flow analysis can be executed to determine the currents for the next integration step. Current phasors flowing through the electric network remain unchanged until new voltages are found by integrating the machine dynamic equations, and based on assumption 1, these currents change instantly at the beginning of the new time step.



z_a, z_b, z_c: Load impedance for the three phases

z_{la}, z_{lb}, z_{lc}: Line impedance for the three phases

$(v_{abc})_{k+1}$: Updated three phase voltage at generator terminal

$(i_{abc})_k$: Three-phase current vector used for obtaining $(v_{abc})_{k+1}$

Figure 2.2: Flowchart showing the “Partitioned Explicit” method

2.2 ABC → DQ0 → ABC

Dynamic modeling for machines is greatly simplified by transforming from the static *abc* frame to the rotating *dq0* frame. Figure 2.3 [20] shows these frames of reference.

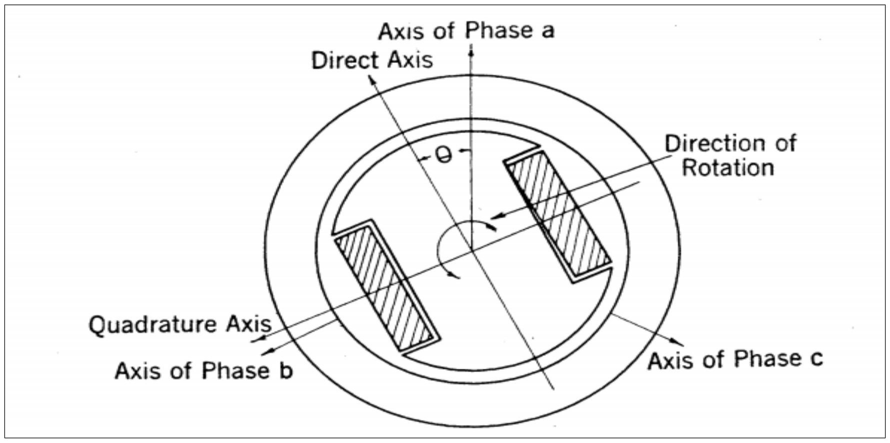


Figure 2.3: abc-dqo frames [20] R. H. Park, "Two-reaction theory of synchronous machines generalized method of analysis-part I", IEEE Trans. of the AIEE, vol. 48, no. 3, pp. 716-727, July 1929. Used under fair use, 2015.

Such a change in the reference frame can be mathematically expressed as:

$$\begin{bmatrix} V_d \\ V_q \\ V_0 \end{bmatrix} = S \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \dots (1)$$

where

$$S = \left(\frac{2}{3}\right) \begin{bmatrix} \sin\,(\omega_r t) & \sin\,(\omega_r t - 2\pi/3) & \sin\,(\omega_r t + 2\pi/3) \\ \cos\,(\omega_r t) & \cos\,(\omega_r t - 2\pi/3) & \cos\,(\omega_r t + 2\pi/3) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \dots (2)$$

$$\text{where } \omega_r t = \ \omega_s t + \ \delta = \left(\frac{P}{2}\right) * \theta_{shaft} \dots (3)$$

ω_s is the synchronous electrical speed in radians per second and δ is the rotor angle that is constant for constant shaft speed.

Figure 2.4 shows the variables known at time instant t_1. At t_1, generator terminal voltages are known since they were calculated during the previous iteration of TPDA from t_0 to t_1. Additionally, the machine states at t_1 calculated during the previous iteration from t_0 to t_1 are also known. Using power flow analysis, the new a-b-c currents that have been affected by the disturbance in the system at t_1 are calculated. Then equation (1) is used to find dq0 values for voltages and currents.

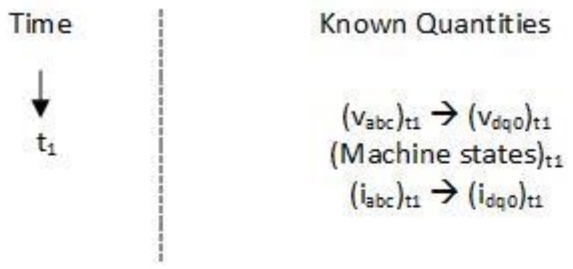

Time
t_1
Known Quantities
$(v_{abc})_{t1} \rightarrow (v_{dq0})_{t1}$
(Machine states)$_{t1}$
$(i_{abc})_{t1} \rightarrow (i_{dq0})_{t1}$

Figure 2.4: Known quantities at time t_1

Using machine states at time t_1 and $(i_{dq0})_{t1}$ the new machine states at time t_2 are calculated by integrating the machine differential equations (next section). Subsequently, the machine terminal voltages at t_2, $(v_{dqo})_{t2}$ are calculated using these newly calculated states. This is illustrated in figure 2.5.

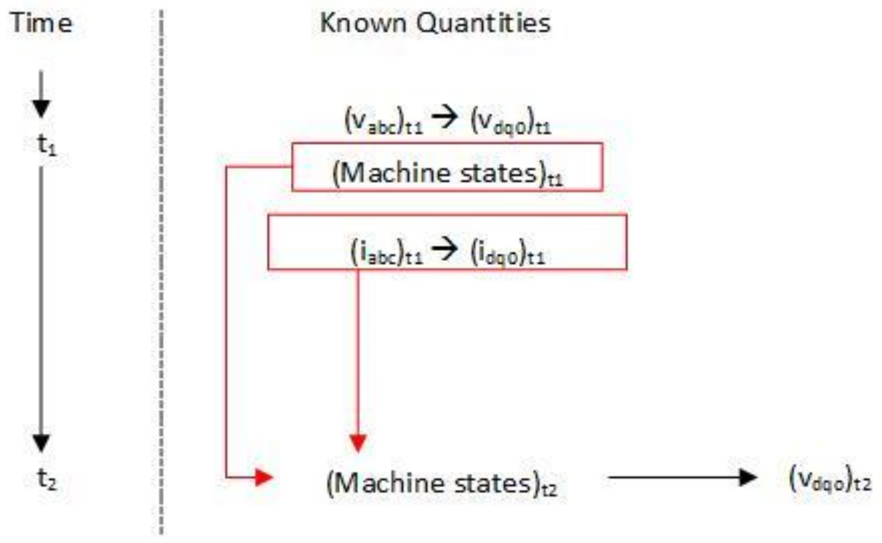


Figure 2.5: From t_1 to t_2

$(v_{dqo})_{t2}$ can then be converted to the abc frame of reference by employing the inverse transform as per equation 4. This gives the instantaneous value of machine terminal voltages at time t_2. However, the multi-phase power flow technique being used employs phasor analysis, and thus these voltages at the next time step need to be converted to the phasor domain. In order to convert to the phasor domain, the instantaneous voltages at at-least two different time instants are needed so that the new voltage waveform can be estimated and the phasors calculated. In other words, each phasor has two unknowns, magnitude and phase angle and therefore, estimating three such phasors (for each of phase A, B and C) represents six unknowns. But equation 4 represents only three equations. So, three more equations are needed that can be used to estimate the voltages phasors at time t_2. So the idea is to divide the time $t_2 - t_1 = \Delta t$ into two parts (figure 2.6), one from t_1 to (t_2 - δ_1t) and the other from (t_2 - δ_1t) to t_2. Here δ_1t represents a small time so that machine states do not change appreciably from (t_2 - δ_1t) to t_2.

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = S^{-1} \begin{bmatrix} V_d \\ V_q \\ V_0 \end{bmatrix} \ldots (4)$$

where

$$S^{-1} = \begin{bmatrix} \sin(\omega_r) & \cos(\omega_r) & 1 \\ \sin(\omega_r - 2\pi/3) & \cos(\omega_r - 2\pi/3) & 1 \\ \sin(\omega_r + 2\pi/3) & \cos(\omega_r + 2\pi/3) & 1 \end{bmatrix} \ldots (5)$$

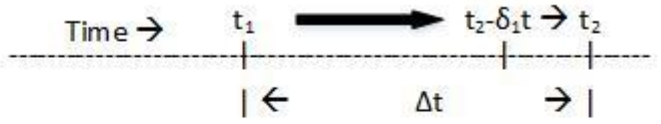


Figure 2.6: Illustrating time points for calculating phase voltages twice in Δt, providing information needed to calculate phasors at time t_2

Now, with the time Δt divided into two parts, the method explained previously using figure 2.5 is implemented twice between t1 and t2 (without exchanging information with power flow). This new process is outlined in figure 2.7. First, v_{dq0} is calculated at t_2-δ_1t as per the method illustrated in figure 2.5. Since exchange of results only takes place at t_2, the current phasor is the same till t_2 (assumption 1). Therefore, the same current phasor from time t_1 is used to compute i_{dq0} at t_2-δ_1t. Having found (machine states)$_{t2-\delta1t}$ and (i_{dq0})$_{t2-\delta1t}$, these values are used to compute machine states at t_2 and subsequently, v_{dq0} at t_2. Since δ_1t is very small, the change in machine states from t_2-δ_1t to t_2 is negligible and ignored. Thus v_{dq0} values have been obtained twice within the time interval δ_1t. However, machine states were changed only once from t_1 to t_2 (same as t_2-δ_1t for the slow machines).

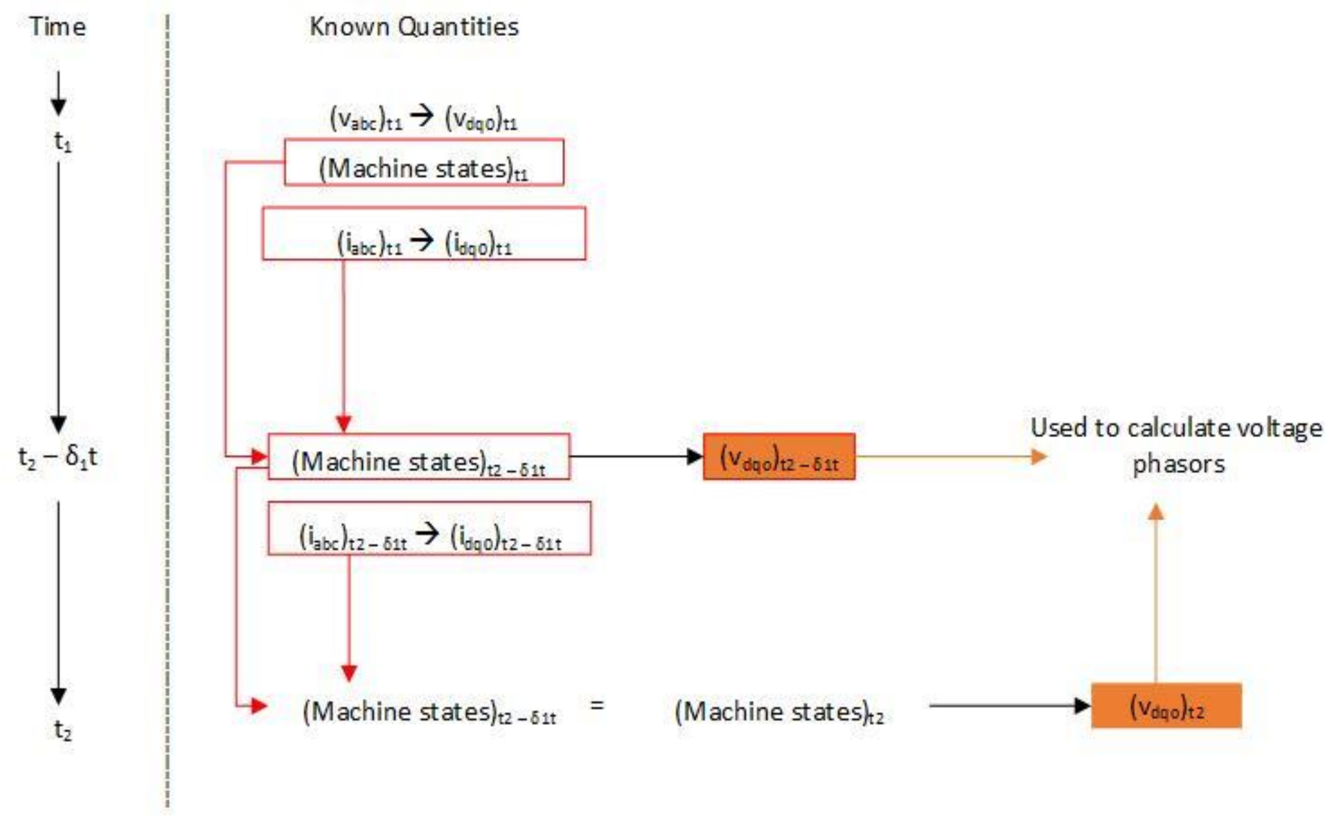


Figure 2.7: Timeline showing progress of TPDA

Thus, for the two sets of d-q-0 voltages, equation (4) is used to develop six equations in six unknowns. These equations are then solved to find the voltage phasors at machine terminals [16]. The equations used to calculate instantaneous voltage values are shown below in equations 6, 7, 8 and 9.

$$\mathbb{V}_{abc} = \mathbb{S}_{new}^{-1} * \mathbb{V}_{dq0} \ldots (6)$$

$$\mathbb{V}_{abc} = \begin{bmatrix} V_a e^{j\theta_a} \\ V_b e^{j\theta_b} \\ V_c e^{j\theta_c} \\ V_a e^{j\theta_a} \\ V_b e^{j\theta_b} \\ V_c e^{j\theta_c} \end{bmatrix} \ldots (7)$$

$$\mathbb{V}_{dq0} = \begin{bmatrix} (V_d)_{t2-\delta_1 t} \\ (V_q)_{t2-\delta_1 t} \\ (V_0)_{t2-\delta_1 t} \\ (V_d)_{t2} \\ (V_q)_{t2} \\ (V_0)_{t2} \end{bmatrix} \dots (8)$$

$$\mathbb{S}_{new}^{-1} = \begin{bmatrix} \sin(\omega_r t') & \cos(\omega_r t') & 1 & 0 & 0 & 0 \\ \sin(\omega_r t' - 2\pi/3) & \cos(\omega_r t' - 2\pi/3) & 1 & 0 & 0 & 0 \\ \sin(\omega_r t' + 2\pi/3) & \cos(\omega_r t' + 2\pi/3) & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sin(\omega_r t_2) & \cos(\omega_r t_2) & 1 \\ 0 & 0 & 0 & \sin\left(\omega_r t_2 - \frac{2\pi}{3}\right) & \cos\left(\omega_r t_2 - \frac{2\pi}{3}\right) & 1 \\ 0 & 0 & 0 & \sin\left(\omega_r t_2 + \frac{2\pi}{3}\right) & \cos\left(\omega_r t_2 + \frac{2\pi}{3}\right) & 1 \end{bmatrix}$$

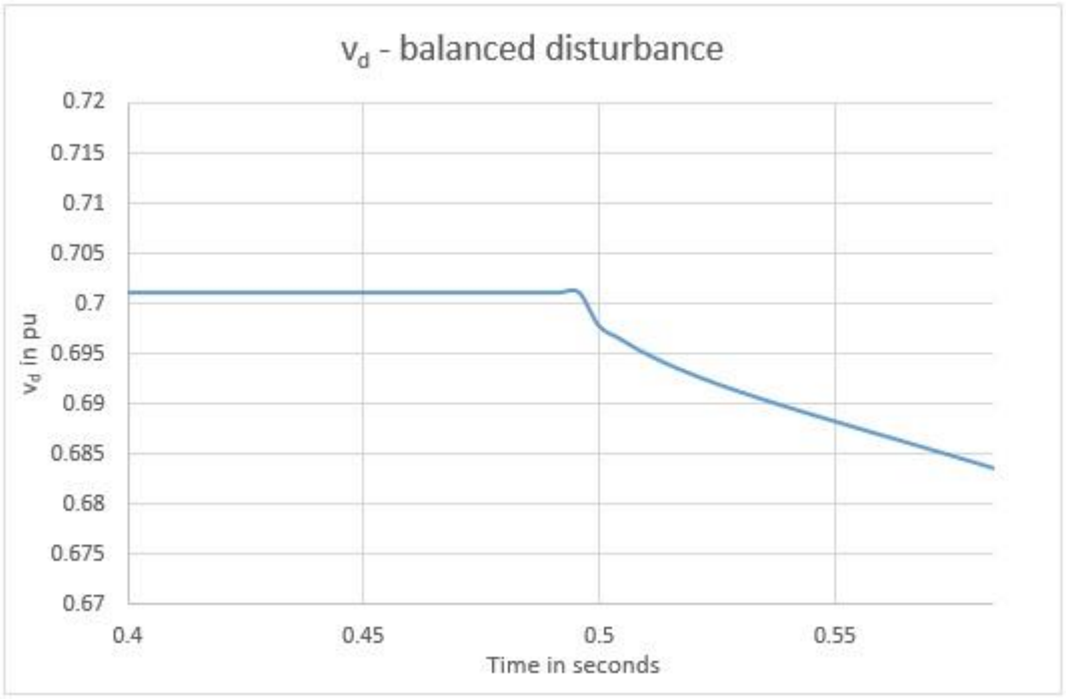
…. (9)

Where $t' = t_2 - \delta_1 t$

It must be noted that appropriate base values were used to convert all voltages and currents to per unit values. This enabled avoiding the use of very large and very small quantities in calculations.

2.2.1 How TPDA Captures Imbalance

It is known that under balanced system conditions, the d-q-0 currents and voltages have a fixed constant value. Hypothetically, let v_d = 0.7011 (as shown in figures 2.8 and 2.9 below). If there is no change in the system, these quantities remain fixed at their original values. Now, assume the system experiences a disturbance at time t_1 = 0.5 seconds. There are two possibilities: it is a balanced disturbance, or it is an unbalanced disturbance. If the change occurring in the system is balanced, these d-q-0 currents and voltages change without undergoing any 120 Hz oscillations. However, if the change is unbalanced, these d-q-0 currents and voltages undergo a change along with 120 Hz oscillations. Figures 2.8 and 2.9 plot v_d for balanced and unbalanced disturbances occurring in the system at time t = 0.5 seconds.



/

Figure 2.8: No 120 Hz oscillations in v_d for a balanced disturbance

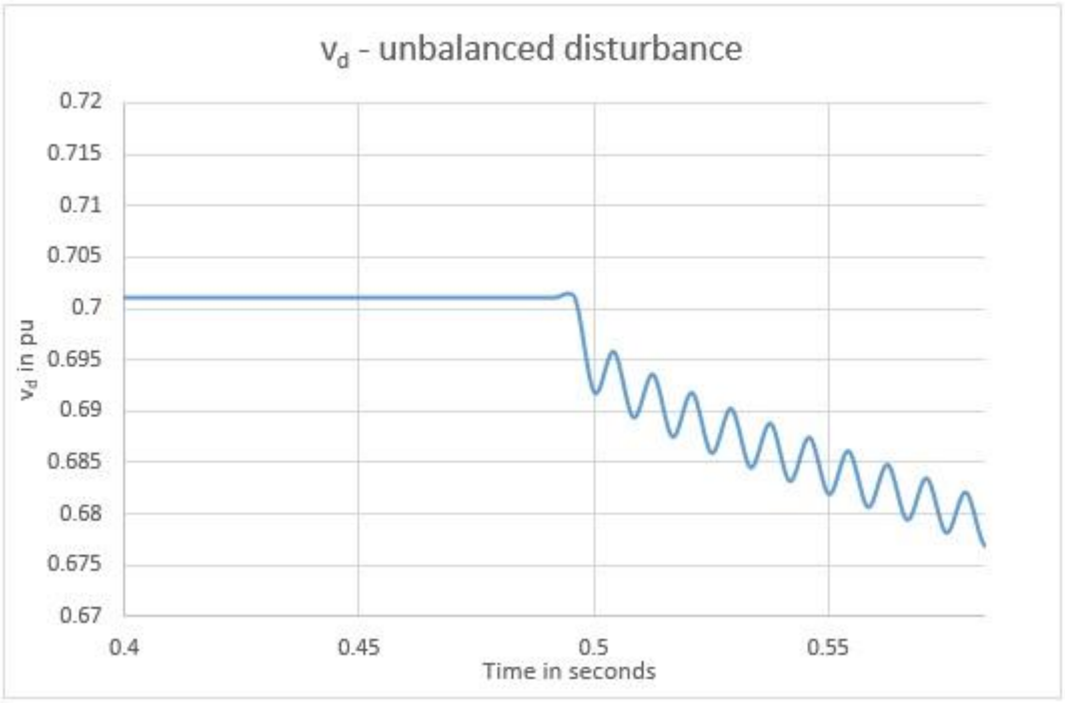


Figure 2.8: 120 Hz oscillations in v_d for an unbalanced disturbance

Let us now examine how TPDA captures system imbalance and distinguishes balanced and unbalanced disturbances. To do so, let's examine the method explained using figure 2.7 under both balanced and unbalanced scenarios.

Balanced disturbance at t_1

If the disturbance experienced by the system at time t_1 is balanced, multiphase power flow captures this change in the system and calculates the new balanced currents at t_1. When these currents are transformed to the d-q-0 frame, the d-q-0 quantities jump from their original steady state value to a new value (no 120 Hz oscillations at the new value at t_1). Using this $(i_{dq0})_{t1}$, new machine states are computed along with d-q-0 voltages at $(t_2-\delta_1 t)$. This captures the change in v_{dq0} values. It must be noted that since the change is balanced, there are no 120 Hz oscillations in i_{dq0} and since there is no exchange of results with power flow until we reach time t_2, i_{dq0} remains unchanged from t_1 to $(t_2-\delta_1 t)$. Further, machine states also remain unchanged from $(t_2-\delta_1 t)$ to t_2. Thus, $(v_{dq0})_{t2}$ which is a function of i_{dq0} at $(t_2-\delta_1 t)$ and machine states at t_2, is also unchanged (no 120 Hz oscillations). Hence, the two sets of d-q-0 voltages used to calculate the new phasor at t_2 are the same (jump to a new value and hold constant there without any 120 Hz oscillations). In this manner, a balanced change is successfully captured. This explanation is summarized in table 2.1, which shows if i_{dq0} and machine states change during each iteration between t_1 and t_2, and the effect this has on v_{dq0}.

Unbalanced disturbance at t_1

If the disturbance occurring at t_1 is unbalanced, this imbalance is captured by the a-b-c current computed using the multi-phase power flow. When this current is transformed into d-q-0 frame, $(i_{dq0})_{t1}$, 120 Hz oscillations result. Using this $(i_{dq0})_{t1}$, the new machine states and v_{dq0} at $t_2-\delta_1 t$ are computed. This captures the change in v_{dq0} values. Now, in the process of calculating $(v_{dq0})_{t2}$, i_{dq0} is again computed at $(t_2-\delta_1 t)$. It must be noted here that since the change in the system is unbalanced, i_{dq0} oscillates, which in turn results in changed i_{dq0} value at $(t_2-\delta_1 t)$. However, machine states don't change from $(t_2-\delta_1 t)$ to t_2. Therefore, v_{dqo} at t_2 which is a function of both i_{dq0} at $(t_2-\delta_1 t)$ and machine states at t_2, undergoes a change (120 Hz oscillations). So the two sets of d-q-0 voltages used to calculate the voltage phasor at t_2 change from time $t_2-\delta_1 t$ to t_2 (jump to a new value and undergo 120 Hz oscillations). In this manner TPDA is successful in capturing the d-q-0 oscillations in currents and voltages. (Results for test cases in chapter 4 show these 120 Hz oscillations). This explanation is summarized in table 2.2 which shows if i_{dq0} and machine states change during each iteration between t_1 and t_2, and the effect this has on v_{dq0}.

Table 2.1: Table showing how TPDA captures a balanced disturbance

Balanced Change at t_1	i_{dq0}	Machine States	v_{dqo}
Iteration # 1	Change	Change	Change (Jump to a new value)
Iteration # 2	No Change	No Change	No Change

Table 2.2: Table showing how TPDA captures an unbalanced disturbance

Unbalanced Change at t_1	i_{dq0}	Machine States	v_{dqo}
Iteration # 1	Change	Change	Change (Jump to a new value)
Iteration # 2	Change (120 Hz Oscillations)	No Change	Change (120 Hz Oscillations)

2.3 Dynamic Machine Modeling

This section briefly discusses the power plant equipment models that are included in TPDA. While the modeling of these equipment was not performed as part of this thesis and was already included in TPDA, it is important to explain these models as the provided explanations can greatly help in understanding the dynamic analysis results that are presented later. Only those models included in TPDA are explained here that were used in research leading to this thesis. These models include Genrou (round rotor synchronous machine model), IEEE (2005) type AC7B excitation system and general governor model (GGOV1).

2.3.1 Generator Model

The generator is modeled in TPDA using the synchronous machine model described in [19]. This synchronous machine model is equivalent to the standard solid round rotor generator model. This is also referred to as the “Genrou” model in various texts. Equations 10-15 are the six state equations representing the synchronous machine model of [19]. All symbols used in equations 10-18 have been defined in Appendix A. Table 2.1 gives a one line description for each of these state equations.

$$T'_{d0}\frac{dE'_q}{dt} = -E'_q - (X_d - X'_d)\left[I_d - \frac{X'_d - X''_d}{(X'_d - X_{ls})^2}\left(\psi_{1d} + (X'_d - X_{ls})I_d - E'_q + S_{1d}\right)\right] - S_{fd} + E_{fd} \dots (10)$$

$$T''_{d0}\frac{d\psi_{1d}}{dt} = -\psi_{1d} + E'_q - (X'_d - X_{ls})I_d - S_{1d} \dots (11)$$

$$T'_{q0}\frac{dE'_d}{dt} = -E'_d + \left(X_q - X'_q\right)\left[I_q - \frac{X'_q - X''_q}{\left(X'_q - X_{ls}\right)^2}\left(\psi_{2q} + \left(X'_q - X_{ls}\right)I_q + E'_d + S_{2q}\right)\right] + S_{1q} \dots (12)$$

$$T''_{q0}\frac{d\psi_{2q}}{dt} = -\psi_{2q} - E'_d - \left(X'_q - X_{ls}\right)I_q - S_{2q} \dots (13)$$

$$\frac{d\delta}{dt} = \omega - \omega_s \dots (14)$$

$$\frac{2H}{\omega_s}\frac{d\omega}{dt} = T_M - \left(\psi_d I_q - \psi_q I_d\right) - T_{FW} \dots (15)$$

Table 2.3: Synchronous machine model state equation description

State Equation #	**Description**
10	Field Circuit state equation
11	d-axis damper winding circuit state equation
12	1st q-axis damper winding circuit state equation
13	2nd q-axis damper winding circuit state equation
14	Rotor angle state equation
15	Rotor speed state equation

In TPDA, these generator state equations are numerically integrated choosing an appropriate time step in order to obtain the next step generator states. Now, using the newly calculated states and i_{dqo} values, the following equations are used to compute the next step v_{dq0} values:

$$V_d = -X_q''I_q + D_1E_d' - E_1\psi_{2q} - R_sI_d \ldots (16)$$

$$V_q = -X_d''I_d + DE_q' + E\psi_{1d} - R_sI_q \ldots (17)$$

$$V_0 = -R_sI_0 \ldots (18)$$

In order to use the above described generator model, we need certain generator parameters as input. Table 2.2 describes these input parameter requirements.

Table 2.4: Generator model input parameter description

Input Variable	**Description**
T_{d0}'	d-axis transient rotor time constant
T_{d0}''	d-axis sub-transient rotor time constant
T_{q0}'	q-axis transient rotor time constant
T_{q0}''	q-axis sub-transient rotor time constant
H	Inertia constant
D	Damping factor
X_d	d-axis synchronous reactance
X_q	q-axis synchronous reactance
X_d'	d-axis transient reactance
X_q'	q-axis transient reactance
X_d''	d-axis sub-transient reactance
X_q''	q-axis sub-transient reactance
X_{ls}	Stator leakage reactance
$S(1.0)$	Saturation factor at 1 p.u. flux
$S(1.2)$	Saturation factor at 1.2 p.u. flux
R_s	Stator resistance
R_{comp}	Compounding resistance for voltage control

X_{comp}	Compounding reactance for voltage control

2.3.2 Exciter and Governor Models

IEEE (2005) type AC7B excitation system and the general governor model (GGOV1) are implemented in TPDA using the control diagrams shown in figures 2.10 [21] and 2.11 [22]. These model implementations require a number of input parameters which are listed in appendix A.

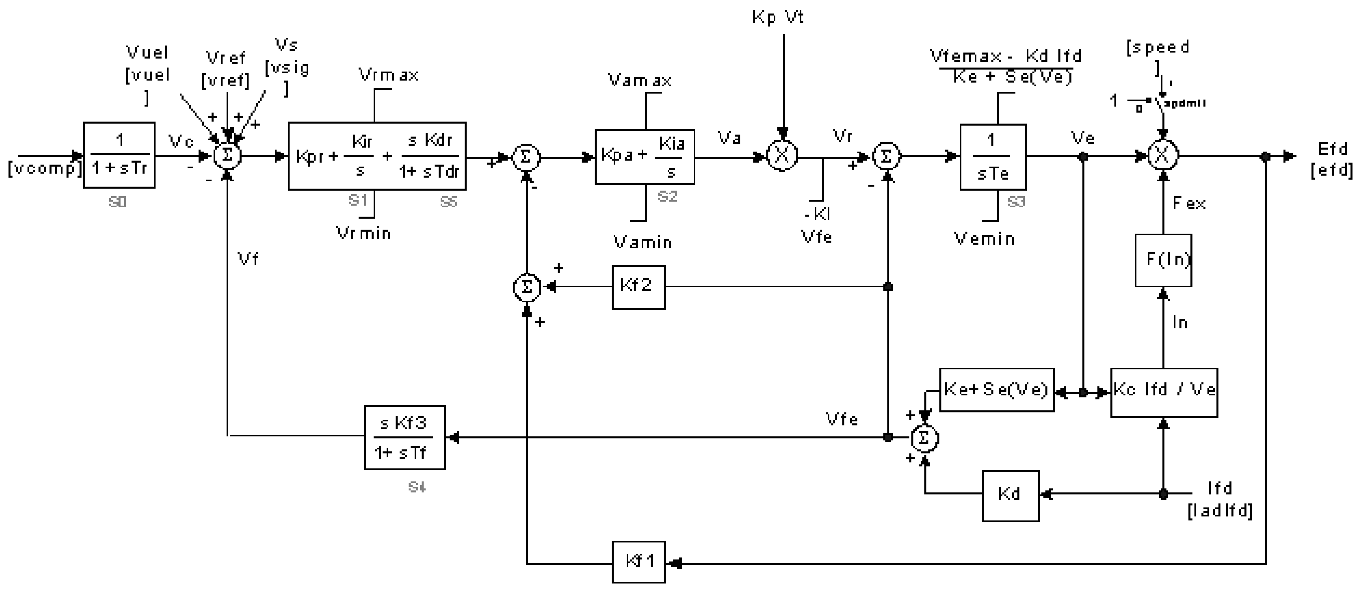


Figure 2.10: Control diagram representing AC7B excitation system [21] Components, Circuits, Devices & Systems | Power, Energy, & Industry Applications, "IEEE Standard Definitions for Excitation Systems for Synchronous Machines", IEEE Std. 421.1.2007, July 2007. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=5981340 , [22] GE Energy's Positive Sequence Load Flow (PSLF) Software and User Manual, http://site.ge-energy.com/prod_serv/products/utility_software/en/ge_pslf/index.htm. Used under fair use, 2015.

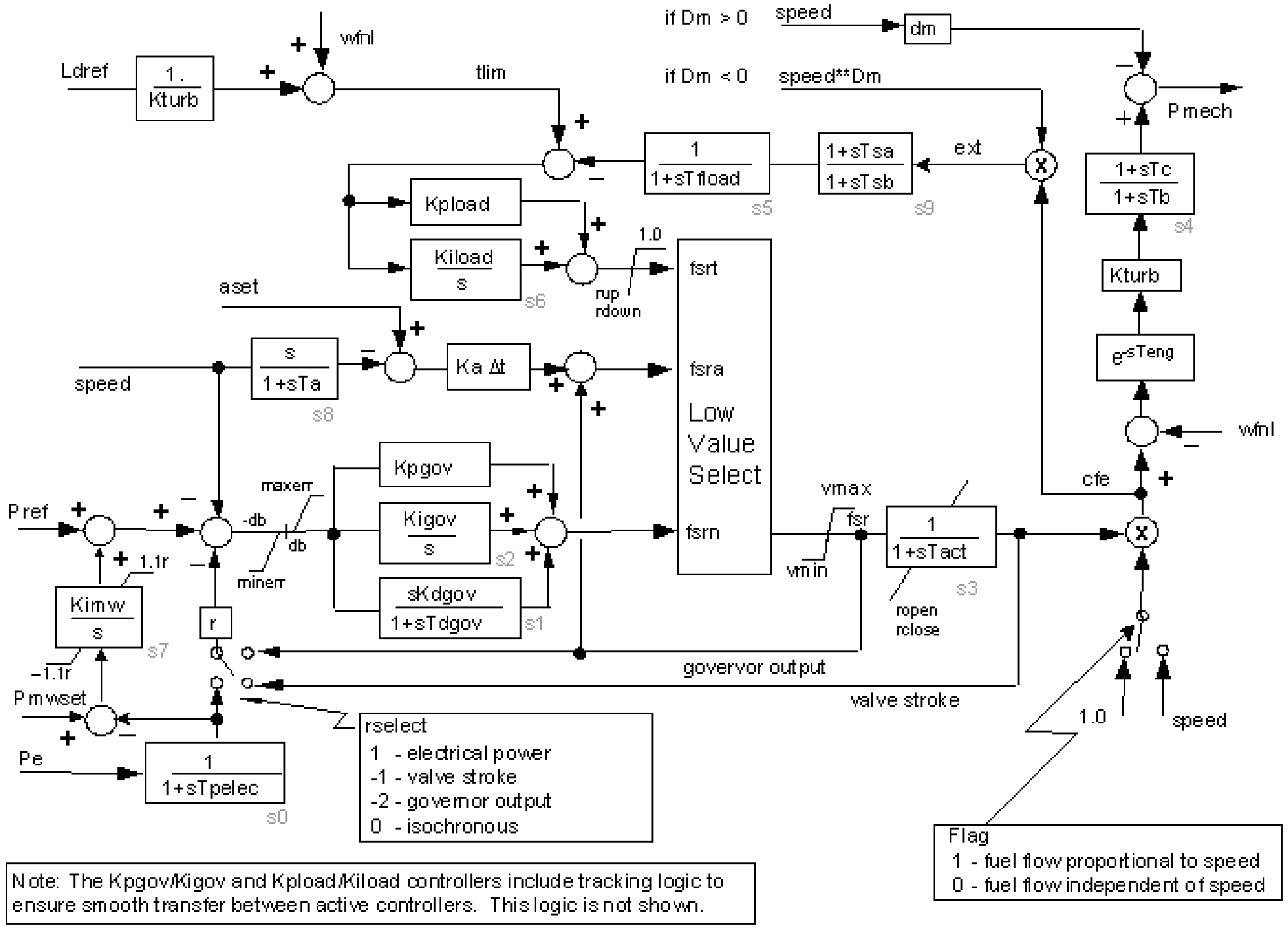
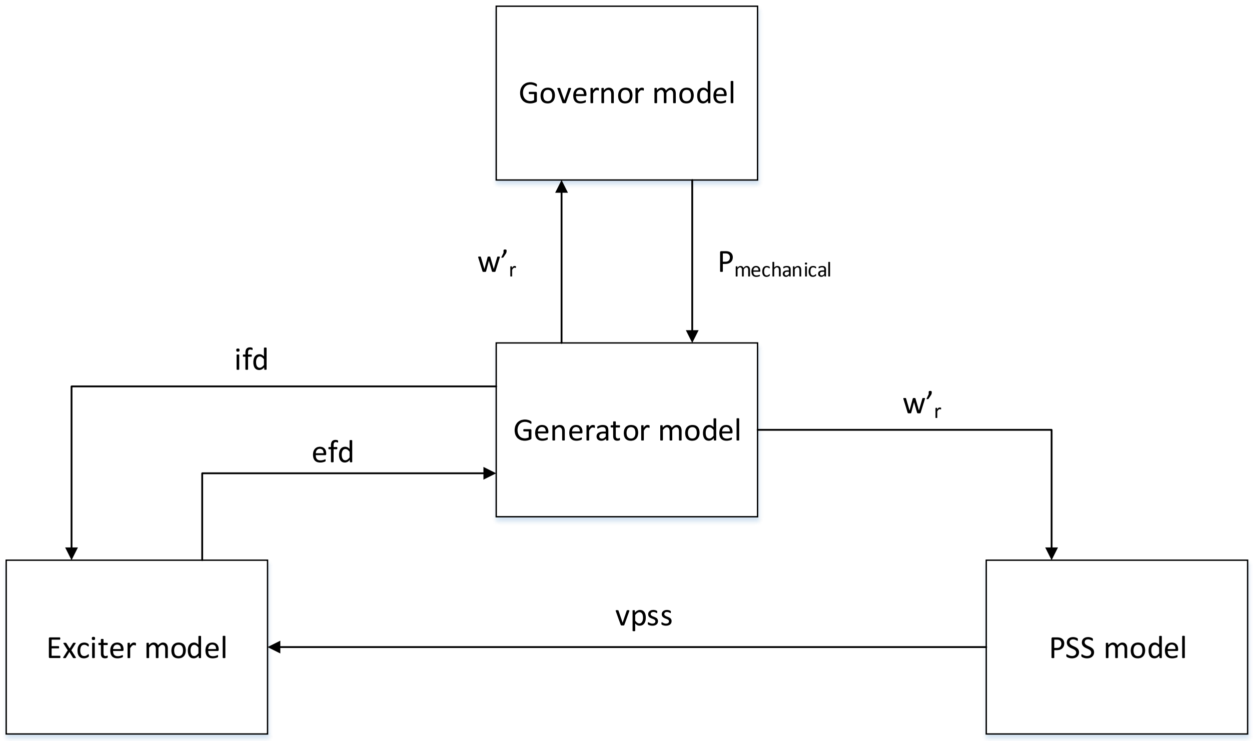


Figure 2.11: Control diagram representing GGOV1 governor model [22] GE Energy's Positive Sequence Load Flow (PSLF) Software and User Manual, http://site.ge-energy.com/prod_serv/products/utility_software/en/ge_pslf/index.htm. Used under fair use, 2015.

In TPDA, the models described above are solved together for each machine in the system. Figure 2.13 shows how the generator, exciter, governor and power system stabilizer interact with each other. In the absence of a power system stabilizer, the power system stabilizer output is set to zero. In the absence of a governor, $P_{mechanical}$ (i.e., the turbine's mechanical output) is set to its initialized value equal to the electrical power demand under steady state conditions. Finally, in the absence of an exciter, the efd (i.e., the generator field voltage) is set to its initialized value as computed by the generator model. Initialization is achieved by equating the differential equations to zero.



w'_r: actual rotor speed in radians per second/synchronous rotor speed in radians per second

vpss: power system stabilizer output

efd: generator field voltage

ifd: generator field current

$P_{mechanical}$: Turbine mechanical power

Figure 2.12: Interaction between generator, exciter, PSS and governor

2.4 Object Oriented Implementation of TPDA

The flow chart of figure 2.13 shows a generalized implementation of TPDA. It begins by loading the electric network model, reading machine parameters for their dynamic models and running multi-phase power flow analysis [17]. The electric network is modelled in, and read from DEW (Distributed Engineering Workstation). The machine parameters are read from text files. This loading of data is done only during the first iteration. Subsequently, every other iteration begins by running power flow analysis. Using power flow results, all machines in the system are initialized during the first iteration. Network currents calculated by power flow are converted to the d-q-0 frame and subsequently machine dynamics are solved, computing the machine terminal voltages, as explained in the previous sections. Each machine is solved separately, allowing for the potential use of distributed computing in the future. The methodology explained in section 2.2 is represented by the “machine dynamics” block of figure 2.13.

Once dynamic models are solved for all machines and their terminal voltages computed, power flow analysis is executed again and the process is repeated. The time step used for this explicit partitioned method is 1/ (60*N) seconds, where N can be varied to achieve the desired balance between accuracy, speed and numerical stability.

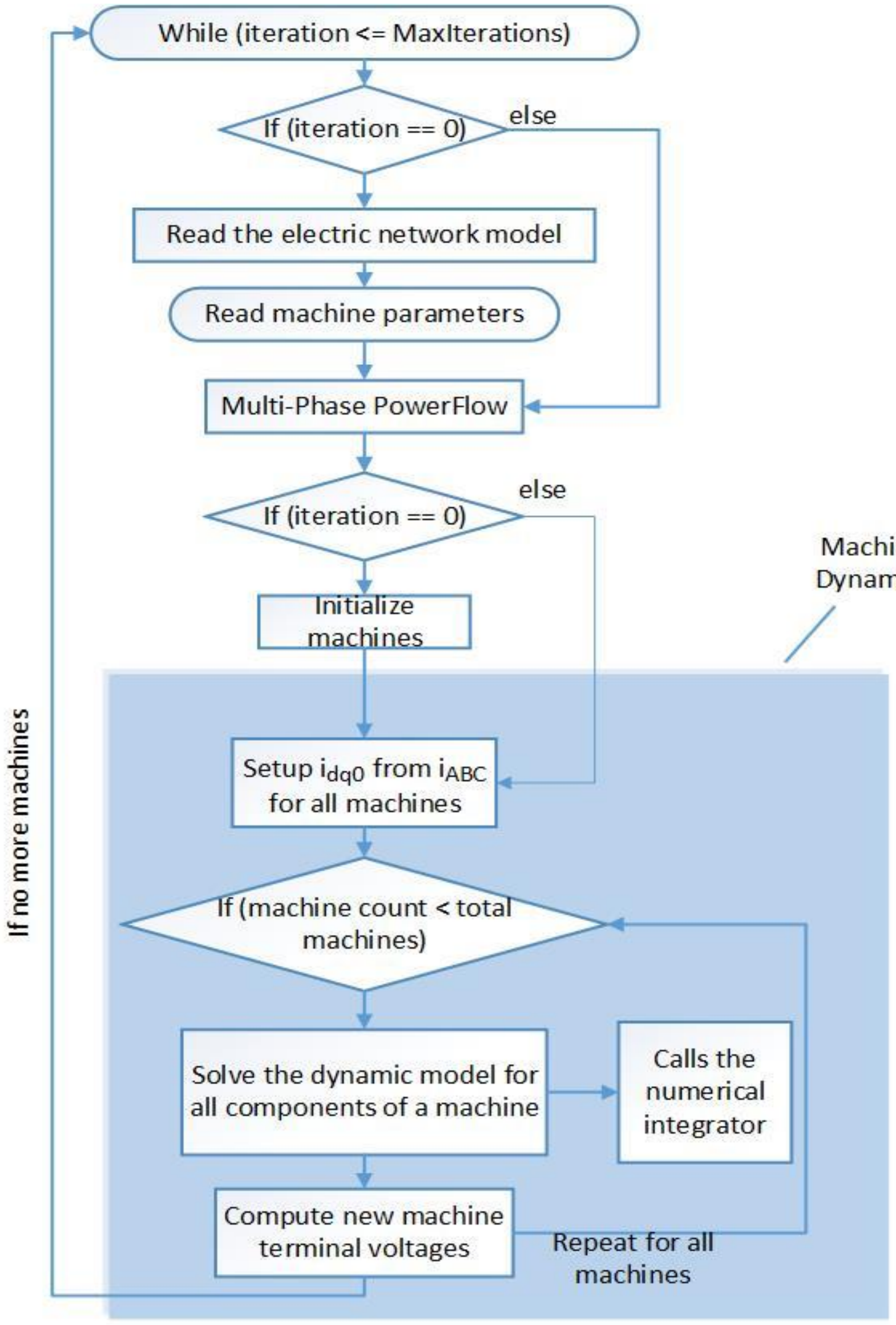

While (iteration <= MaxIterations)
If (iteration == 0)
else
Read the electric network model
Read machine parameters
Multi-Phase PowerFlow
If (iteration == 0)
else
Machine Dynamics
Initialize machines
Setup i_{dq0} from i_{ABC} for all machines
If (machine count < total machines)
Solve the dynamic model for all components of a machine
Calls the numerical integrator
Compute new machine terminal voltages
Repeat for all machines
If no more machines

Figure 2.13: Flowchart of TPDA

In this thesis, the TPDA algorithm was implemented in C++/C# using Microsoft Visual Studio environment. An object oriented approach that employed inheritance was used. Parent class definitions for generators, exciters, governors and power system stabilizers were implemented. Each of these parent classes extend various machine models that could constitute our system. For example, the generator parent class extends two types of generator models at present: “round rotor synchronous machine model” and the “two-axis model”, while the exciter parent class extends the “IEEE (2005) type AC7B excitation system”. Such an implementation allows for addition of more types of machine models (as per need) in the future, without affecting the rest of the implementation.

An Euler based numerical integrator was implemented in C++ for integrating the differential equations. In future, it is envisaged that more robust numerical integration methods such as the implicit Euler or the implicit Runge-Kutta or 2nd order Rosenbrock formula will also become available in the C++/C# implementation of TPDA. The use of C++/C# allows for the easy distribution of the entire implementation to different processors, thus lending high efficiency to TPDA. At present, TPDA interfaces with Matlab to make use of its ODE (Ordinary Differential Equation) functions like “ode23s”, “ode23t” or “ode23tb” for numerically integrating a stiff set of machine differential equations if the Euler method shows instability.

Chapter 3: TPDA: Verification & Motivation

This chapter provides a verification of TPDA by comparing the results of dynamic simulation performed on balanced networks with GE-PSLF® [22] (which is based on positive sequence network assumption). Moreover, the advantages of using TPDA for studying electromechanical transients instead of using positive sequence network model based software are also discussed. Reference [16] can be referred to for comparison of TPDA's performance against ATP (Alternative Transients Program).

3.1 Electric Network Used

The electric network shown in Figure 3.1 is used for all subsequent analysis presented in this chapter. It is the WECC 9-bus system with the associated loads and generation. Specific changes made to the WECC 9-bus system are presented in subsequent relevant sections. Table 3.1 summarizes the three machine types (with generator, exciter and governor types) used in the simulations. All the model parameters for the machines were obtained from an actual utility.

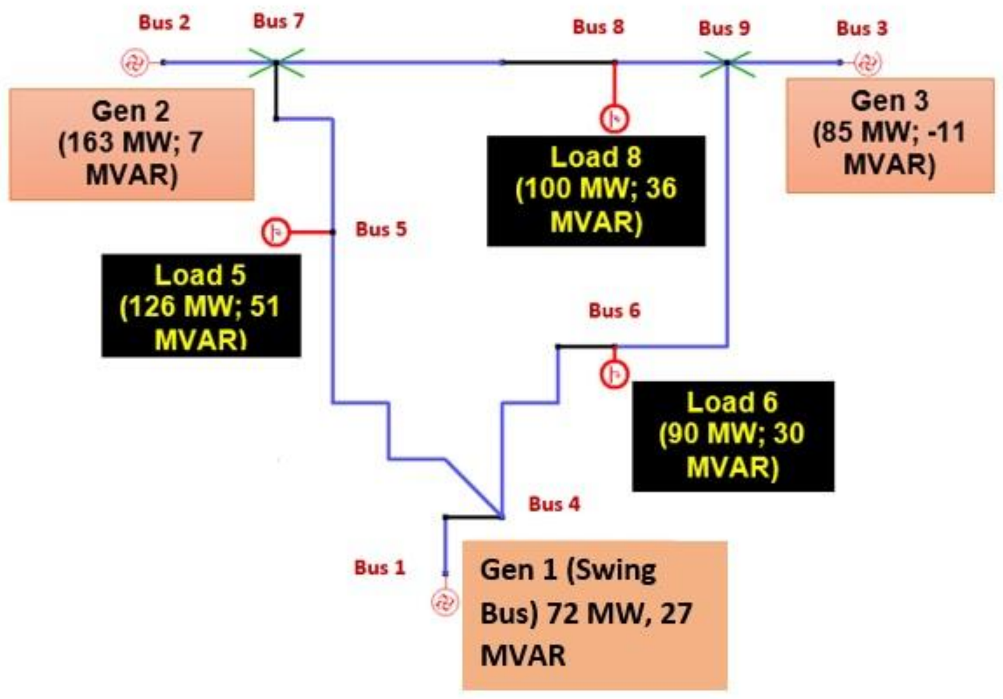


Figure 3.1: WECC 9-bus system – Circuit used for analysis

Table 3.1: Types of machines used for presenting and comparing test case results

	Generator	Exciter	Governor
Gen 1	GENROU	AC7B	GGOV1
Gen 2	GENROU	AC7B	GGOV1
Gen 3	GENROU	AC7B	-

3.2 Verification of TPDA

Power system dynamic analysis is commonly performed using the assumption of balanced power system operations, as found in the industry standard software GE-PSLF®, Positive Sequence Load Flow program [22]. Therefore, to evaluate the performance of TPDA, equivalent models were created in both DEW and PSLF. These models were then used to perform a comparative study of results obtained using TPDA and the traditional dynamic analysis module of PSLF. Positive Sequence Dynamic Analysis (abbreviated as PSDA from hereon), such as used in PSLF, assumes a balanced model for the electric network, and thus cannot be used for evaluating the proposed algorithm's performance under unbalanced system conditions.

3.2.1 Modeling in PSLF

PSLF [22] is a large-scale power system simulation software package that includes the following main modules:

- Main load flow program and working case maintenance commands
- Dynamic Analysis program and working case maintenance commands
- Short Circuit Analysis module (scsc)
- One-Line Graphic Subsystem (olgr)
- Engineering Process Control Language (epcl)
- Dynamic Result Plotting module (plot)
- Linear Network Analysis module (lina)
- Economic Dispatch module (econ)

In this thesis, the main load flow program, dynamic analysis program and working case maintenance commands along with the “olgr” and “plot” modules were used for the modeling, simulation and analysis of the desired test cases in PSLF.

Two test cases were used for the verification of TPDA. Case I used a single machine system shown in figure 3.2. A disturbance was introduced in the system at time t = 0.5 seconds in the form of a load increment. The load was increased to 42 MW, 10.5 MVAR from the base case of 40 MW, 10 MVAR; thus the load was increased maintaining a constant power factor. The total simulation time considered was 40 seconds.

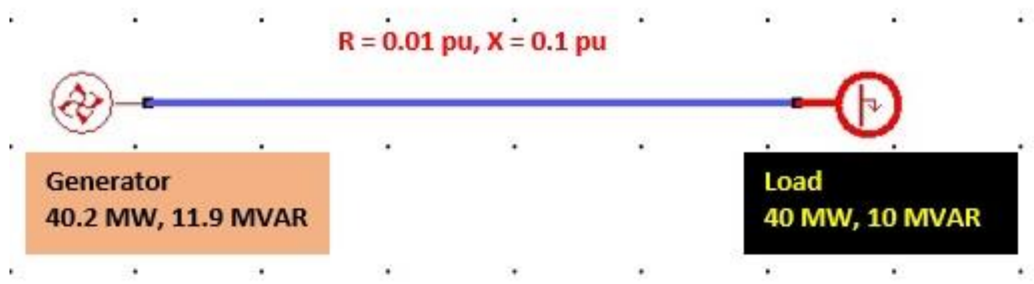


Figure 3.2: Single machine system

Case II used the WECC 9-bus test system of figure 3.1. A total simulation time of 40 seconds was used. Disturbance was introduced in the form of a 4.5 MW, 1.5 MVAR balanced load increment on load bus 6 of figure 3.1 at time t = 0.5 seconds. In both test cases, the results obtained using PSDA were compared to those obtained using TPDA.

3.2.2 Results and Analysis

Case I: Single machine system

Case I as described above was simulated using both PSDA and TPDA. Plots comparing results obtained using the two approaches are shown in figures 3.3-3.8. It is shown in figure 3.3 that all the three phase voltages are seen to move together under balanced system conditions. While figure 3.3 shows perfectly coinciding voltage curves for the three phases under balanced conditions, figure 3.4 compares the generator terminal voltage output of PSDA to the average of the three-phase voltages computed using TPDA. It is seen that the average of the three-phase voltages, also equal to the three individual phase voltages (since all of them are equal under balanced conditions) follows PSDA output very closely.

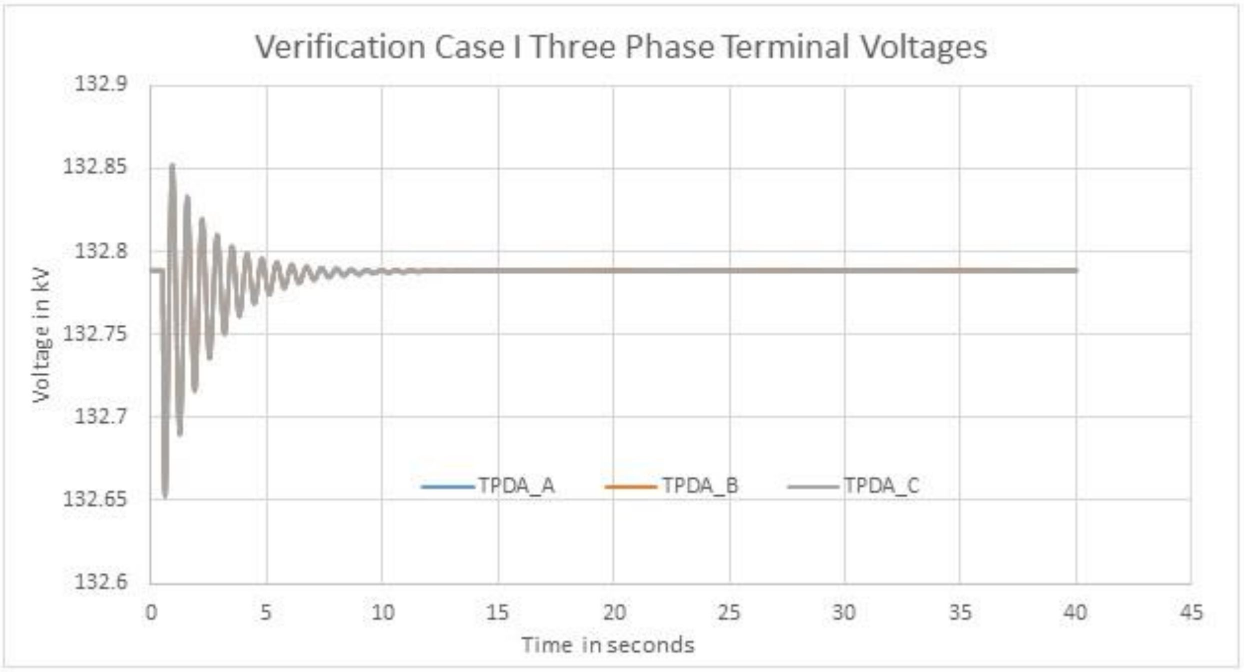


Figure 3.3: Verification Case I: Three-phase generator terminal voltages

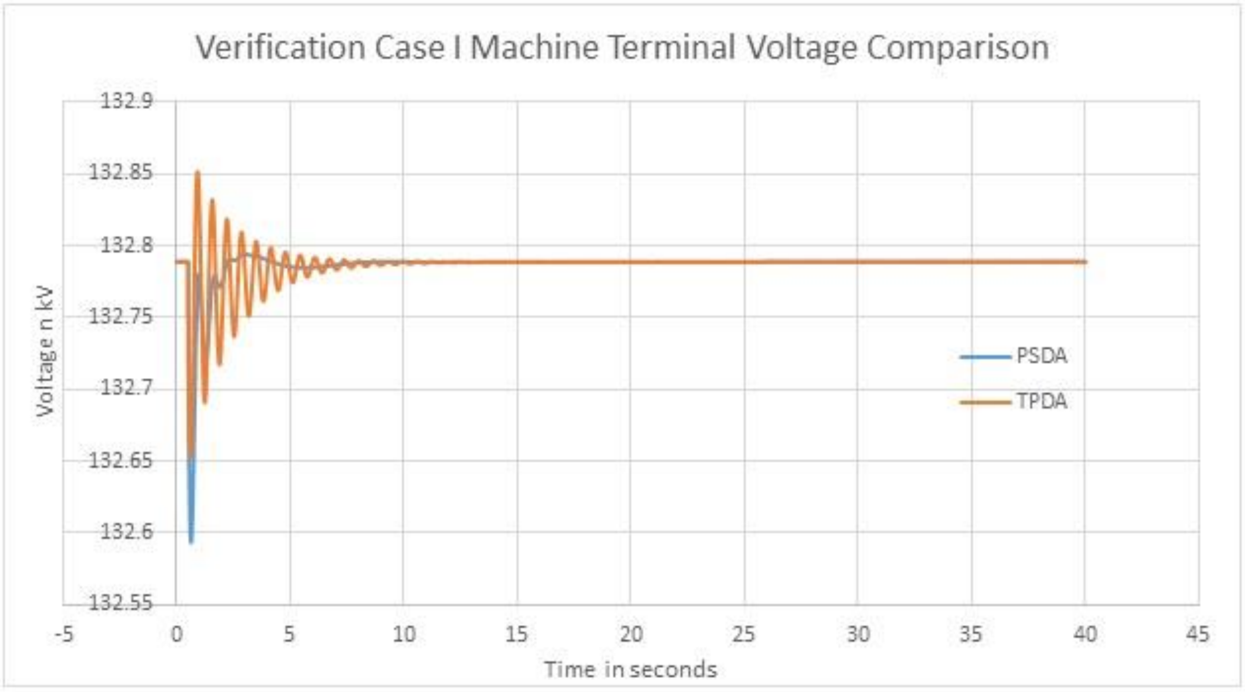


Figure 3.4: Verification Case I Voltage Comparison: TPDA –vs- PSLF

Figure 3.5 plots rotor speed computed by the two methods. Both methods are found to generate results that match closely.

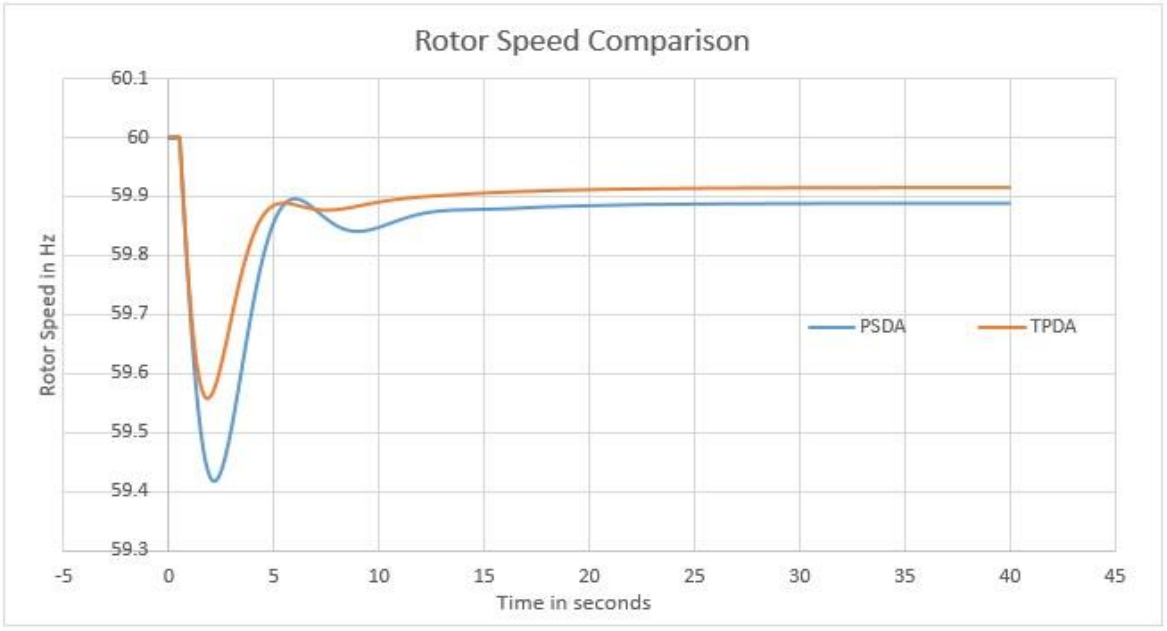


Figure 3.5: Verification Case I Rotor speed comparison

Next, the field currents (ifd) and field voltages (efd) are compared to verify TPDA's performance with respect to the exciter. Both the field current and the output field voltage of the exciter are noted to have been initialized at slightly different values (lower by about 0.067 p.u.) using three-phase analysis. Further, both ifd and efd settle at a lower value (lower by about 0.07 p.u.) at the end of the simulation in case of three-phase analysis. However, it must be noted that the three-phase analysis results are off by almost the same small amount at the start and the end of simulation, indicating a consistency in the waveform profiles. It can be observed from figures 3.6 and 3.7 that the waveforms profiles (for ifd and efd respectively) are almost the same, irrespective of the method used for analysis.

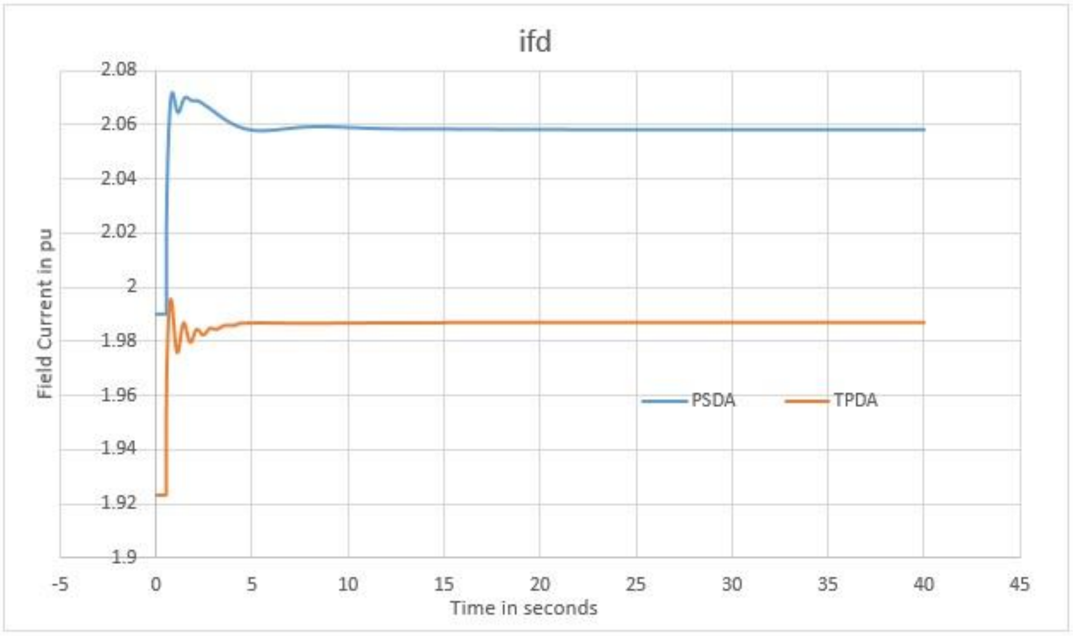


Figure 3.6: Verification Case I Field current comparison

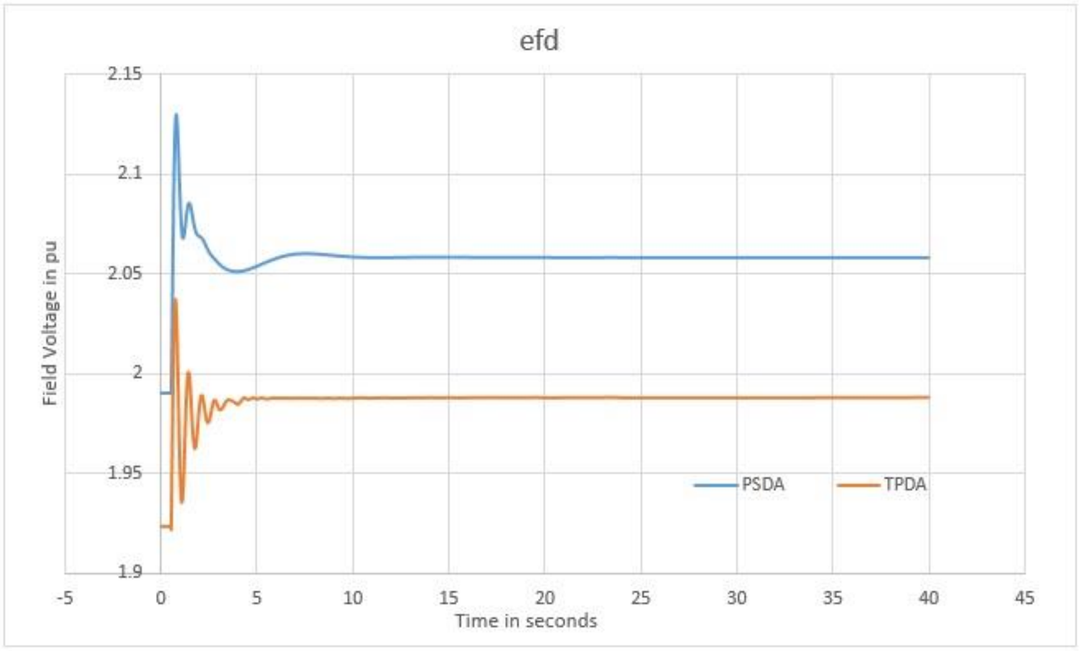


Figure 3.7: Verification Case I: Comparing the output field voltage of the exciter

Finally, the mechanical power output of the governor is compared to verify the accuracy of TPDA. Figure 3.8 shows closely matched results obtained using the two analysis techniques.

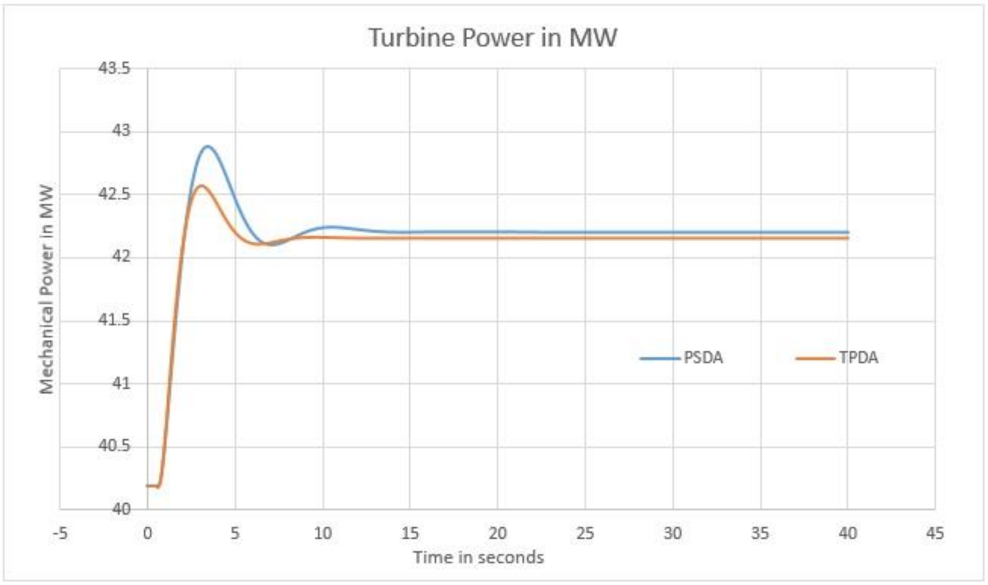


Figure 3.8: Verification Case I Mechanical power output comparison

Hence, based on the plots shown above, TPDA can closely match PSDA results. Next, we compare the performance of TPDA and PSDA on the WECC 9-bus test system.

Case II: WECC 9-bus test system

Figures 3.9, 3.10 and 3.11 compare the terminal voltages for the three machines. The average of the three-phase voltages is used for comparison with PSDA result. All three plots suggest a high level of accuracy for TPDA.

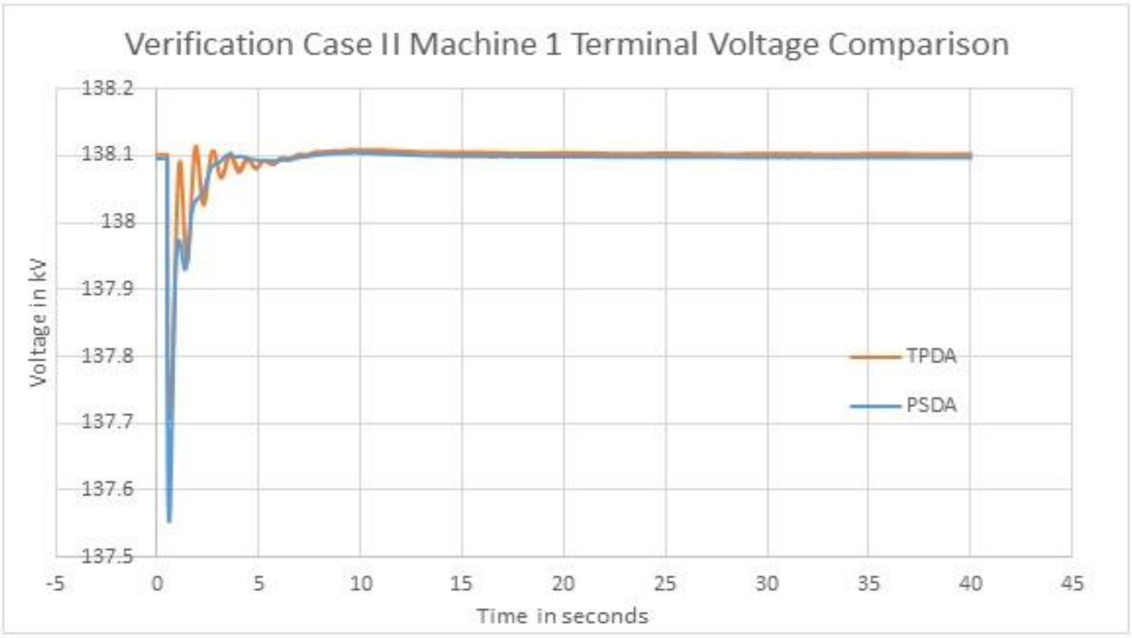


Figure 3.9: Verification Case II Machine 1 Terminal Voltage Comparison

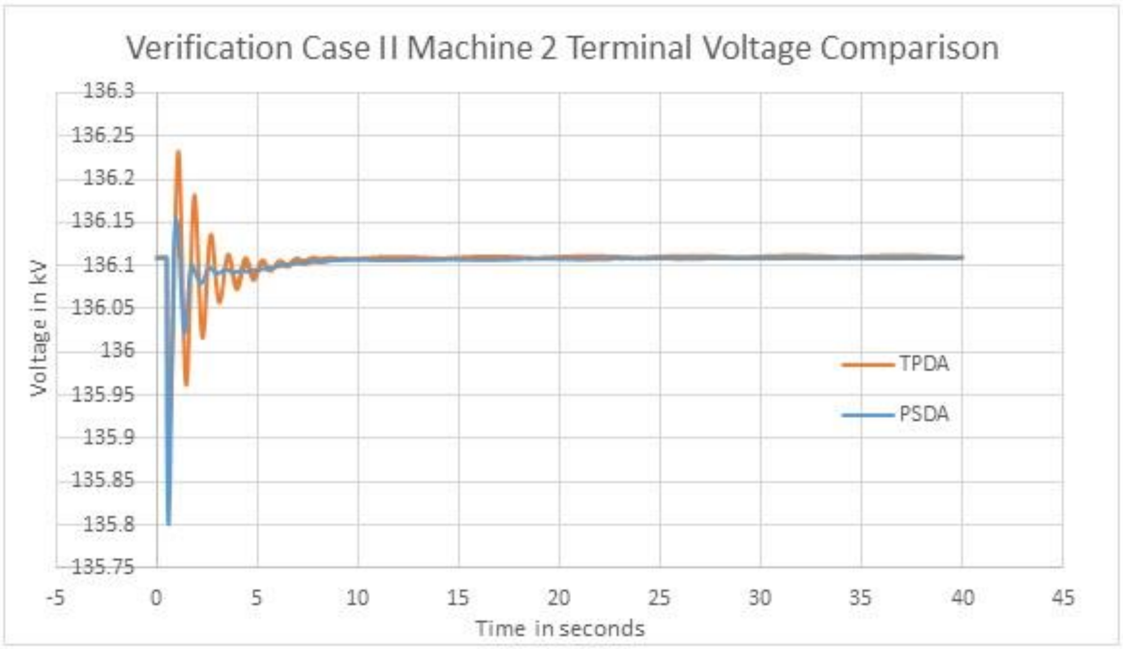


Figure 3.10: Verification Case II Machine 2 Terminal Voltage Comparison

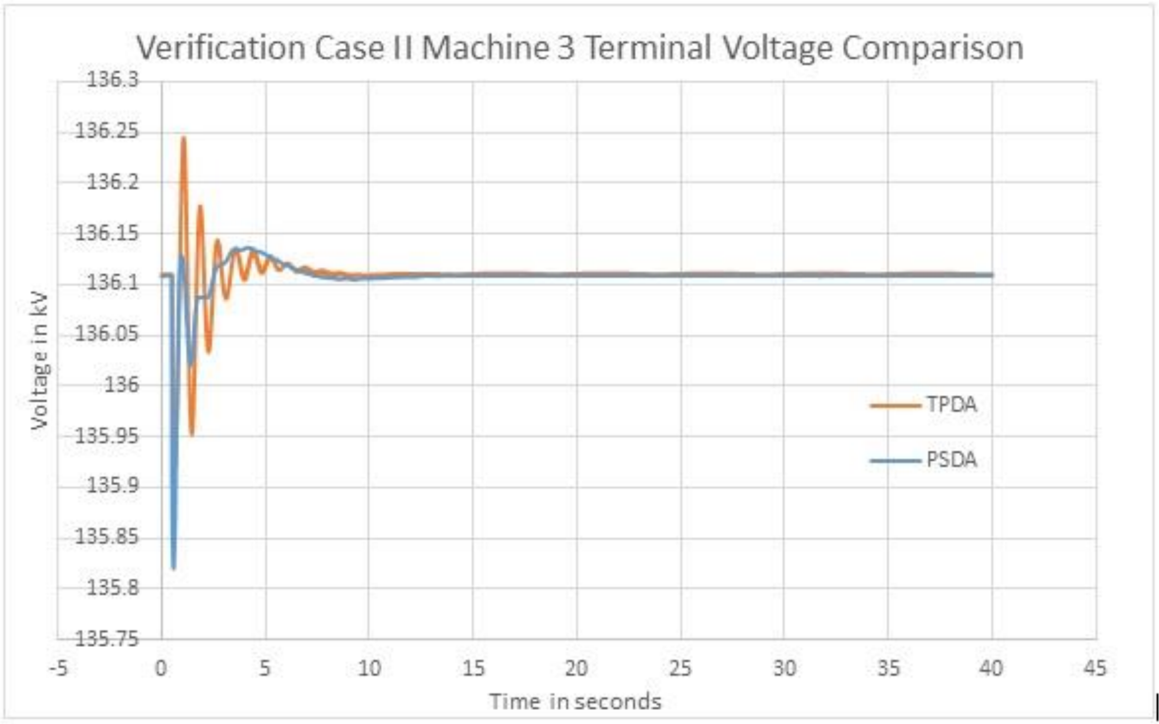


Figure 3.11: Verification Case II Machine 3 Terminal Voltage Comparison

Further, the three-phase voltages are again seen to perfectly overlap each other for all three machines. This is depicted in plots 3.12-3.14.

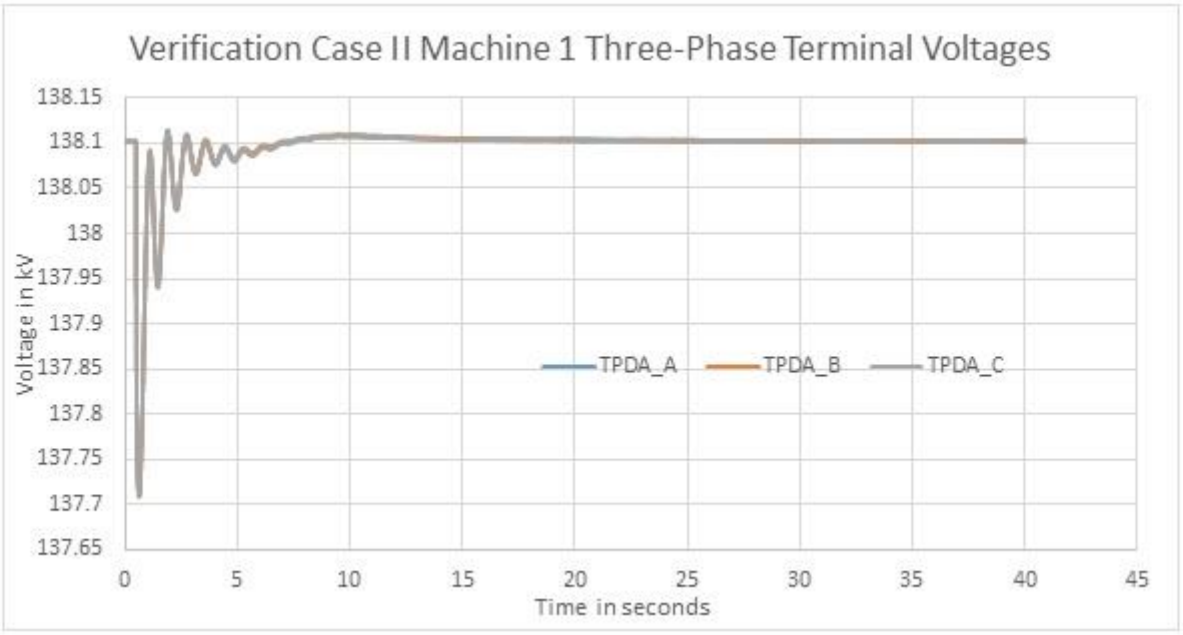


Figure 3.12: Verification Case II: Machine 1 Three-Phase Terminal Voltages

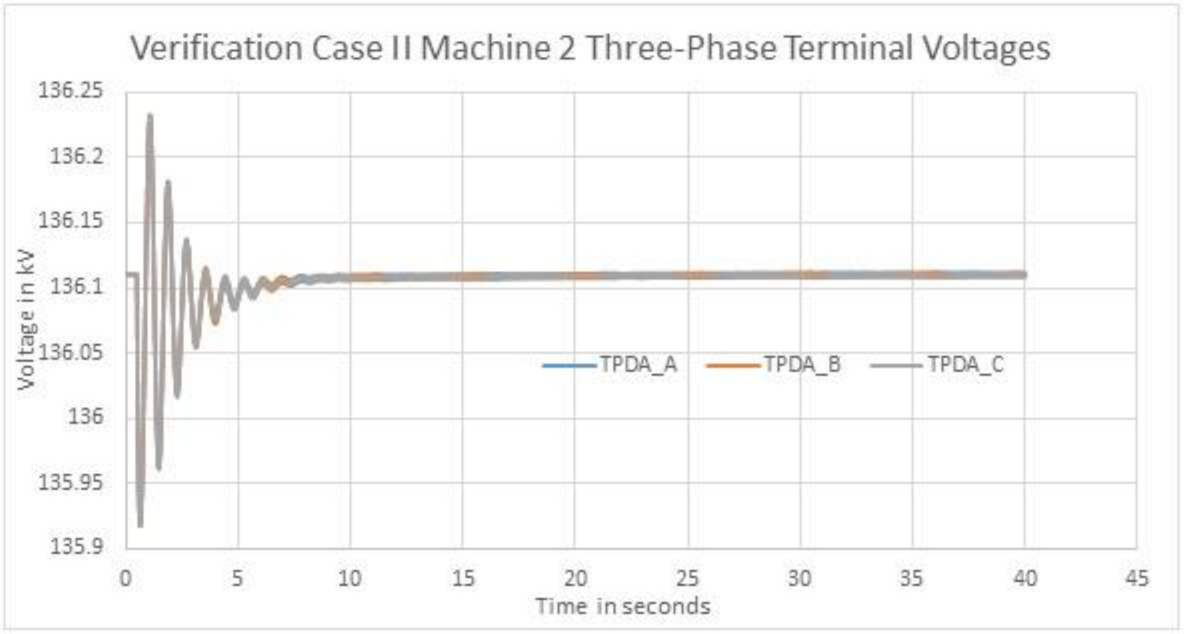


Figure 3.13: Verification Case II: Machine 1 Three-Phase Terminal Voltages

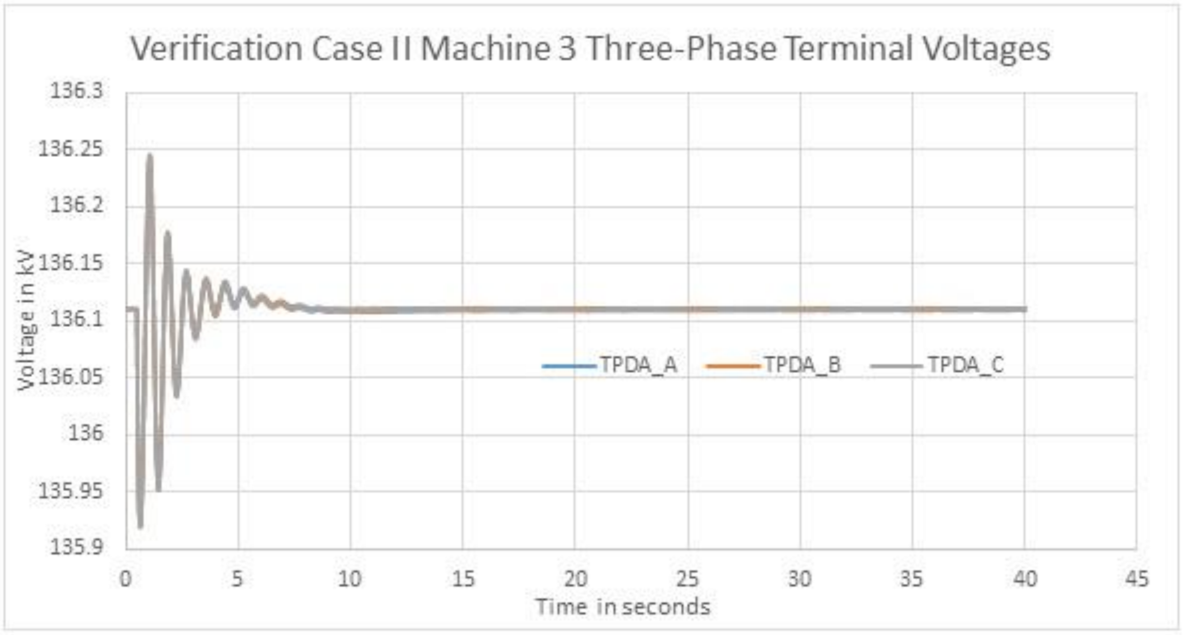


Figure 3.14: Verification Case II: Machine 1 Three-Phase Terminal Voltages

Next, rotor speeds for each machine are plotted using both analysis techniques. Figure 3.15 shows that the expected synchronism between the three machines is not lost under the proposed technique that solves the dynamic models for every machine component separately. Moreover, rotor speed results obtained using TPDA compare well to those obtained using PSDA.

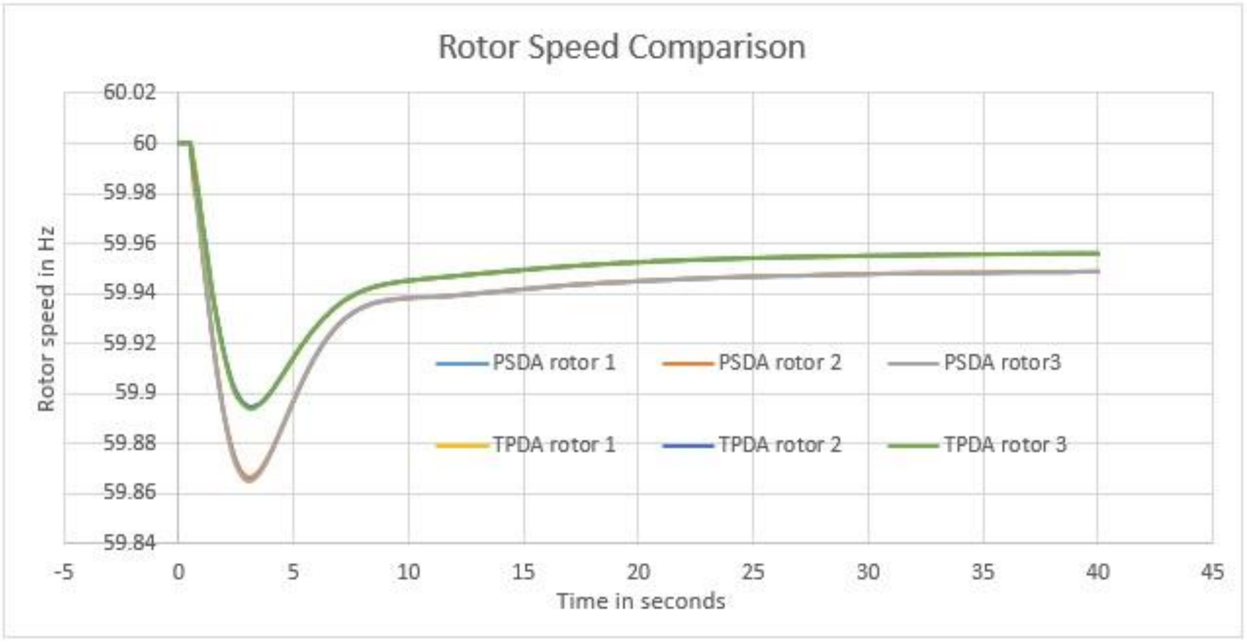


Figure 3.15: Verification Case II: Rotor Speed Comparison

Figures 3.16 and 3.17 show the field current and field voltage comparisons for all three machines. Finally, the mechanical power output of the two governors in the system is compared in figure 3.18. From the figures it may be visually verified that TPDA results closely match PSDA results.

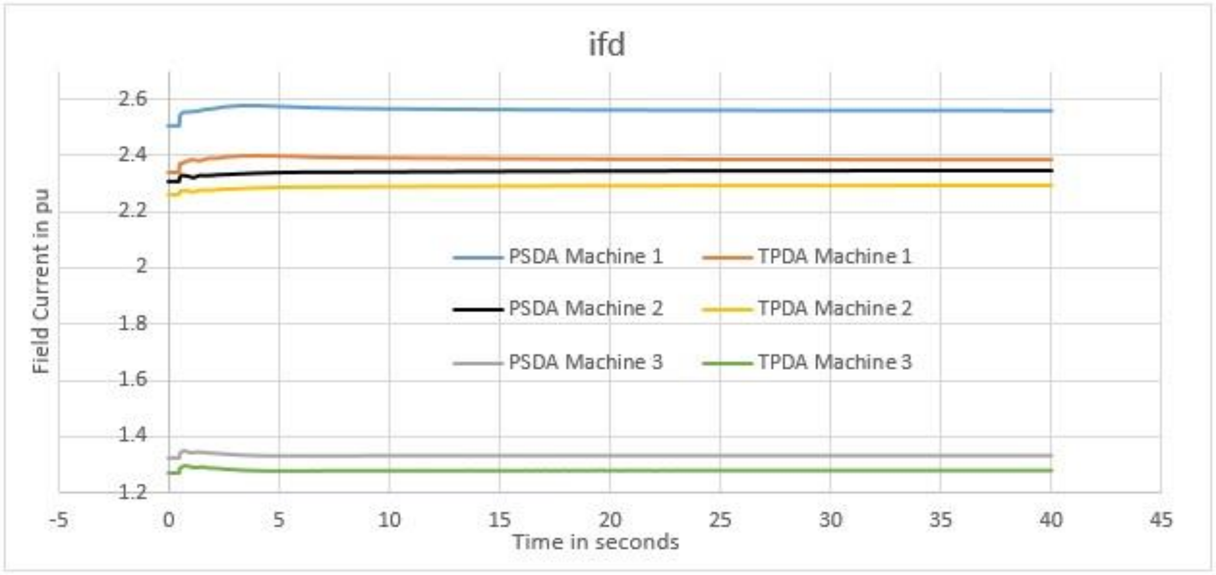


Figure 3.16: Verification Case II: Field Current (ifd) comparison for all machines

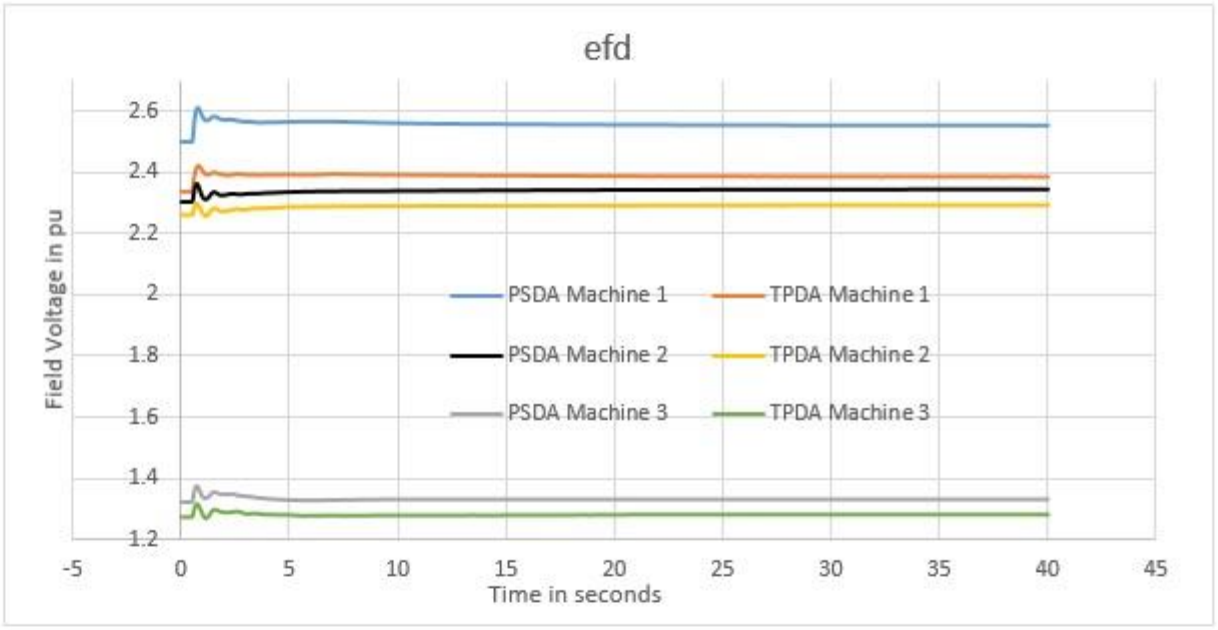


Figure 3.17: Verification Case II: Field Voltage (efd) comparisons for all machines

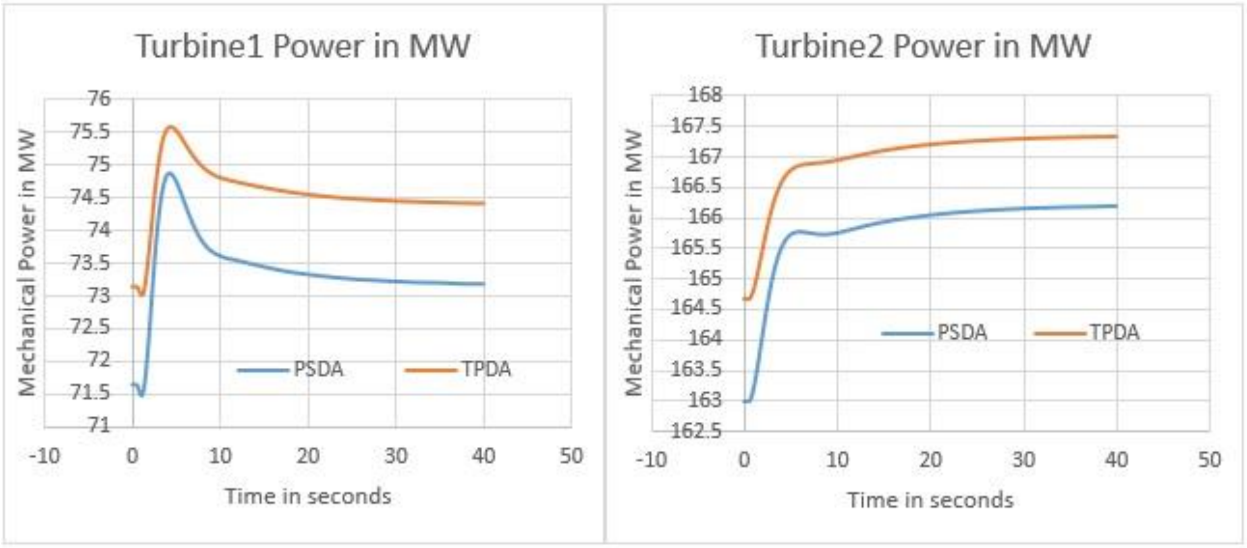


Figure 3.18: Verification Case II: Mechanical power output comparison for all governors in the system

Based on the above verification results it is concluded that TPDA can closely match PSDA results under balanced conditions.

3.3 Advantages of TPDA over PSDA for Performing Dynamic Simulations.

Most electro-mechanical dynamic analysis in the power industry is done using software tools that assume a positive sequence model for the transmission network. In this section dynamic analysis results obtained using PSDA are compared to the results obtained using TPDA in an attempt to highlight the benefits of three-phase dynamic analysis. The electric network of figure 3.1 is used. In the simulations, load reductions at load busses 5 and 6 occur at time t = 0.5 seconds. At t = 20 seconds loads are restored to the original values. The total simulation time considered is 40 seconds. The aim is to capture the effects of imbalance on the transmission network (usually assumed to be balanced) due to unbalanced DG generation or load dynamics associated with the inherently unbalanced distribution networks. Furthermore, transmission system impedances are also generally unbalanced, having some effect on the overall system balance.

In the three phase dynamic analysis the load reduction is modeled as a single phase reduction on phase A. The positive sequence analysis software does not allow for a single phase load reduction, and thus in this case a three-phase load reduction of equal magnitude is applied over the same time interval.

Three different load reductions are considered: a 6MW load reduction, a 30 MW load reduction and a 60 MW load reduction, where the total load on the system of figure 3.1 is approximately 315 MW. The load reduction is split equally between load busses 5 and 6, and is implemented with a 0.95 leading power factor. Table 3.2 summarizes the three cases.

Table 3.2: Test Cases

Case #	Load Reduction (Split equally between busses 5 and 6)
Case 1	6 MW @ 0.95 leading pf => 6 MW and 2 MVAR
Case 2 =Case 1 * 5	30 MW @ 0.95 leading pf => 30 MW and 10 MVAR
Case 3 = Case 1 * 10	60 MW @ 0.95 leading pf => 60 MW and 20 MVAR

3.3.1 Results and Analysis

Three phase analysis is shown to capture some valuable new information that was missed by assuming a balanced positive sequence model for the transmission system. This section is divided into four sub-sections to provide a comprehensive analysis of the results obtained using TPDA. These are: Machine Terminal Voltage Analysis, Voltage Imbalance Analysis at Other Network Busses, Current Imbalance Analysis and Rotor Speed Analysis. In the first three sub-sections, three parameters are used to analyze results. These are:

1) MI (Maximum Imbalance): Maximum imbalance in the three-phase results. This is a direct measure of the imbalance in the network.
    a. MIR (Maximum Imbalance Ratio) is defined as $\frac{MI\ for\ Case\ j}{MI\ for\ Case\ 1}; j = 1,2,3$
2) PD (Peak Deviation): Peak deviation from initial steady state values. This is a direct measure of the impact of the disturbance.
    a. PDR (Peak Deviation Ratio) is defined as $\frac{PD\ for\ Case\ j}{PD\ for\ Case\ 1}; j = 1,2,3$
3) MS (Maximum Swing): Maximum magnitude of oscillations when the voltages and currents are seen to oscillate about a reasonably constant value during unbalanced system conditions (from t=5 seconds to 20 seconds for voltages and t=10 seconds to 20 seconds for currents). This provides a measure for the swing in currents and voltages over back-to-back iterations of TPDA.

a. MSR (Maximum Swing Ratio) is defined as $\frac{MS\ for\ Case\ j}{MS\ for\ Case\ 1}; j = 1,2,3$

Rotor speed deviations are analyzed by measuring the maximum rotor speed deviations and comparing them to PSDA results.

3.3.1.1 Machine Terminal Voltage Analysis

Generator terminal voltages are analyzed here. Comparison is made between PSDA and TPDA. This comparison reveals a very interesting result: the two-point moving average of the mean of three-phase voltages is seen to closely follow PSDA results for generator terminal voltage. This is shown in figures 3.19, 3.21 and 3.23. Further, three-phase results are presented in figures 3.20, 3.22 and 3.24 showing significant voltage variations. Tables 3.3, 3.4 and 3.5 summarize the results for parameters 1, 2 and 3 mentioned above. It is seen that as the magnitude of the disturbance increases from Case 1 to Case 3, generator 1 terminal voltage results corresponding to all three parameters (MI, PD and MS) also increases almost proportionally. This is reflected by a near linear increase in the three parameter ratios (MIR, PDR and MSR) from Case 1 to Case 3.

Three phase dynamic analysis of the test system has revealed MI = 0.470 kV, PD = 2.66 kV and MS = 1.3 kV at generator 1 terminals for a 30 MW unbalanced change when total system load = 315 MW (<10% unbalanced change). This information could not be captured by assuming a positive sequence model for the electric network (figures 3.19, 3.21 and 3.23). This is a good measure of the importance and need for three-phase dynamic analysis.

Case 1: Total Load Reduction = 6 MW (3 MW on load bus 5 and 6 each) at 0.95 leading pf.

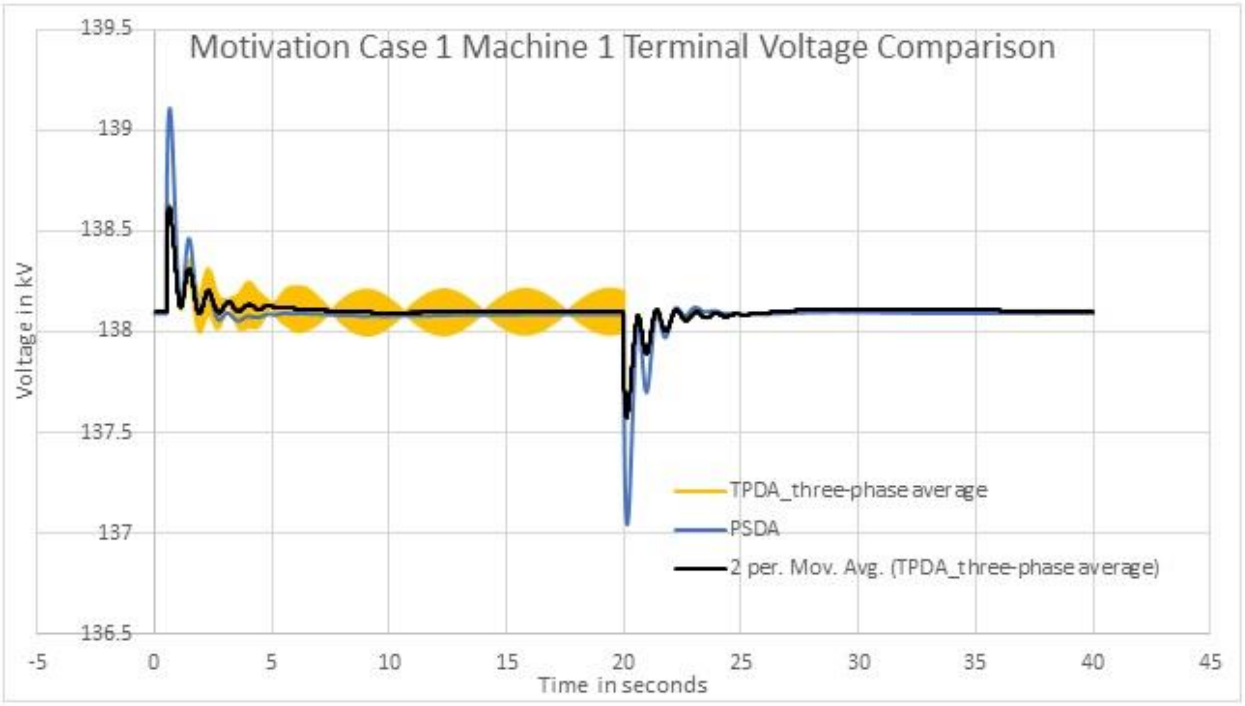


Figure 3.19: Motivation Case 1: Machine 1 Terminal Voltage Comparison

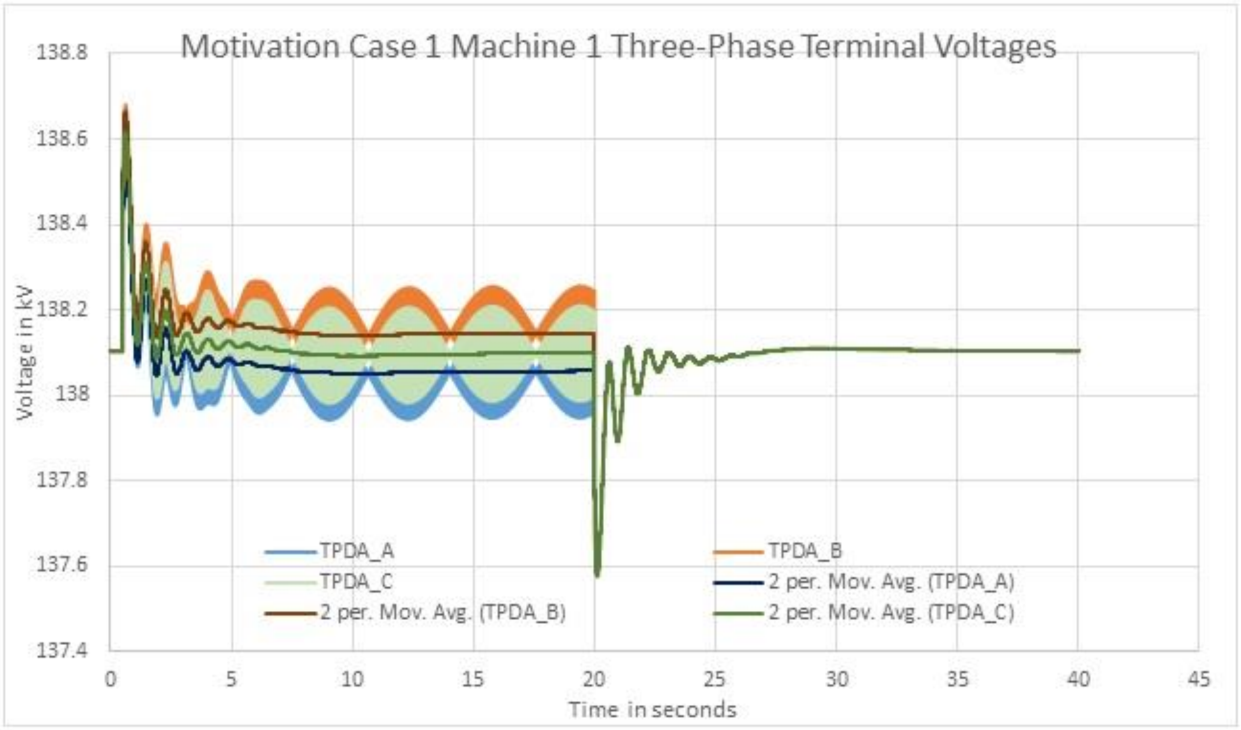


Figure 3.20: Motivation Case 1: Machine 1 Three-Phase Terminal Voltages

Case 2: Total Load Reduction = 30 MW (15 MW on load bus 5 and 6 each) at 0.95 leading pf.

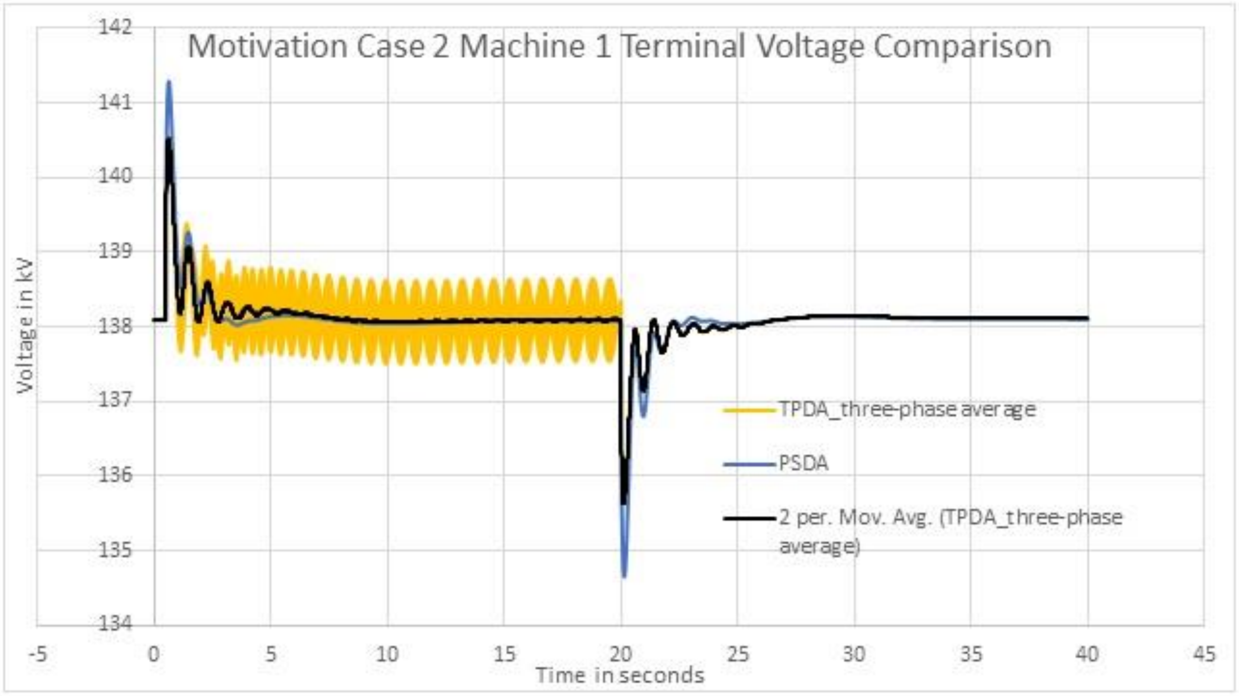


Figure 3.21: Motivation Case 2: Machine 1 Terminal Voltage Comparison

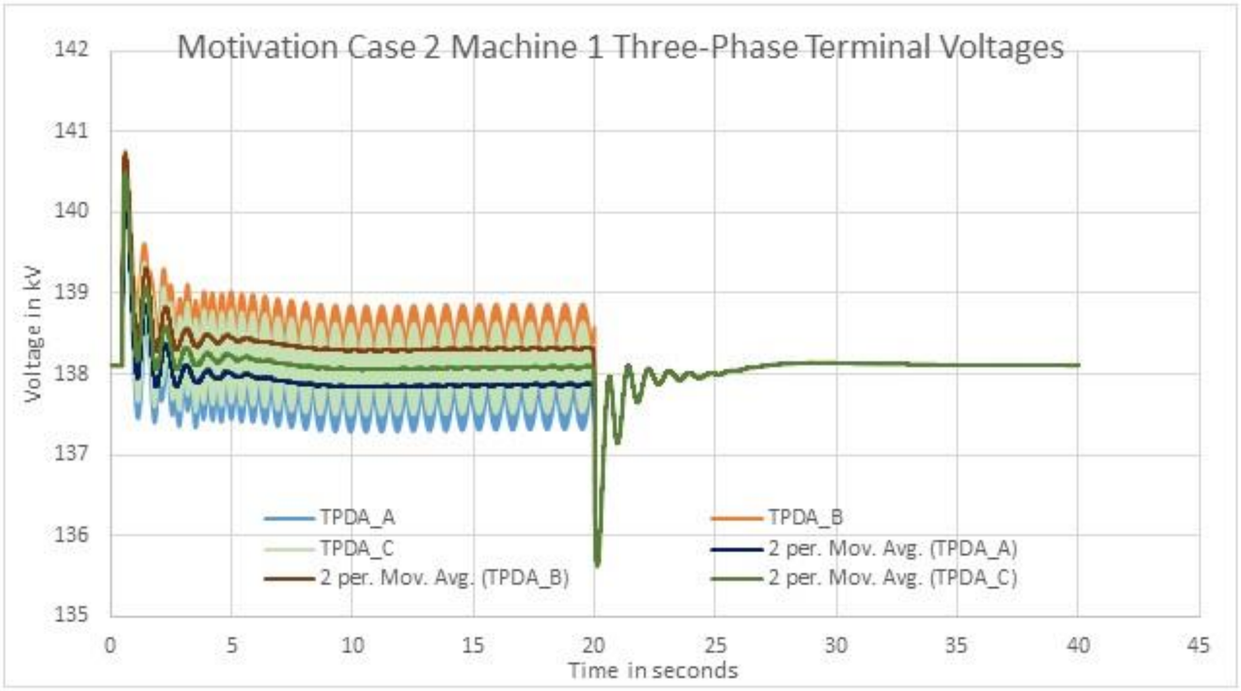


Figure 3.22: Motivation Case 2: Machine 1 Three-Phase Terminal Voltages

Case 3: Total Load Reduction = 60 MW (30 MW on load bus 5 and 6 each) at 0.95 leading pf.

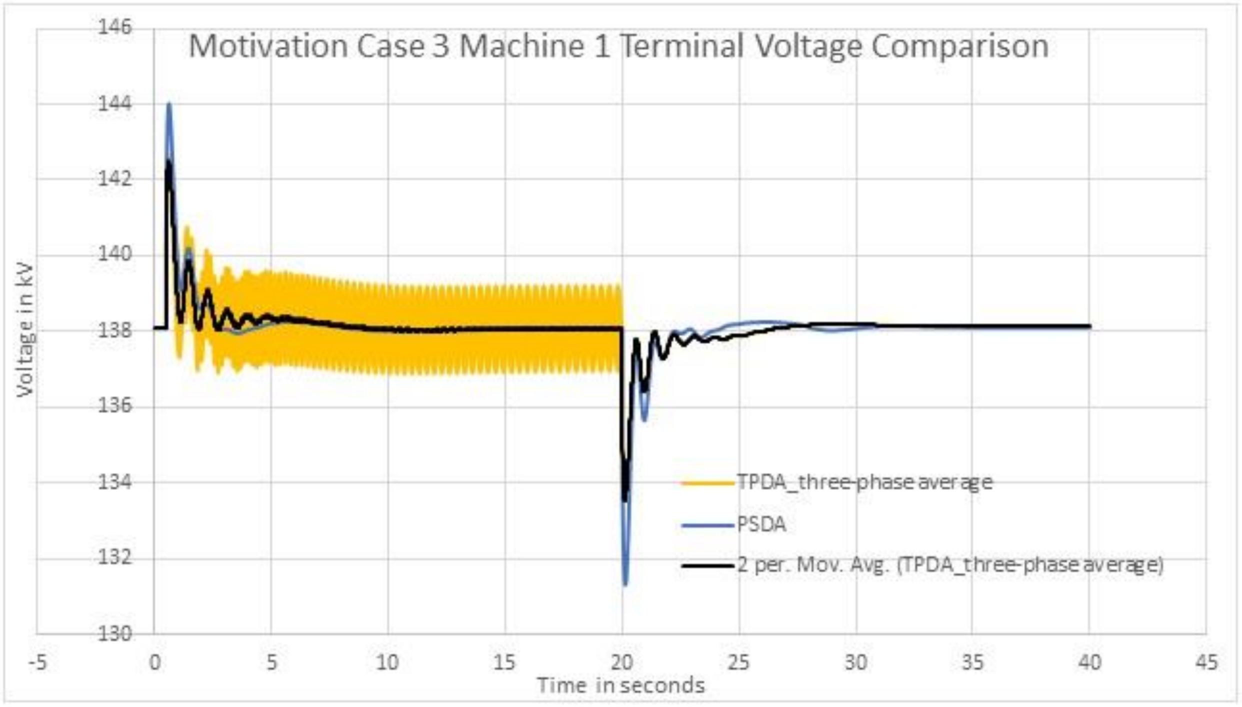


Figure 3.23: Motivation Case 1: Machine 1 Terminal Voltage Comparison

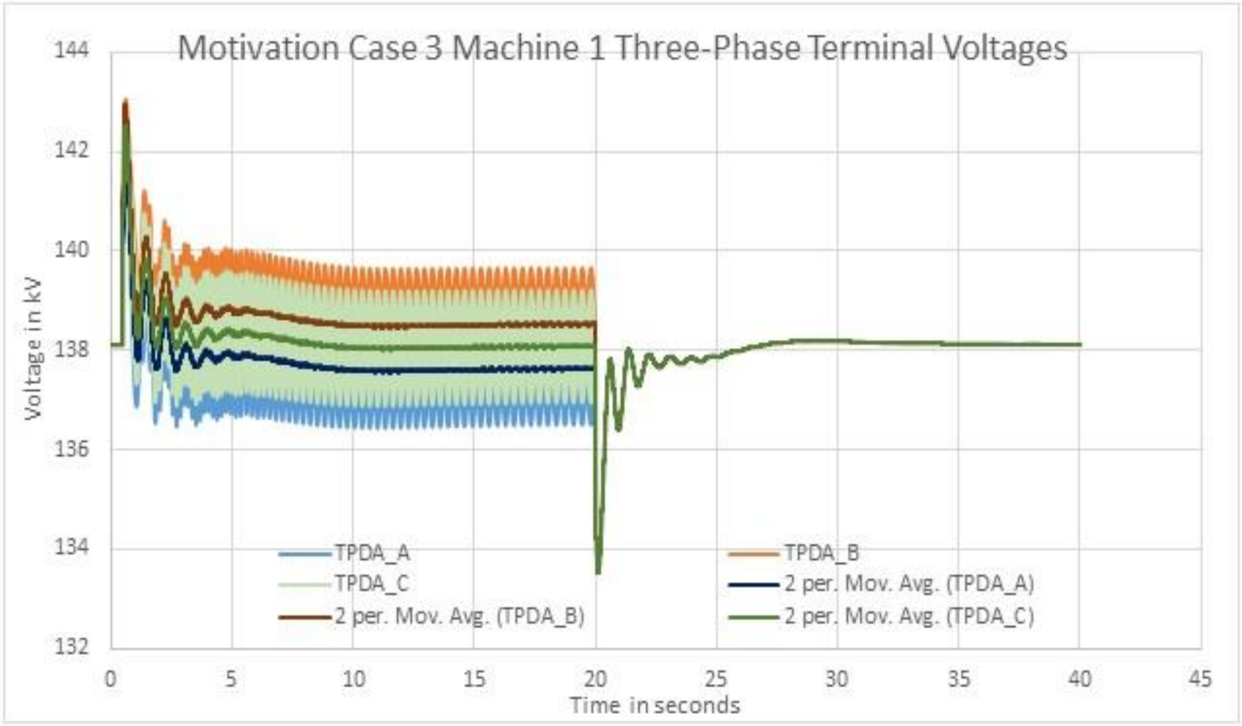


Figure 3.24: Motivation Case 3: Machine 1 Three-Phase Terminal Voltages

Table 3.3: Analyzing the maximum imbalance in generator terminal voltages

Case #	Max Imbalance (MI) at any time (in kV)	MIR
6 MW (Base Case)	0.094289 kV	0.094289/0.094289 = **1**
30 MW = Base Case*5	0.470033 kV	0.470033/0.094289 = **4.985**
60 MW = Base Case*10	0.935307 kV	0.935307/0.094289 = **9.920**

Table 3.4: Analyzing the peak deviation in generator terminal voltages

Case #	Peak deviation (PD) from initial value (in kV)	PDR
6 MW (Base Case)	0.57741	0.57741/0.57741 = 1
30 MW = Base Case*5	2.656021	2.656021/0.57741 = **4.600**
60 MW = Base Case*10	4.9520988	4.9520988/0.57741 = **8.576**

Table 3.5: Analyzing the maximum magnitude of oscillations in generator terminal voltages over two subsequent iterations (excluding transients)

Case#	Max. Mag. of oscillations (MS) from t=5 to t=20 sec (in kV)			MSR		
	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
6 MW	0.235296	0.235105	0.239988	**1**	**1**	**1**
30 MW	1.281881	1.28763	1.30156	**5.448**	**5.477**	**5.423**
60 MW	2.681172	2.692381	2.699794	**11.395**	**11.452**	**11.250**

A similar in-depth analysis can be carried out for other machines in the system. Note that generator terminal voltages for other machines in the network were found to have different magnitudes, however, the trend is seen to be followed by all machines.

3.3.1.2 Load Bus Voltage Analysis

Figures 3.25 – 3.33 show that the voltages at all the load busses (5, 6 and 8) are highly imbalanced. It must be noted that although the disturbance occurred only at load busses 5 and 6, load bus 8 is also affected in a similar manner. Understandably, the magnitude of MI, PD and MVS is much less at load bus 8, however, it is not negligible. Further, the proportionally increasing trend observed in case of generator terminal voltages is also observed at all load busses.

For an unbalanced load reduction of 30 MW on a system of 315 MW (<10% change), MI of about 3.5 kV is seen on load busses 5 and 6 (where the load reduction occurred) and 0.9 kV on load bus 8. Moreover, large peak voltage deviations and voltage swings are observed at all load buses. PD is found to be >5 kV at load busses 5 and 6, and >2 kV at load bus 8. Finally, MS is found to be >750 V at load busses 5 and 6, and ~ 300 V at load bus 8. Such large imbalance and deviations would go un-noticed if PSDA is adopted.

Tables 3.6 – 3.14 summarize the impact of unbalanced load reduction on all load bus voltages. It is observed that as the load reduction increases from Case 1 to Case 3, MI, PD and MS - all increase almost proportionally (Exact rates of increase are noted in the MIR, PDR and MSR columns of tables 3.6-3.14).

Load bus 5

Case 1: Total Load Reduction = 6 MW (3 MW on load bus 5 and 6 each) at 0.95 leading pf.

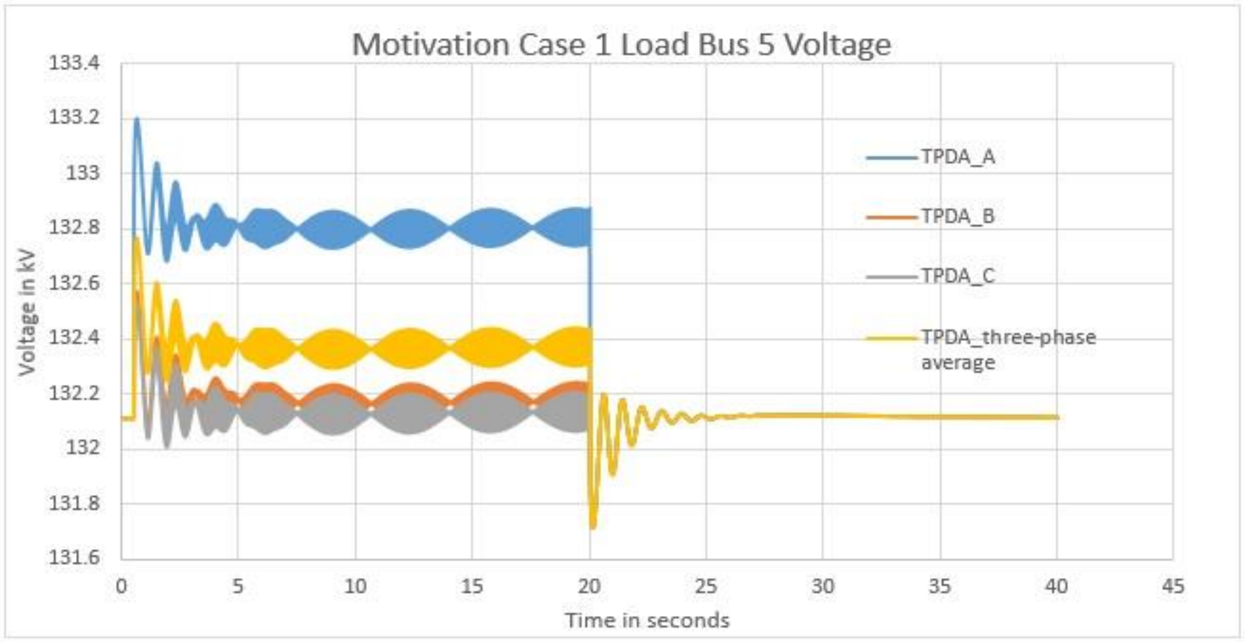


Figure 3.25: Motivation Case 1: Load Bus 5 Three-Phase Voltages

Case 2: Total Load Reduction = 30 MW (15 MW on load bus 5 and 6 each) at 0.95 leading pf.

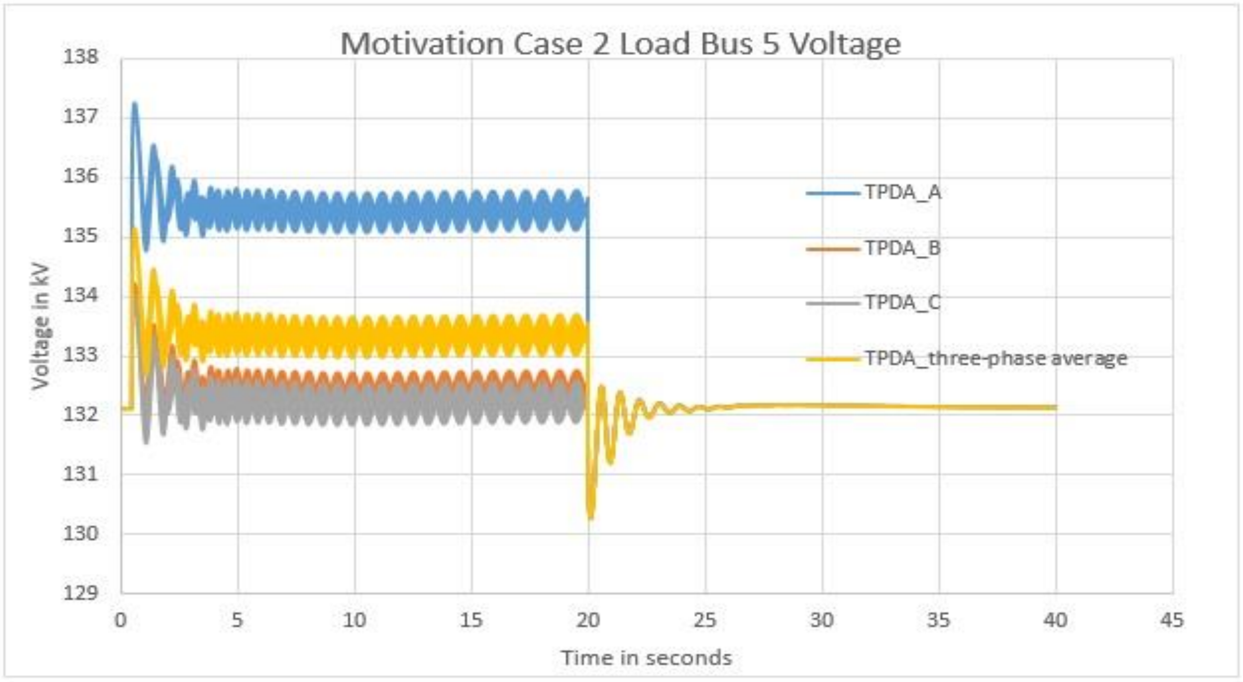


Figure 3.26: Motivation Case 2: Load Bus 5 Three-Phase Terminal Voltages

Case 3: Total Load Reduction = 60 MW (30 MW on load bus 5 and 6 each) at 0.95 leading pf.

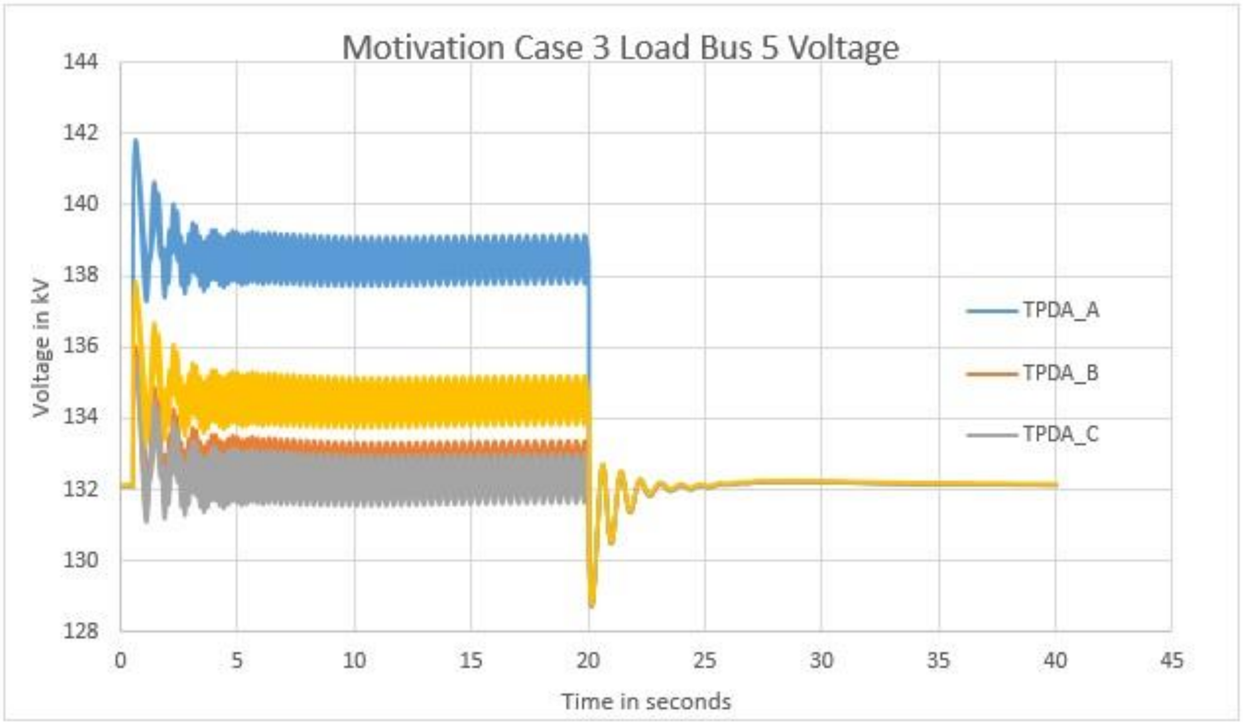


Figure 3.27: Motivation Case 3: Load Bus 5 Three-Phase Terminal Voltages

Table 3.6: Analyzing the maximum imbalance in load bus 5 voltages

Case #	Max Imbalance (MI) at any time (in kV)	MIR
6 MW (Base Case)	0.698	**1**
30 MW = Base Case*5	3.38	**4.842**
60 MW = Base Case*10	6.488	**9.295**

Table 3.7: Analyzing the peak deviation in load bus 5 voltages

Case #	Peak Deviation (PD) from initial value (in kV)	PDR
6 MW (Base Case)	1.087	**1**
30 MW = Base Case*5	5.141	**4.730**
60 MW = Base Case*10	9.727	**8.950**

Table 3.8: Analyzing the maximum magnitude of oscillations in load bus 5 voltages over two subsequent iterations (excluding transients)

Case #	Max. Mag. of oscillations (MS) from t=5 to t=20 sec (in kV)			MSR		
	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
6 MW	0.142	0.145	0.149	**1**	**1**	**1**
30 MW	0.738	0.761	0.777	**5.197**	**5.248**	**5.215**
60 MW	1.55	1.593	1.612	**10.916**	**10.986**	**10.819**

Load bus 6

Case 1: Total Load Reduction = 6 MW (3 MW on load bus 5 and 6 each) at 0.95 leading pf.

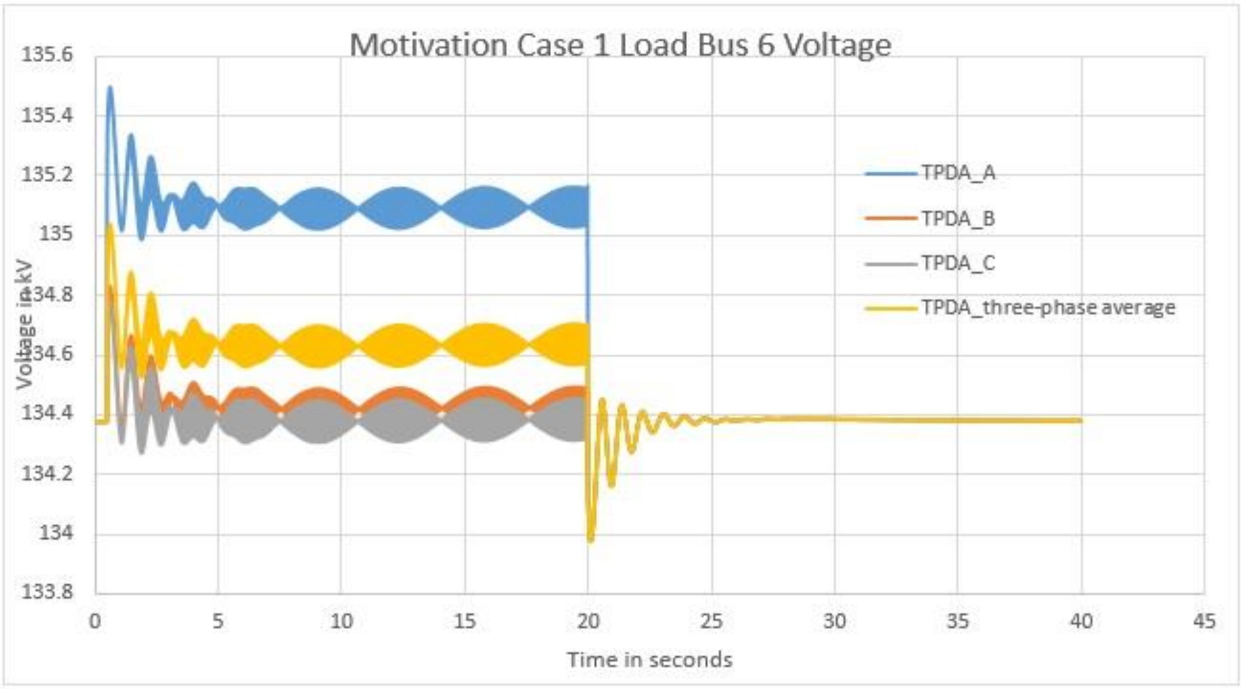


Figure 3.28: Motivation Case 1: Load Bus 6 Three-Phase Terminal Voltages

Case 2: Total Load Reduction = 30 MW (15 MW on load bus 5 and 6 each) at 0.95 leading pf.

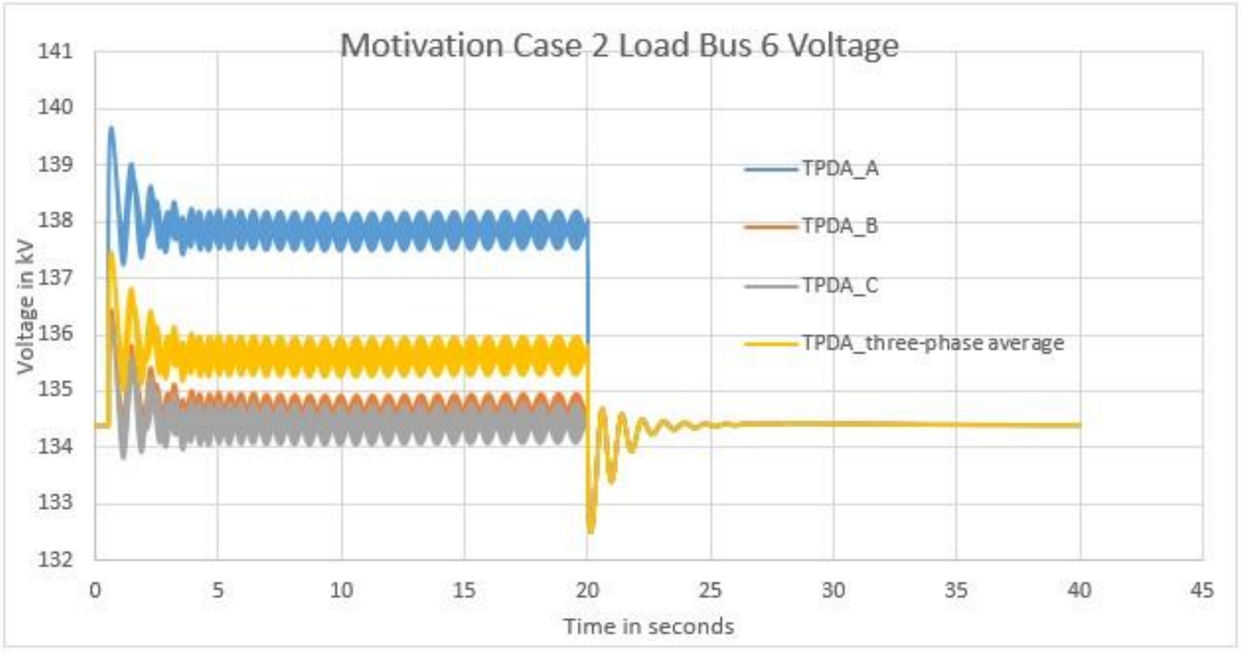


Figure 3.29: Motivation Case 2: Load Bus 6 Three-Phase Terminal Voltages

Case 3: Total Load Reduction = 60 MW (30 MW on load bus 5 and 6 each) at 0.95 leading pf.

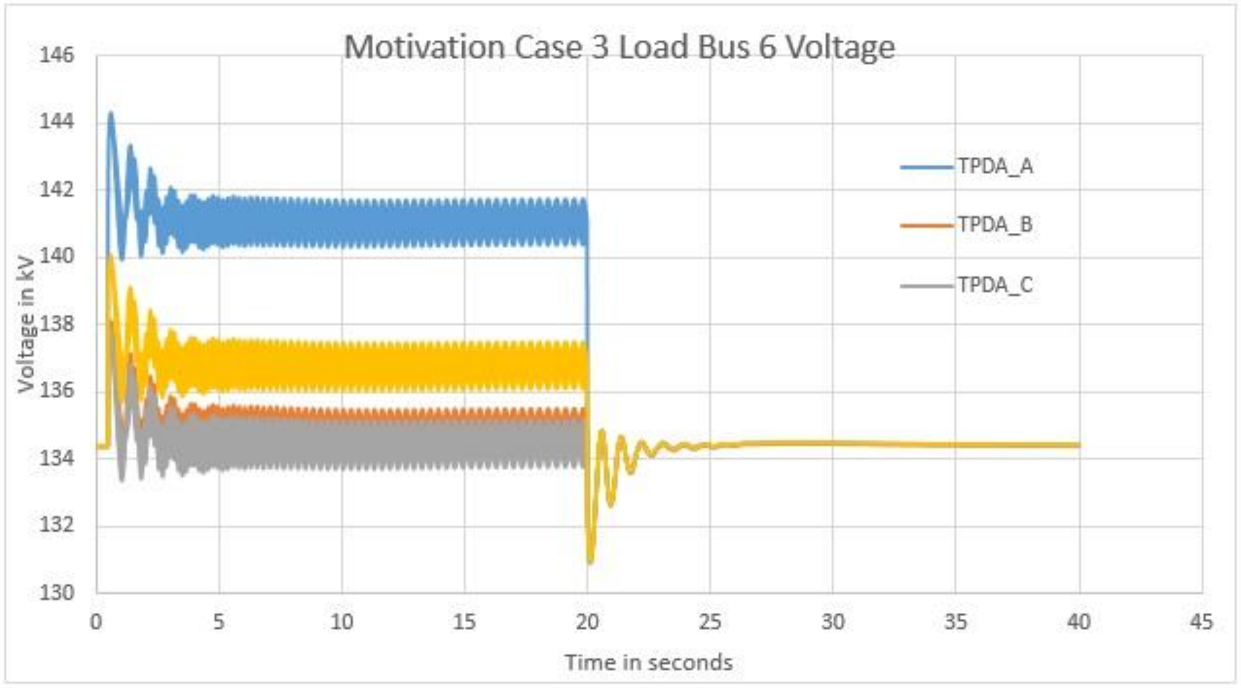


Figure 3.30: Motivation Case 3: Load Bus 6 Three-Phase Terminal Voltages

Table 3.9: Analyzing the maximum imbalance in load bus 6 voltages

Case #	Max Imbalance (MI) at any time (in kV)	MIR
6 MW (Base Case)	0.741	**1**
30 MW = Base Case*5	3.594	**4.850**
60 MW = Base Case*10	6.909	**9.324**

Table 3.10: Analyzing the peak deviation in load bus 6 voltages

Case #	Peak Deviation (PD) from initial value (in kV)	PDR
6 MW (Base Case)	1.122	**1**
30 MW = Base Case*5	5.301	**4.725**
60 MW = Base Case*10	9.946	**8.865**

Table 3.11: Analyzing the maximum magnitude of oscillations in load bus 6 voltages over two subsequent iterations (excluding transients)

Case #	Max. Mag. of oscillations (MS) from t=5 to t=20 sec (in kV)			MSR		
	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
6 MW	0.14	0.143	0.148	**1**	**1**	**1**
30 MW	0.721	0.743	0.756	**5.15**	**5.196**	**5.108**
60 MW	1.537	1.587	1.615	**10.979**	**11.098**	**10.912**

Load bus 8

Case 1: Total Load Reduction = 6 MW (3 MW on load bus 5 and 6 each) at 0.95 leading pf.

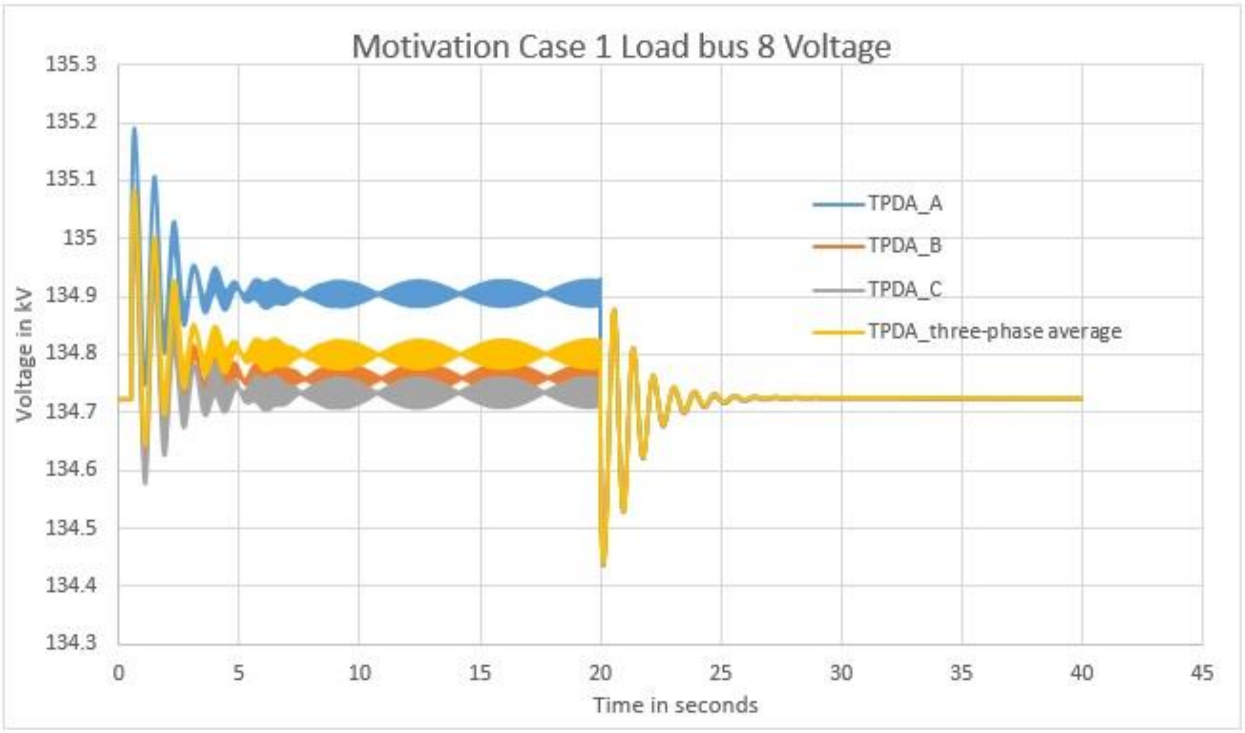


Figure 3.31: Motivation Case 1: Load Bus 8 Three-Phase Terminal Voltages

Case 2: Total Load Reduction = 30 MW (15 MW on load bus 5 and 6 each) at 0.95 leading pf.

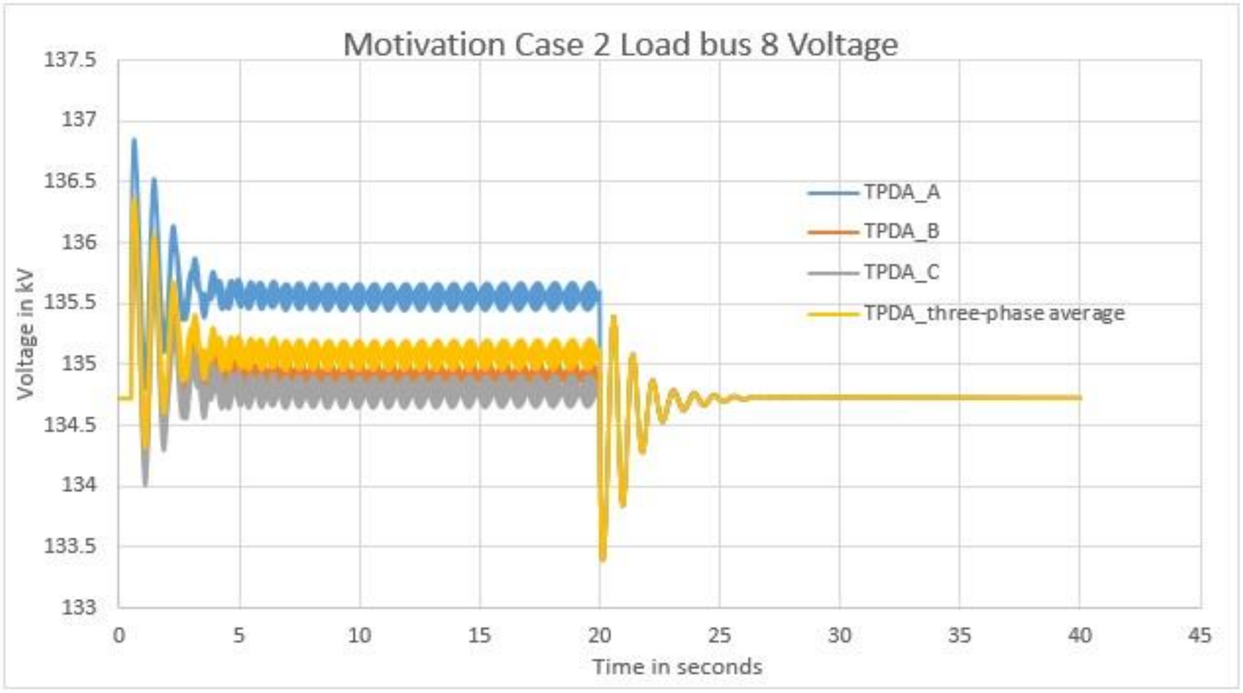


Figure 3.32: Motivation Case 2: Load Bus 8 Three-Phase Terminal Voltages

Case 3: Total Load Reduction = 60 MW (30 MW on load bus 5 and 6 each) at 0.95 leading pf.

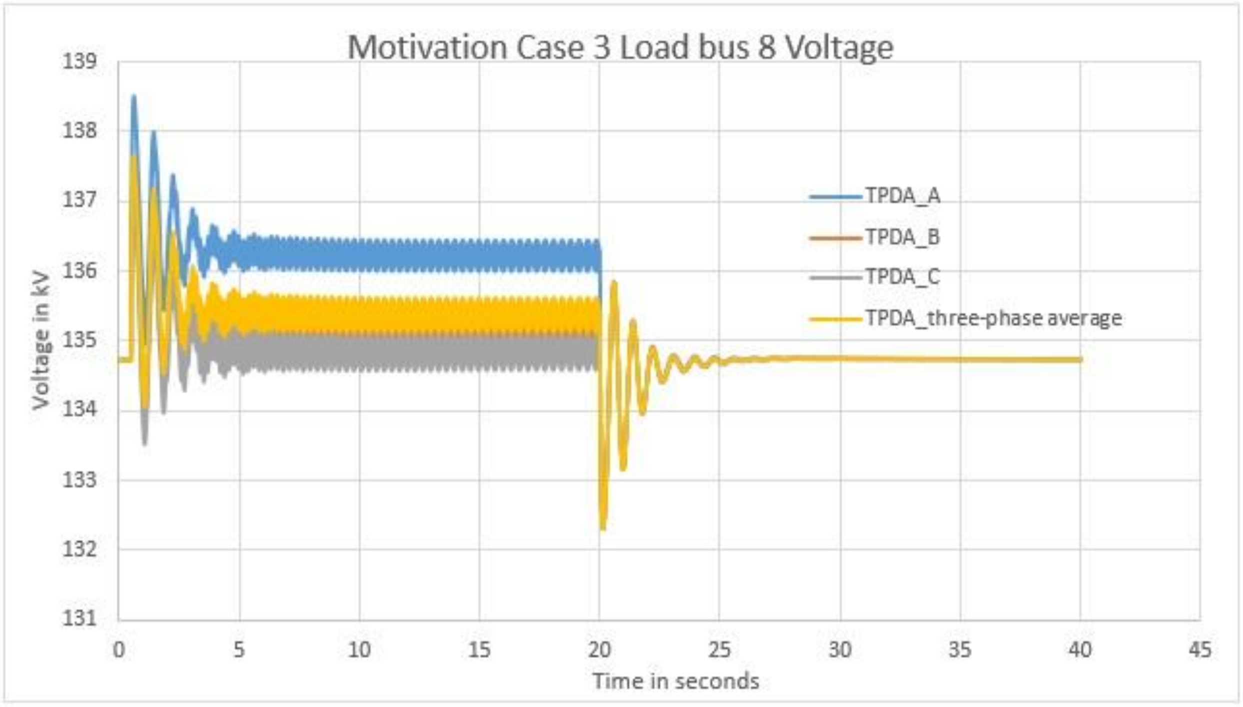


Figure 3.33: Motivation Case 3: Load Bus 8 Three-Phase Terminal Voltages

Table 3.12: Analyzing the maximum imbalance in load bus 8 voltages

Case #	Max Imbalance (MI) at any time (in kV)	MIR
6 MW (Base Case)	0.192	**1**
30 MW = Base Case*5	0.903	**4.703**
60 MW = Base Case*10	1.671	**8.703**

Table 3.13: Analyzing the peak deviation in load bus 8 voltages

Case #	Peak Deviation (PD) from initial value (in kV)	PDR
6 MW (Base Case)	0.467	**1**
30 MW = Base Case*5	2.135	**4.572**
60 MW = Base Case*10	3.793	**8.122**

Table 3.14: Analyzing the maximum magnitude of oscillations in load bus 8 voltages over two subsequent iterations (excluding transients)

Case #	Max. Mag. of oscillations (MS) from t=5 to t=20 sec (in kV)			MSR		
	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
6 MW	0.049	0.056	0.059	**1**	**1**	**1**
30 MW	0.251	0.286	0.299	**5.122**	**5.107**	**5.068**
60 MW	0.537	0.633	0.655	**10.959**	**11.304**	**11.102**

3.3.1.3 Current Analysis

Figure 3.34 shows a highly imbalanced current leaving generator 1 terminals. As expected, phase ‘A’ current suddenly drops in magnitude as load reduction occurs on phase A. On restoration of balanced conditions, the three phase currents are again seen to move together.

For an unbalanced load reduction of 30 MW on a system of 315 MW (<10% change), MI of about 129.5 Amps is seen in the current leaving generator 1 terminals. Moreover, PD in generator 1 current is observed to be >132 Amps. Finally, MS is found to be >14 Amps on phase B and C of generator 1 current.

Tables 3.15 – 3.17 summarize the impact of unbalanced load reduction on generator 1 current. MIR and PDR are observed to increase linearly with the size of the disturbance for small disturbances. However, as we move to large disturbances (as in case 3), there isn't enough current leaving generator 1 terminals under steady state to allow for a proportionally large deviation/imbalance. So, PDR and MIR are seen to saturate for large disturbances. MSR, on the other hand, is seen to grow proportionally with the size of the disturbance. Currents leaving other generators in the system are observed to be large enough to allow a proportional increase in MIR, PDR and MSR under all three test cases.

Table 3.15: Analyzing the maximum imbalance in generator 1 currents

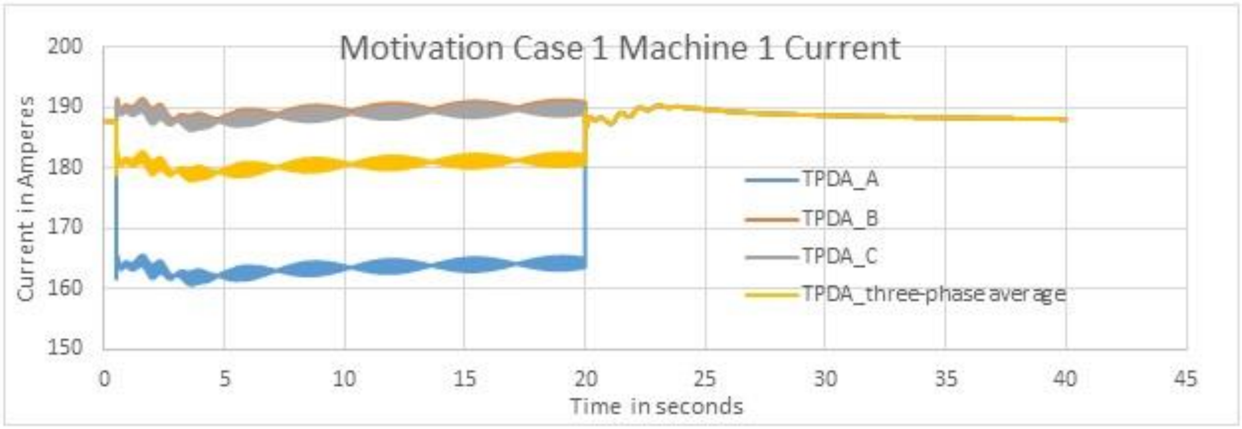
Case #	Max Imbalance (MI) at any time (in Amperes)	MIR
6 MW (Base Case)	25.952	**1**
30 MW = Base Case*5	129.5482	**4.992**
60 MW = Base Case*10	184.1636	**7.096**

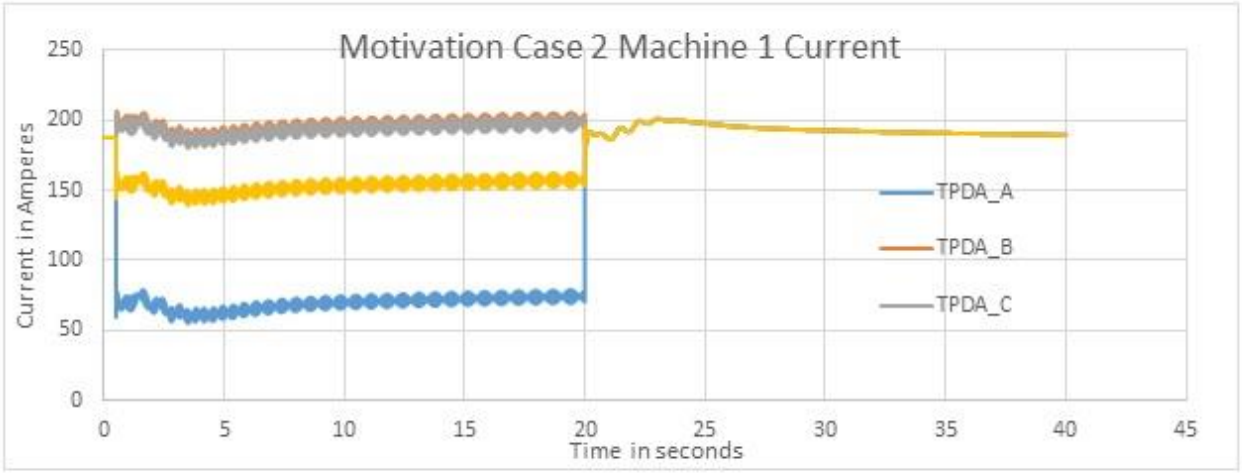
Table 3.16: Analyzing the peak deviation in generator 1 three-phase currents

Case #	Peak Deviation (PD) from initial value (in Amperes)	PDR
6 MW (Base Case)	26.991	**1**
30 MW = Base Case*5	132.398	**4.905**
60 MW = Base Case*10	145.9716	**5.408**

Table 3.17: Analyzing the maximum magnitude of oscillations in load bus 5 voltages over two subsequent iterations (excluding transients)

Case #	Max. Mag. of oscillations (MS) from t=5 to t=20 sec (in Amperes)			MSR		
	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
6 MW	2.723	2.696	2.721	**1**	**1**	**1**
30 MW	13.1136	14.14	14.26	**4.816**	**5.245**	**5.241**
60 MW	25.4325	28.602	28.822	**9.340**	**10.609**	**10.592**





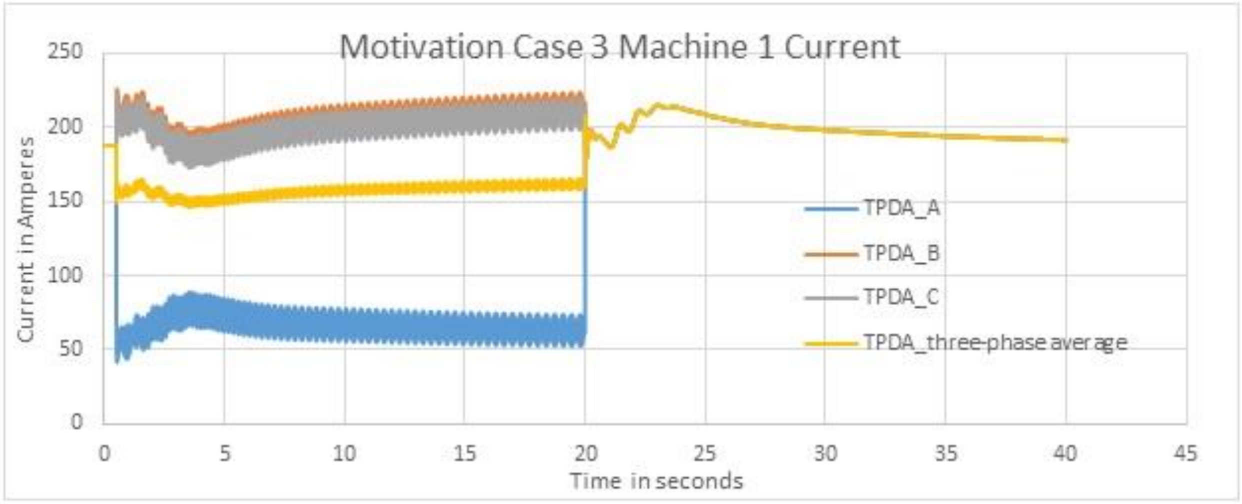


Figure 3.34: Generator 1 Current for Motivation Cases 1, 2 and 3

3.3.1.4 Rotor Speed Analysis

Figures 3.35 – 3.37 show that as the load reduction increases, the deviation in rotor speed about the nominal value of 60 HZ (or 376.9911 rad/sec) gets multiplied by a similar factor (slightly greater) as well. The exact rate of increase in rotor speed deviations is shown in table 3.18. Although there are some differences in the peak deviation results obtained using the three phase and the positive sequence approaches, the rotor speed profile is seen to follow the same pattern irrespective of the method used. Moreover, as the load conditions are restored to their original values, the rotor speeds obtained from the two different techniques converge to the same value.

It is observed that the peak rotor speed deviations are slightly smaller in the case of the three-phase analysis as compared to the positive sequence analysis. It appears that the positive sequence analysis provides a more conservative solution. Other conservative solutions obtained by assuming a positive sequence model approximation for an unbalanced system have been reported in Distributed Series Reactance design studies, where the positive sequence approximation resulted in predicting 1000's of DSRs that were not needed [23]. Here the three-phase approach can provide more accurate limits on rotor speed deviations that might be useful in preventing the system from being considered as unacceptable.

Case I: Total Load Reduction = 6 MW (3 MW on load bus 5 and 6 each) at 0.95 leading pf.

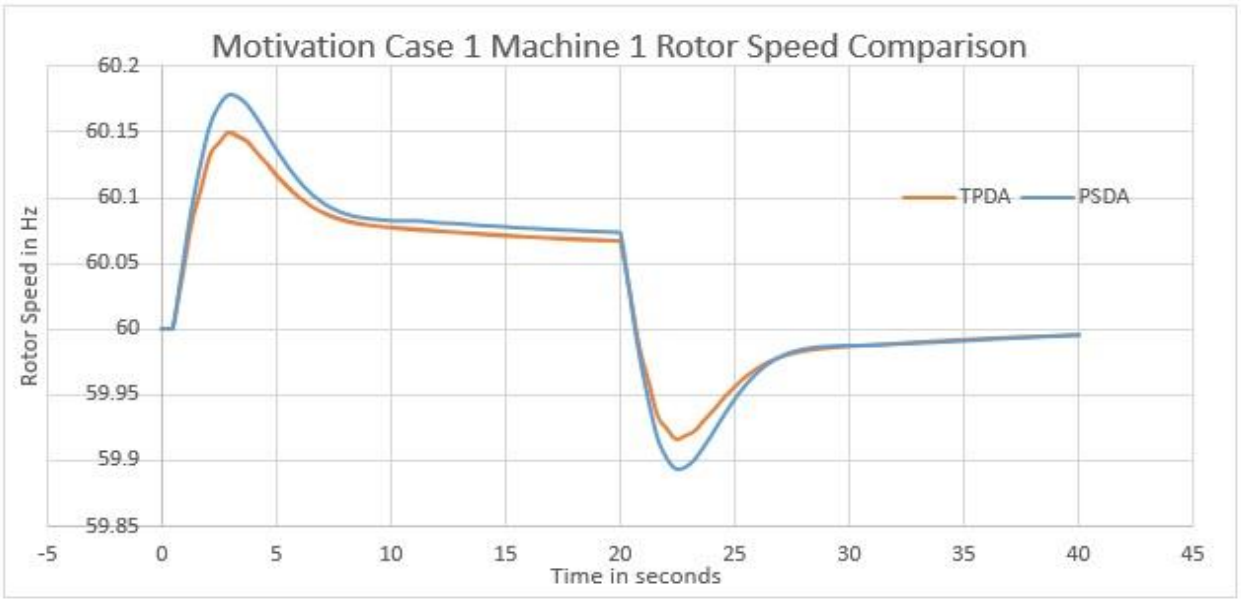


Figure 3.35: Rotor Speed comparison for generator 1 in case of 6 MW load reduction

Case II: Total Load Reduction = 30 MW (15 MW on load bus 5 and 6 each) at 0.95 leading pf.

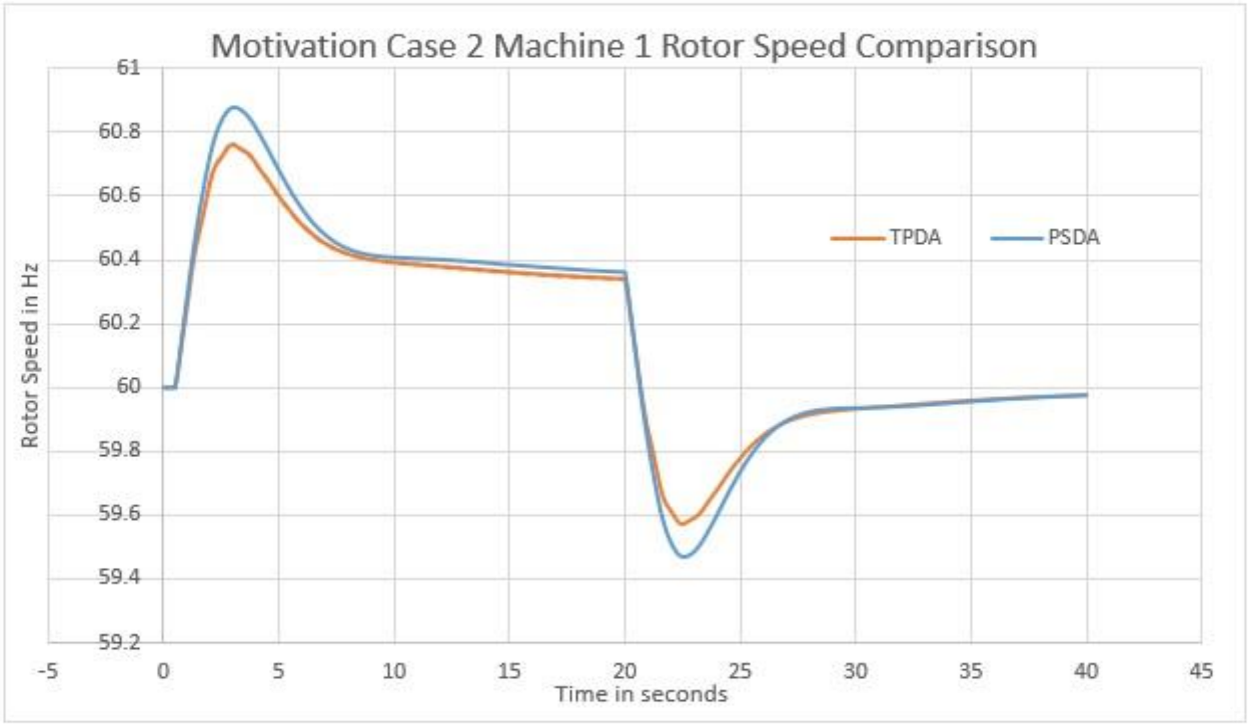


Figure 3.36: Rotor Speed comparison for generator 1 in case of 30 MW load reduction

Case III: Total Load Reduction = 60 MW (30 MW on load bus 5 and 6 each) at 0.95 leading pf.

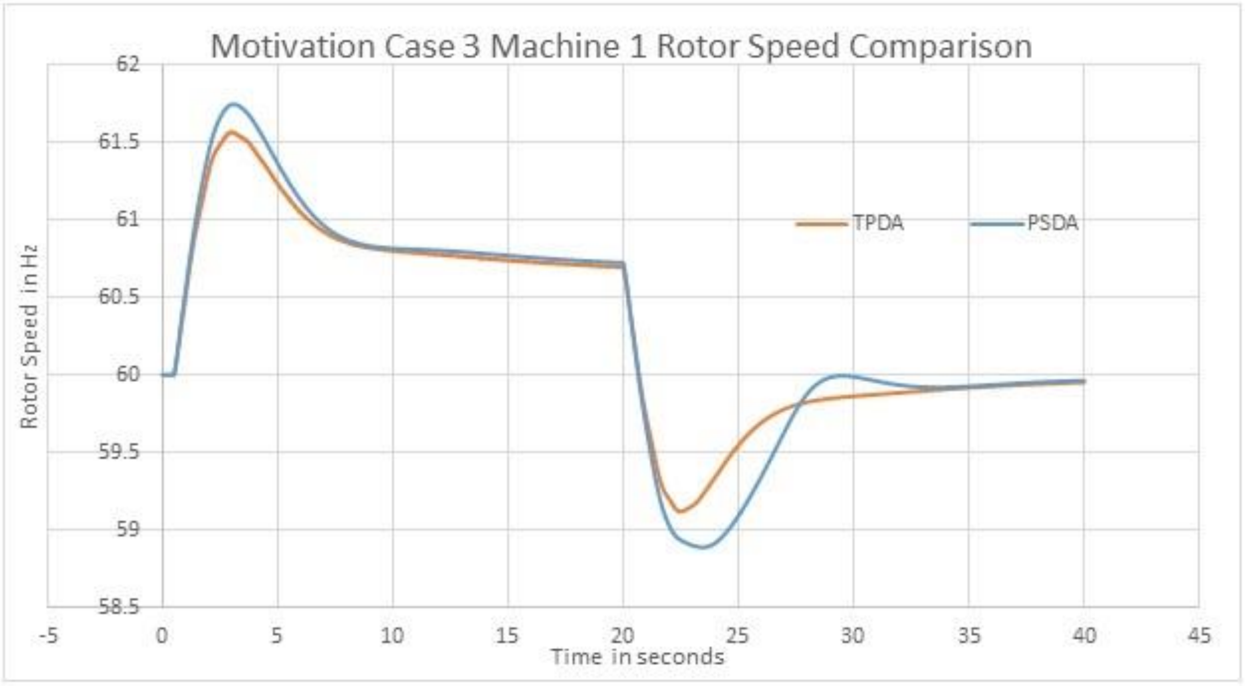


Figure 3.37: Rotor Speed comparison for generator 1 in case of 60 MW load reduction

Table 3.18: Rotor Speed Deviation Comparison for Case 1, 2 and 3

Case #	Maximum Rotor Speed Deviation (Hz)	$\frac{Max. Rotor\ Speed\ Deviation\ for\ Case\ j}{Max. Rotor\ Speed\ Deviation\ for\ Case\ 1}\ ; j = 1,2,3$
Case 1	0.1497	**1**
Case 2	0.7651	**5.110888**
Case3	1.5699	**10.48697**

<u>120 Hz Rotor Oscillations</u>

Finally, it is important to note that under unbalanced system conditions, the TPDA is successful in capturing 120 Hz rotor oscillations. Figures 3.38, 3.40 and 3.42 show 10 complete waves in a time span of 0.08333 seconds (1/12 sec.) for motivation cases 1, 2 and 3. Figures 3.39, 3.41 and 3.43 show PSDA results which fail to capture the 120 Hz oscillations.

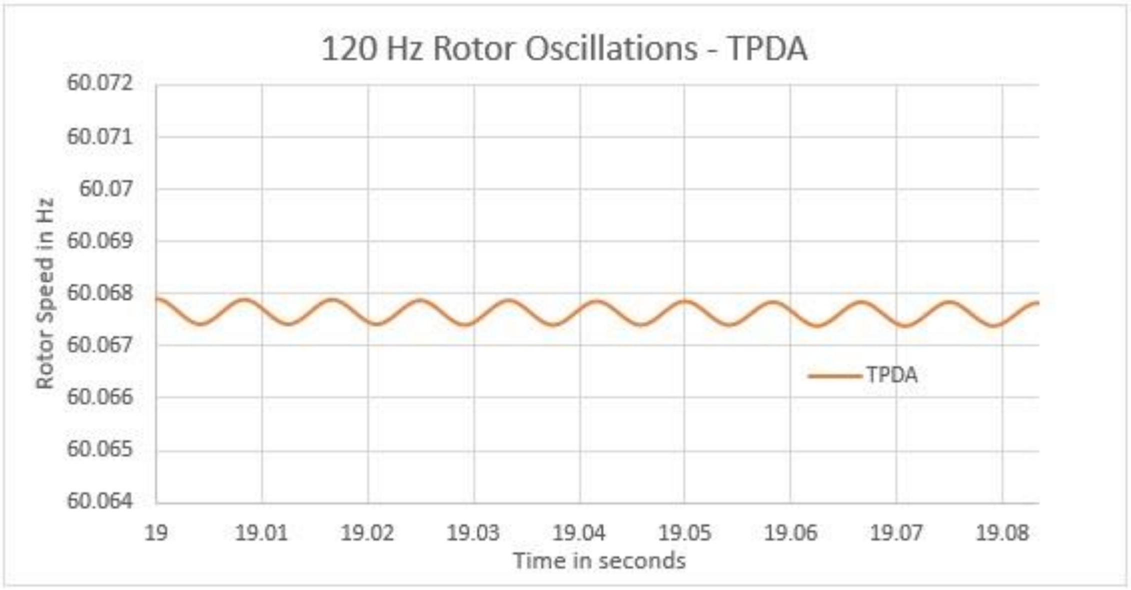


Figure 3.38: Motivation Case 1: 120 Hz oscillations under unbalanced system conditions

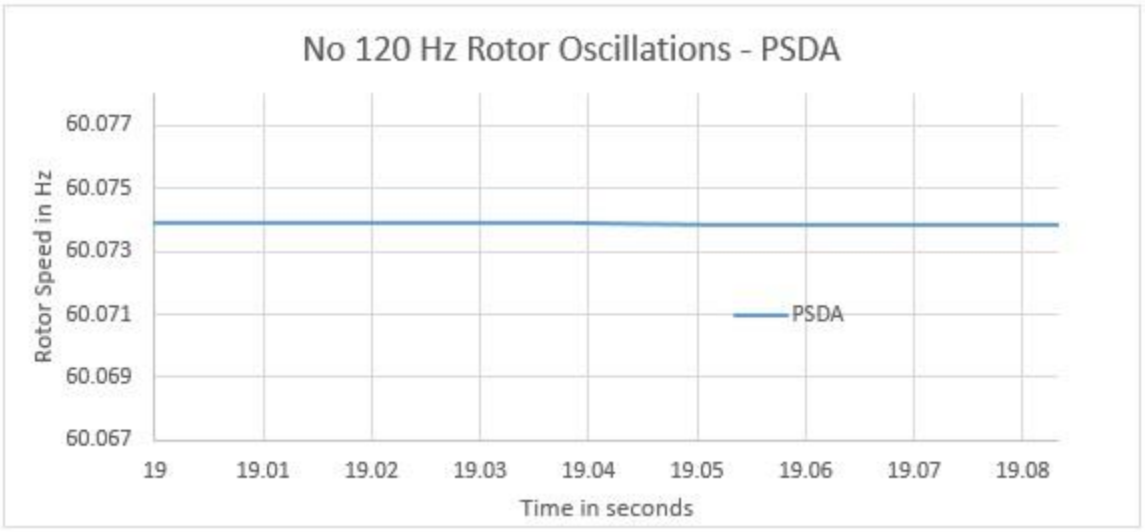


Figure 3.39: Motivation Case 1: No 120 Hz oscillations under unbalanced system conditions

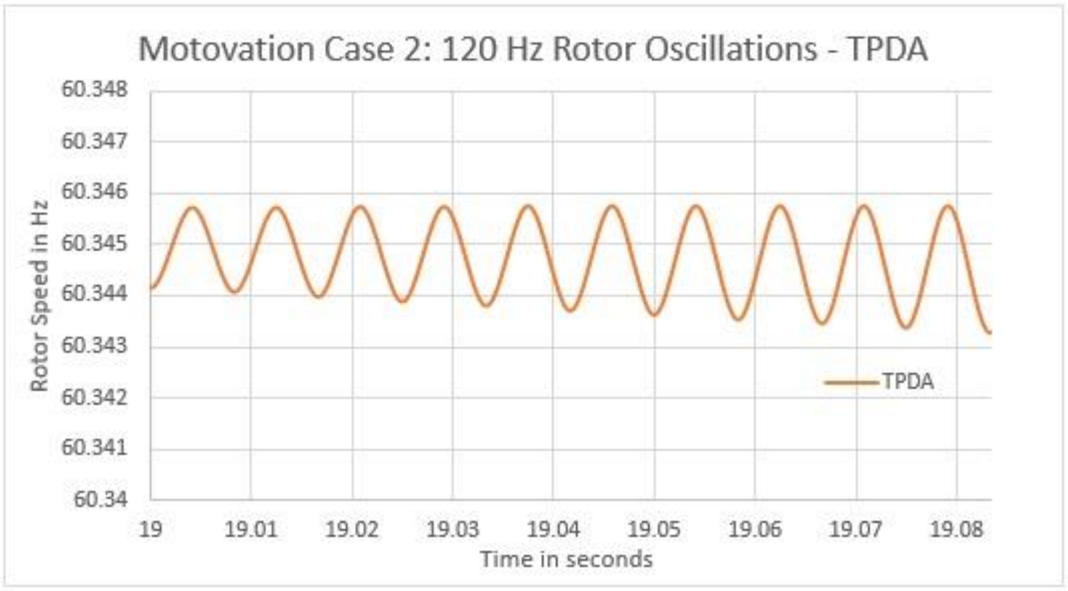


Figure 3.40: Motivation Case 2: 120 Hz oscillations under unbalanced system conditions

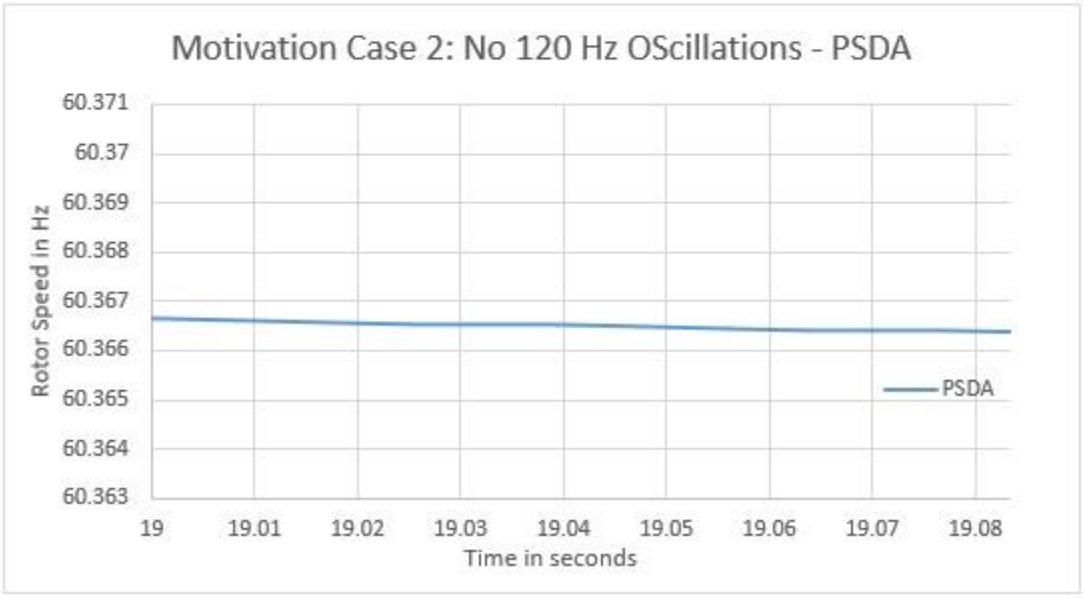


Figure 3.41: Motivation Case 2: No 120 Hz oscillations under unbalanced system conditions

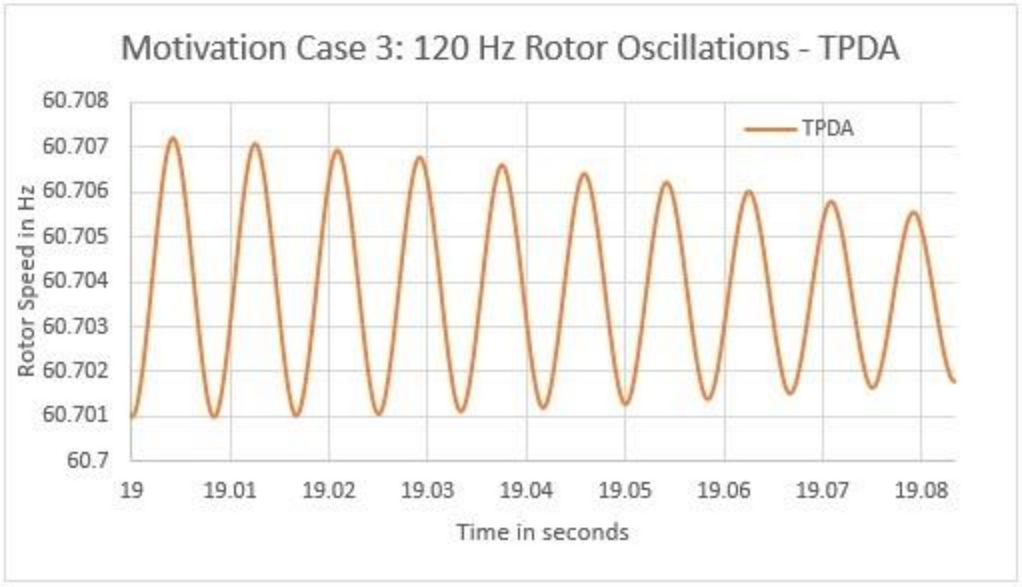


Figure 3.42: Motivation Case 3: 120 Hz oscillations under unbalanced system conditions

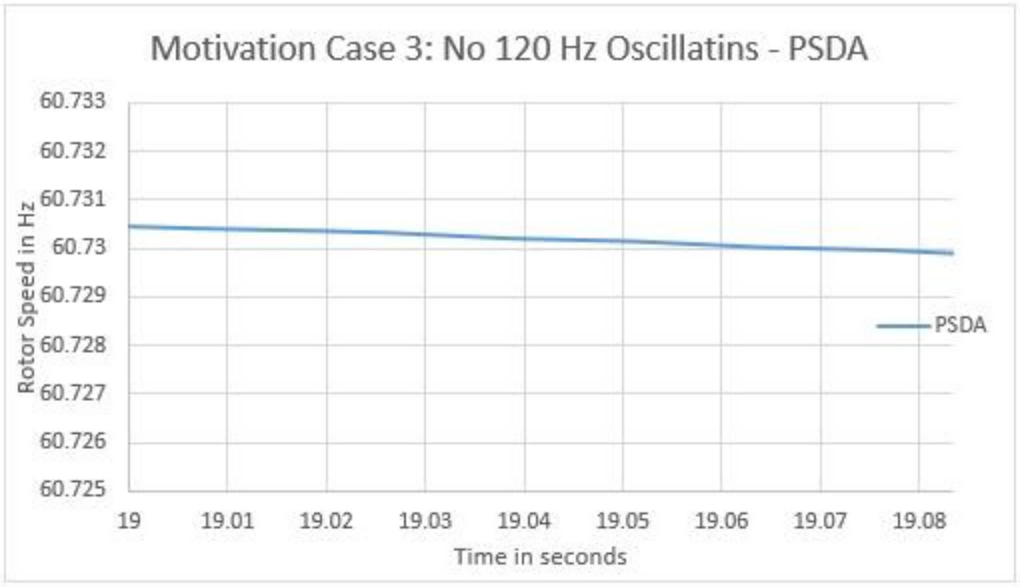


Figure 3.43: Motivation Case 3: No 120 Hz oscillations under unbalanced system conditions

Hence, we can see a large imbalance in currents and voltages throughout the electric network as well as large rotor speed deviations that grow proportionally with the size of disturbance. Further, TPDA successfully captures 120 Hz rotor oscillations. These oscillations and imbalances cannot be detected by software tools that assume the transmission system to be balanced. Determining such imbalances necessitates the need for three-phase analysis, especially with a growing penetration of DG generation in distribution circuits and an interest in such smart grid activities as Conservation Voltage Reduction. The dynamics introduced by varying levels of DG generation connected to the inherently unbalanced distribution system present a very important application of the multi-phase combined T&D dynamic analysis.

Chapter 4: Analysis of the Impact of Solar PV based DG on the Electric Grid using TPDA

This chapter presents an application of TPDA for analyzing the impact of DG on the electric grid. TPDA is used to study the impact of distribution level PV generation on an electric grid consisting of a transmission network and conventional generators. A quantitative analysis of this impact has been presented for varying levels of PV penetration. Real PV generation data with a sample rate of one second from a 2 megawatt PV generator has been used for the analysis, where a scaling factor is applied to the real data to scale the size of the PV generation for simulation purposes.

In order to add solar measurements to the model, a modification was made to the algorithm explained in chapter 2 (figure 2.13). Assume that an integration step size of 1/ (60*N) seconds or 1/N cycles was chosen for the partitioned explicit method, such that the exchange of data between power flow and the machine dynamics occurs N times every cycle (or 60*N times every second). So to add three-phase solar measurements that update every second, the measurements are read from the database every (60*N) cycles. The flow chart for the modified algorithm is shown in figure 4.1 with the change in the original TPDA algorithm highlighted in yellow.

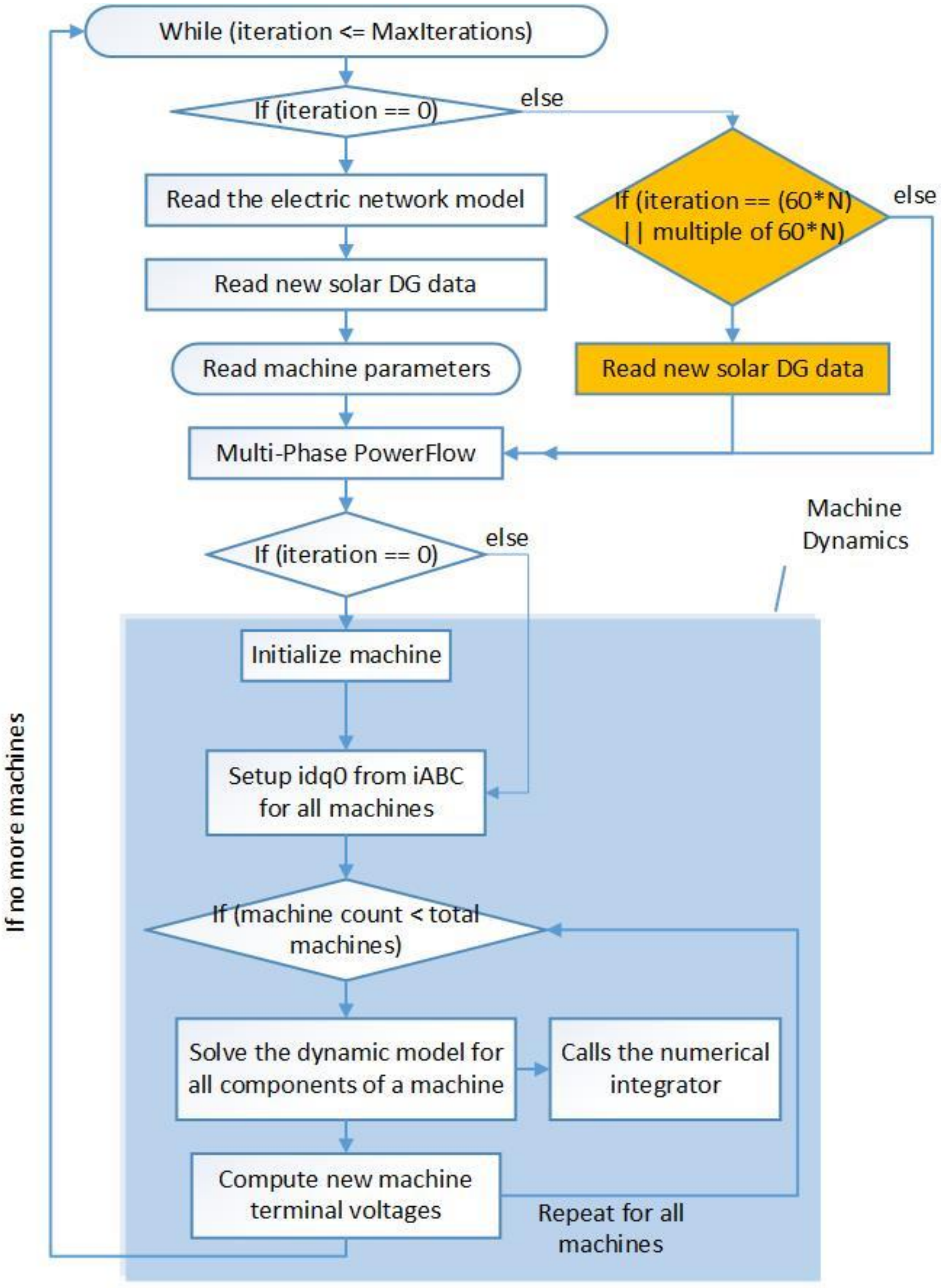
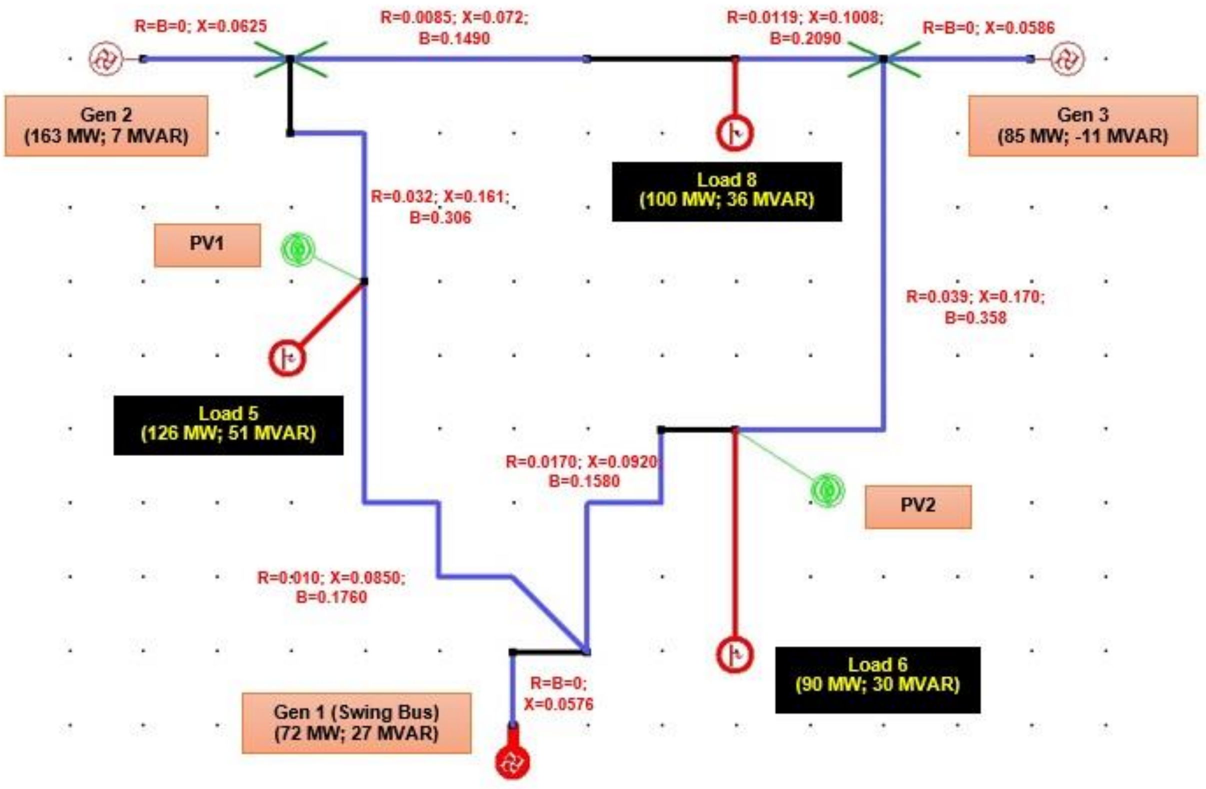

While (iteration <= MaxIterations)
If (iteration == 0)
else
Read the electric network model
If (iteration == (60*N) || multiple of 60*N)
else
Read new solar DG data
Read new solar DG data
Read machine parameters
Multi-Phase PowerFlow
Machine Dynamics
If (iteration == 0)
else
Initialize machine
Setup idq0 from iABC for all machines
If (machine count < total machines)
Solve the dynamic model for all components of a machine
Calls the numerical integrator
Compute new machine terminal voltages
Repeat for all machines
If no more machines

Figure 4.1: Modified TPDA algorithm to allow loading solar measurements

The remainder of this chapter is organized as follows. Section 4.1 introduces the electric network, solar measurements and test cases used for analysis. Results assessing the impact of PV generation on power flow, voltages and currents throughout the entire grid have been presented in sections 4.2, 4.3 and 4.4 respectively. Finally section 4.5 evaluates the effects of increasing PV generation on system frequency/rotor speeds.

4.1 Electric Network, Solar Measurements and Simulation Cases

The electric network shown in Figure 4.2 is used for all subsequent analysis presented in this chapter. It is a slightly modified version of the network used for analysis in chapter 3 (figure 3.1). Table 4.1 summarizes the three machine types (with generator, exciter and governor types) used in the simulations. Solar DG penetration is modeled to be occurring on distribution circuits connected to load busses 5 and 6. This is done by adding solar measurements to load busses 5 and 6 (shown as PV1 and PV2 in figure 4.2).



R=B=0; X=0.0625
R=0.0085; X=0.072; B=0.1490
R=0.0119; X=0.1008; B=0.2090
R=B=0; X=0.0586
Gen 2 (163 MW; 7 MVAR)
Gen 3 (85 MW; -11 MVAR)
Load 8 (100 MW; 36 MVAR)
R=0.032; X=0.161; B=0.306
PV1
R=0.039; X=0.170; B=0.358
Load 5 (126 MW; 51 MVAR)
R=0.0170; X=0.0920; B=0.1580
PV2
R=0.010; X=0.0850; B=0.1760
Load 6 (90 MW; 30 MVAR)
R=B=0; X=0.0576
Gen 1 (Swing Bus) (72 MW; 27 MVAR)

Figure 4.2: Electric Network used for three-phase dynamic analysis of PV generation

Table 4.1: Types of machines used for the presenting and comparing test case results

	Generator	Exciter	Governor
Gen 1	GENROU	AC7B	GGOV1
Gen 2	GENROU	AC7B	GGOV1
Gen 3	GENROU	AC7B	-

Figures 4.3 and 4.4 show the solar profiles used at PV1 and PV2, respectively. The figures show three-phase PV measurements over a period of 40 seconds, which is the simulation time used for all test cases in this chapter. Solar data is seen to be varying every second. Furthermore, all the three phases are seen to have nearly equal solar generation, i.e., the imbalance in the three-phase solar dataset used for this study is very small. It must be noted that these figures show the solar PV output corresponding to one test case (TC1 or the base case) where the maximum overall solar generation in the network is less than 15 MW on a total system load of 315 MW. In other words, these figures represent the base case for which the maximum solar generation is less than 5% of the total system load.

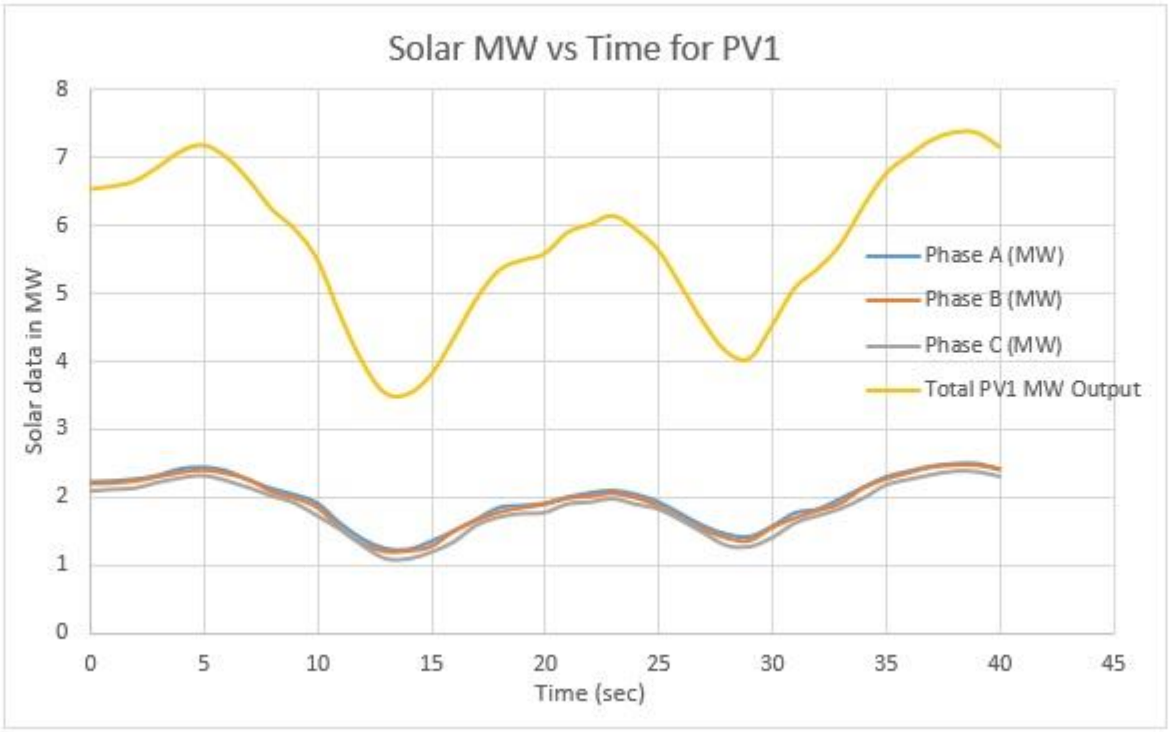


Figure 4.3: Base Case Solar Profile used at PV1

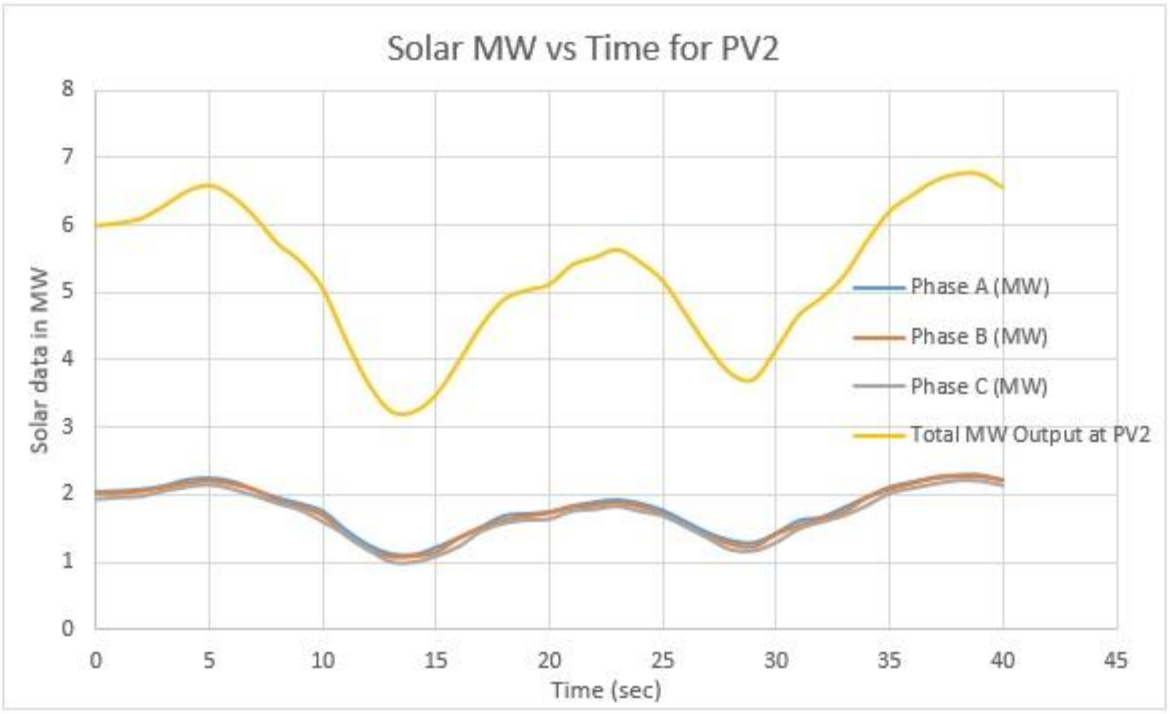


Figure 4.4: Base Case Solar Profile used at PV2

A total of ten test cases were simulated: TC1 to TC10. Increased PV penetration is achieved by appropriately scaling the solar generation dataset represented by figures 4.2 and 4.3. For example, TC2 has double the solar generation present in TC1. Since the solar measurements at PV1 and PV2 are just multiplied by a factor of 2 to achieve this, it represents doubling of solar generation at each PV site in the grid. Similar scaling is used as we move linearly in ten steps from TC1 to TC10 (10 times TC1). PV penetration is defined here as

$$PV\ Penetration = \frac{Maximum\ Total\ PV\ Genetraion}{Total\ System\ Load} * 100 \ldots (1)$$

Thus, the PV penetration level goes from 5% in TC1 to 50% in TC10. All ten test cases and their corresponding penetration levels are specified in table 4.2.

Table 4.2: Solar PV Penetration for all test cases

Test Case#	$PV\ Penetration = \frac{Max.Total\ PV\ Genetraion}{Total\ System\ Load} * 100$
TC1 (Base Case)	5%
TC2	10%
TC3	15%
TC4	20%
TC5	25%
TC6	30%
TC7	35%
TC8	40%
TC9	45%
TC10	50%

4.2 Power Flow Analysis

Reverse Power Flow

The model used for analysis in this chapter (figure 4.1) does not incorporate any protection elements or schemes that would prevent reverse power flow in the network – from load bus to generator bus. Thus, we first analyze the direction of power flow for various levels of PV penetration. For power to flow from generator bus to load bus, the voltage at the generator bus must lead the voltage at the load bus by some angle δ, *where*

$$\delta = \theta_{gen\ bus} - \theta_{load\ bus} \dots (2)$$

We note this angle δ corresponding to all three generator busses in the system. As the level of PV penetration increases, the value of δ for generator 1 is seen to reduce and eventually become negative. Figure 4.4 plots the value of δ for generator 1 at DG penetration levels ranging from 10% to 60%. The 60% penetration level has not been included in table 4.2 and it is not addressed in other simulations presented in this chapter, but is included in the reverse power flow work here. Table 4.3 notes the maximum and minimum values of δ (in degrees) at different penetration levels, and it can be seen that at about 45% PV penetration, there are certain times when machine 1, connected to the swing bus, starts behaving like a motor instead of a generator. On increasing the PV generation further, δ is seen to shift down (become more negative), signifying greater power flow from load bus to generator 1 bus. It is important to note that at 45%, 50% or 60% PV penetration load busses 5 and 6 still provide generation. Thus, at the high penetration levels, the load is being completely served by generators 2 and 3 (still acting as generators), while generator 1 (connected to the swing bus) is seen to compensate for the high PV generation by behaving like a motor. If protection devices were modeled in the system, they would trip generator 1 as soon as δ becomes zero, causing large transients in the system. If generator tripping is to be avoided, it necessitates the formulation of an optimized control strategy that can dynamically control generation dispatch taking into account the effects of DG generation. Hence, this analysis is successful in identifying such reverse power flow scenarios and could subsequently be used to test control algorithms. Alternatively, protection devices/schemes could be modeled in the electric network and TPDA could be used for the three-phase analysis of the transients resulting from generator tripping as a consequence of high PV generation.

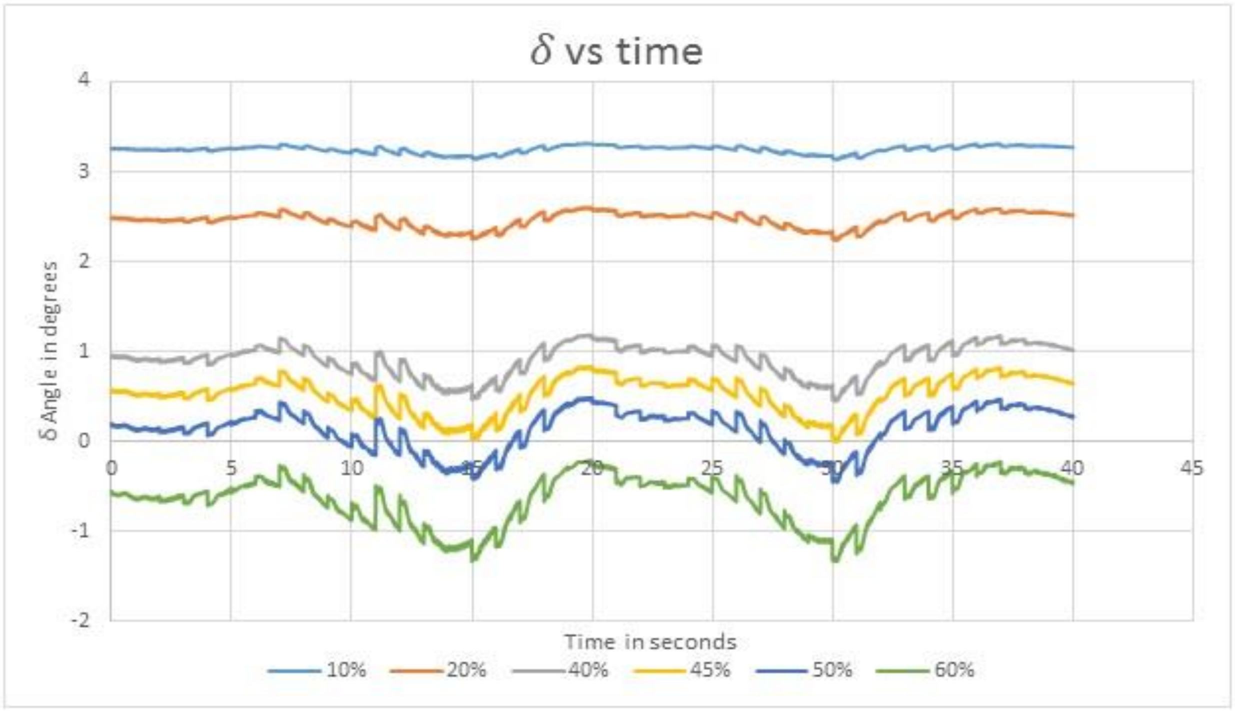


Figure 4.5: Generator 1 δ Angle vs time for varying PV penetration levels

Table: 4.3: Maximum and minimum δ Angles for different PV penetration levels

PV Penetration	Max δ	Min δ
10%	3.313095	3.136452
20%	2.601412	2.237897
40%	1.184671	0.447171
45%	0.834221	-0.00455
50%	0.486271	-0.45227
60%	-0.21072	-1.33737

4.3 Impact of PV Generation on Transmission Voltages

Two measures are used to analyze voltages as PV penetration increases from TC1 to TC10. These measures are: Peak Deviation (PD) and Peak Deviation Ratio (PDR) which are defined as:

$$PD = Maximum - Minimum\ Voltage \ldots.\ (3)$$

$$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10 \ldots (4)$$

As we go from TC1 to TC10, the rate of increase in PDR is also analyzed by comparing increases in PDR to the increase in PV Penetration Ratio (PVR), where PVR is defined as

$$PVR = \frac{PV\ Penetration\ for\ TCi}{PV\ Penetration\ for\ TC1}\ ; i = 1,..,10 \ldots (5)$$

4.3.1 Impact on Machine Terminal Voltages

4.3.1.1 Generator 1 Terminal Voltage

Figures 4.6 and 4.7 show the three-phase voltage profiles at generator 1 terminals for TC1 and TC10, respectively. The system has experienced no fault/disturbance. Yet we see that the voltage profiles are not even close to being steady. The voltage is seen to undergo fast variations, changing every iteration. This volatility in generator 1 terminal voltage is magnified as we move from TC1 to TC10.

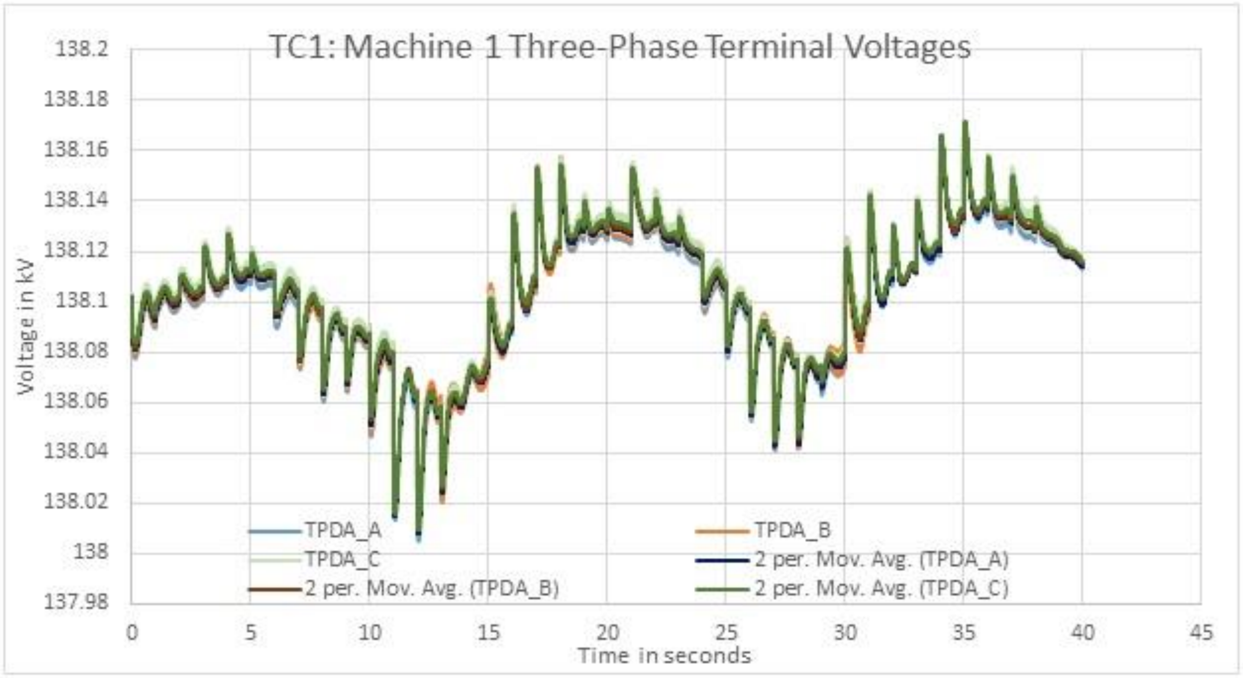


Figure 4.6: Three-Phase Generator 1 Terminal Voltage for TC1

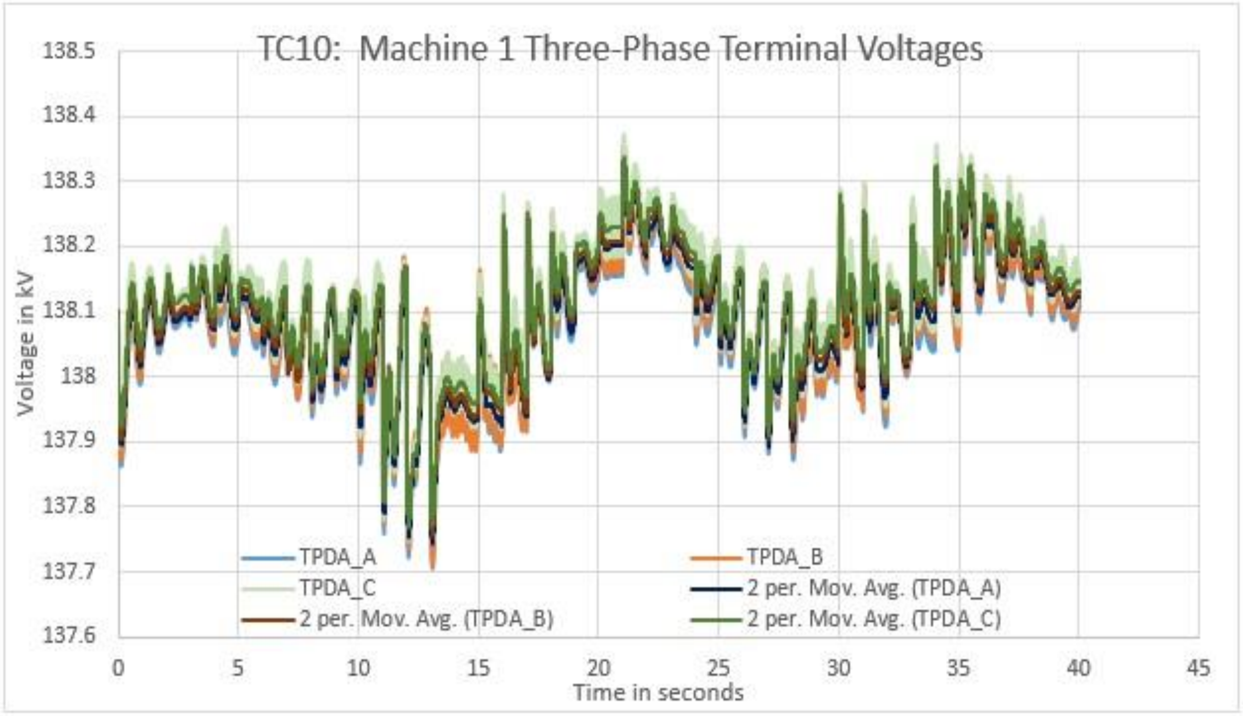


Figure 4.7: Three-Phase Generator 1 Terminal Voltage for TC10

Table 4.4 shows the maximum and minimum generator 1 terminal voltages for all three phases as PV penetration increases from TC1 to TC10. The peak deviation (PD) in generator 1 terminal voltage is calculated as the difference in the maximum and minimum generator 1 terminal voltage. Table 4.5 shows this peak deviation in all three phases for the ten test cases: TC1 to TC10. This table also shows the peak deviation ratio (PDR) as defined earlier in this section. We can see that as the PV output doubles from TC1 to TC2, PD in generator 1 terminal voltage becomes almost 1.9 times the PD in TC1 in all three phases. However, as the solar output increases further, PD is not seen to increase at the same rate. The rate of increase in PD is seen to fall as we move from TC1 towards TC10. This is shown in figure 4.8 which plots PDR against the PV Penetration Ratio (PVR).

It must be noted that PDR is seen to fall for TC9 and TC10 which is when machine 1 is oscillating between generation and motor modes. Before we reach the results as shown for TC9 and TC10, protection devices would trip machine 1 and the voltage profiles will completely change due to the transients resulting from the generator tripping.

Table 4.4: Maximum & minimum generator 1 terminal voltage over all test cases

TC#	Max kV at Generator 1 Terminal			Min kV at Generator 1 Terminal		
	A	B	C	A	B	C
TC1	138.1696	138.1705	138.172	138.0049	138.0078	138.009
TC2	138.2299	138.2321	138.2361	137.9196	137.9201	137.9244
TC3	138.2864	138.2887	138.2944	137.8499	137.8519	137.8541
TC4	138.3143	138.3171	138.3229	137.7685	137.7783	137.7829
TC5	138.329	138.3358	138.3466	137.7317	137.7344	137.7431
TC6	138.3575	138.3632	138.3739	137.7073	137.714	137.7166
TC7	138.3511	138.3571	138.3687	137.6749	137.6905	137.6968
TC8	138.3453	138.3503	138.3726	137.6678	137.6729	137.6833
TC9	138.3555	138.3613	138.3735	137.6936	137.6914	137.7189
TC10	138.3515	138.3577	138.3716	137.7057	137.7088	137.7426

Table 4.5: PD and PDR for generator 1 terminal voltage over all test cases

TC#	Gen 1 Terminal Voltage Peak Deviation (PD) in kV= Max – Min			$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10$		
	A	B	C	A	B	C
TC1	0.164676	0.162714	0.162973	1	1	1
TC2	0.310344	0.31198	0.311628	1.884573	1.917352	1.912145
TC3	0.436465	0.43675	0.440335	2.650447	2.684157	2.701889
TC4	0.545825	0.538874	0.539978	3.314539	3.311786	3.313297
TC5	0.597342	0.601367	0.603526	3.627377	3.695853	3.703227
TC6	0.650176	0.649247	0.657216	3.948213	3.990111	4.032668
TC7	0.676213	0.666553	0.671829	4.106324	4.09647	4.122333
TC8	0.677519	0.677419	0.689248	4.114255	4.16325	4.229216
TC9	0.661807	0.669951	0.654643	4.018843	4.117353	4.01688
TC10	0.645758	0.648919	0.628989	3.921385	3.988096	3.859468

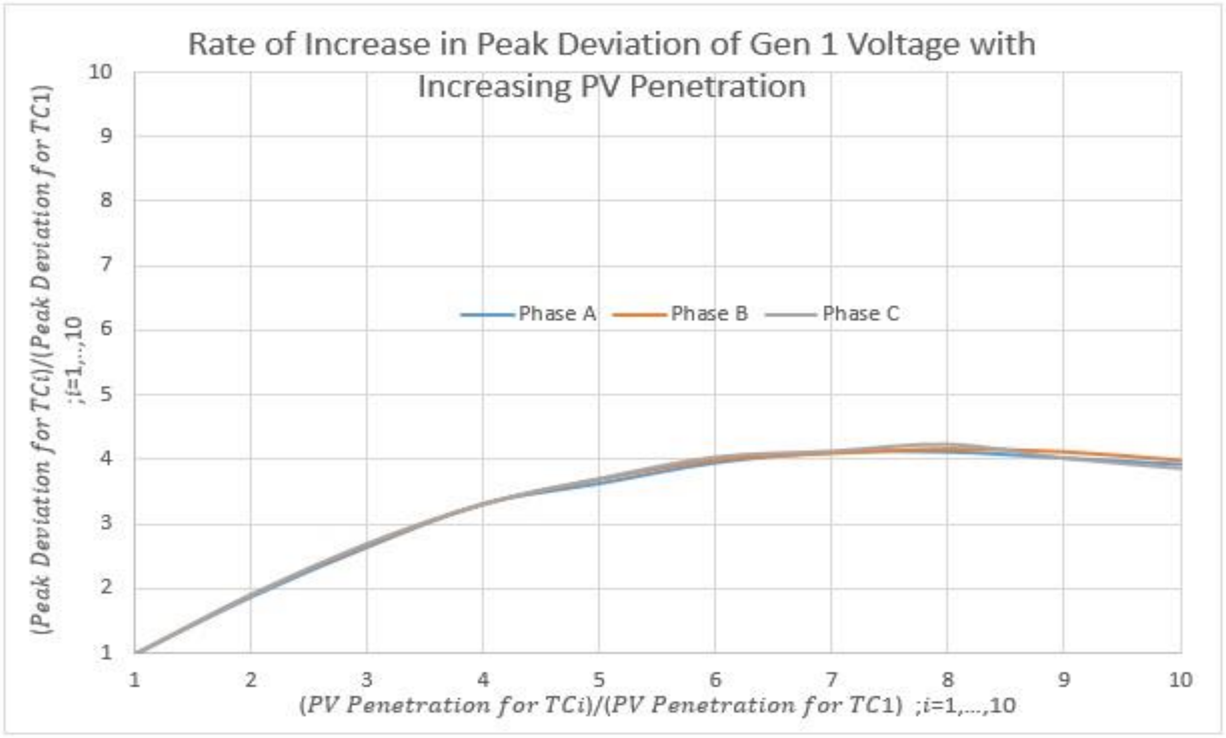


Figure 4.8: Comparing increase in PDR for generator 1 terminal voltage to the increase in PVR

4.3.1.2 Generator 2 Terminal Voltage

Machine 2 is observed to function in the generating mode throughout the ten test cases: TC1 to TC10. Figures 4.9 and 4.10 show an increasingly volatile generator 2 terminal voltage as PV output increases from TC1 to TC10. Tables 4.6 and 4.7 show a detailed analysis of the maximum voltage, minimum voltage, PD and PDR at generator 2 terminals with increasing solar penetration. In this case, PDR is seen to continue increasing throughout the ten cases as PVR goes from 1 to 10. This is plotted in figure 4.11. It can be seen that the increase in PDR is almost proportional to PVR for low levels of PV penetration. However, as PV penetration keeps increasing, the observed increase in PDR is not as high as PVR (less than a proportional increase).

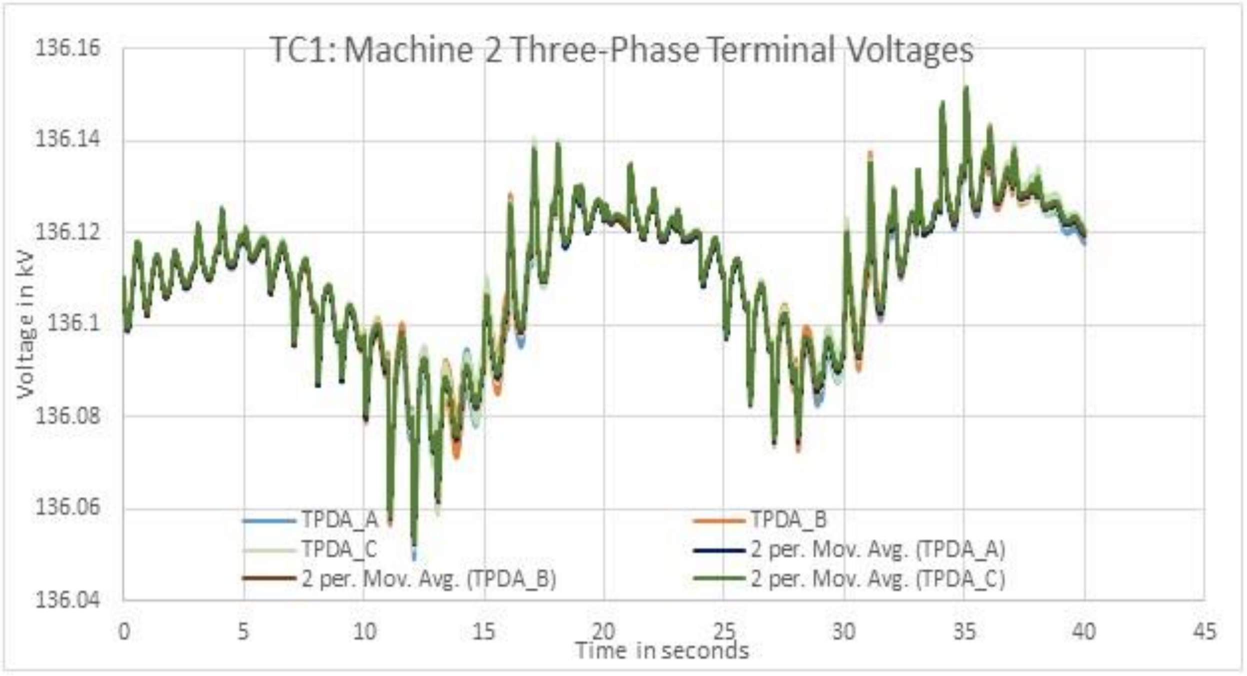


Figure 4.9: Three-Phase Generator 2 Terminal Voltage for TC1

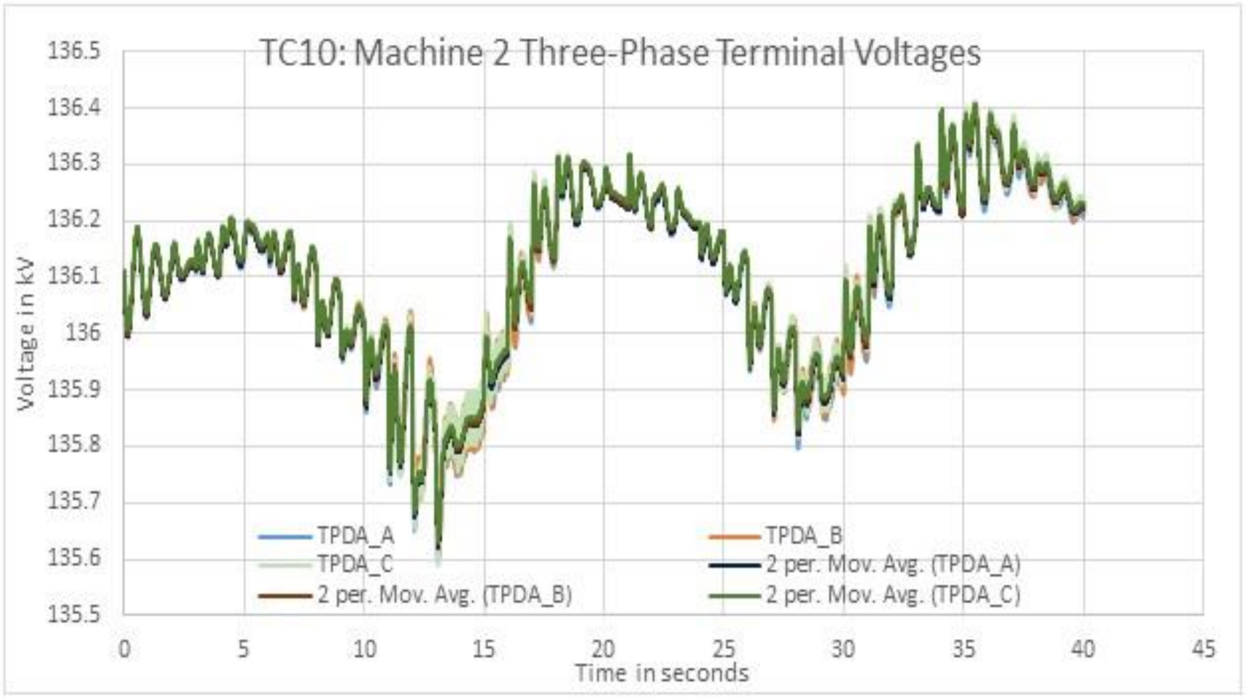


Figure 4.10: Three-Phase Generator 2 Terminal Voltage for TC10

Table 4.6: Maximum & minimum generator 2 terminal voltage over all test cases

TC#	Max kV at Generator 2 Terminal			Min kV at Generator 2 Terminal		
	A	B	C	A	B	C
TC1	136.1516	136.1517	136.1518	136.049	136.0514	136.051
TC2	136.1886	136.1883	136.191	135.995	135.9955	135.9928
TC3	136.2256	136.2281	136.231	135.9444	135.942	135.9472
TC4	136.254	136.2567	136.2571	135.8798	135.8905	135.8868
TC5	136.2846	136.2844	136.2857	135.8492	135.8479	135.8431
TC6	136.3098	136.3087	136.3167	135.8004	135.7989	135.8116
TC7	136.3325	136.3319	136.3386	135.7528	135.7715	135.7638
TC8	136.3492	136.3531	136.3532	135.7242	135.7157	135.7124
TC9	136.3788	136.3755	136.3829	135.6695	135.6545	135.6601
TC10	136.4066	136.4116	136.4116	135.5913	135.6064	135.5918

Table 4.7: PD and PDR for generator 2 terminal voltage over all test cases

TC#	Gen 2 Terminal Voltage Peak Deviation (PD) in kV= Max – Min			$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10$		
	A	B	C	A	B	C
TC1	0.102645	0.10031	0.100787	1	1	1
TC2	0.193638	0.192791	0.198152	1.886483	1.921952	1.966047
TC3	0.281185	0.286077	0.283853	2.739393	2.851929	2.816365
TC4	0.374129	0.366245	0.37032	3.644883	3.651131	3.674283
TC5	0.435441	0.436457	0.442587	4.242204	4.351082	4.39131
TC6	0.509393	0.509735	0.505038	4.962667	5.081597	5.010944
TC7	0.579658	0.560358	0.574851	5.647211	5.586263	5.703622
TC8	0.625021	0.637379	0.640797	6.089152	6.354092	6.357933
TC9	0.709328	0.721023	0.722799	6.910497	7.187947	7.17155
TC10	0.815363	0.805148	0.819876	7.943524	8.026598	8.13474

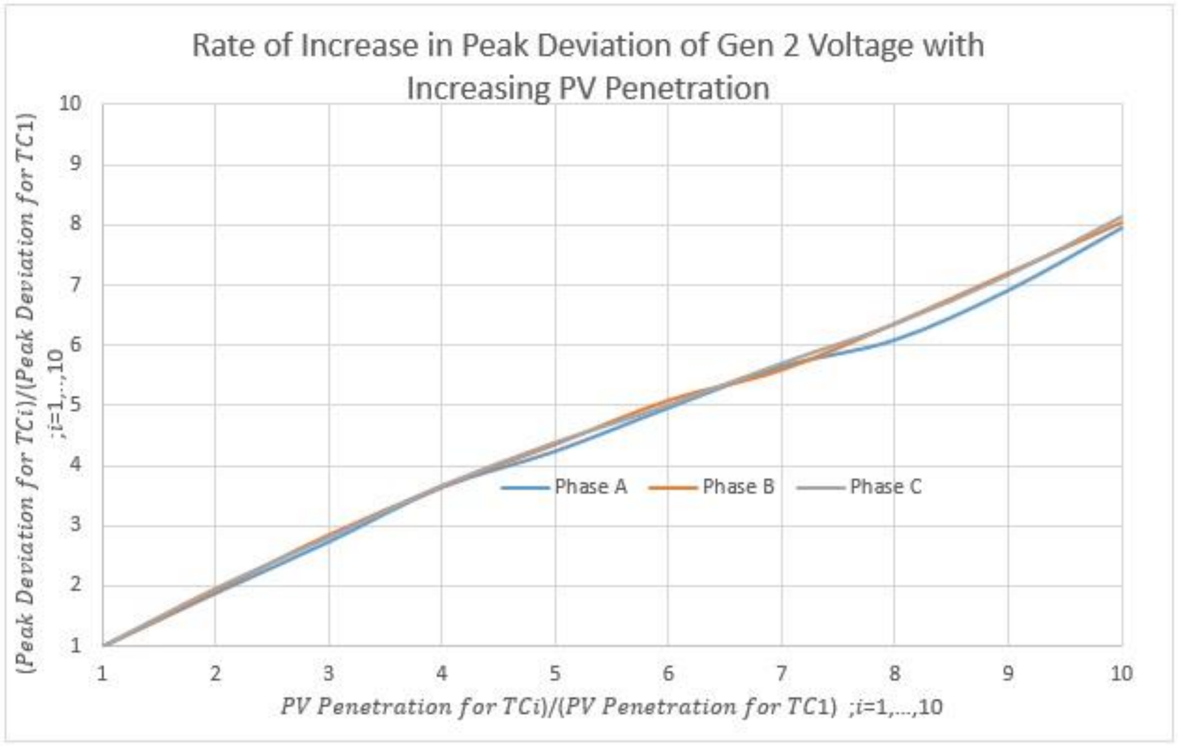


Figure 4.11: Comparing increase in PDR for generator 2 terminal voltage to the increase in PVR

It should be noted that generator 3 terminal voltage results are very similar to those obtained for generator 2.

4.3.2 Impact on Load Bus Voltages

Figures 4.12, 4.13, 4.15, 4.16, 4.18, 4.19 show increasing volatility in voltage profiles at load busses 8, 5 and 6 as PV output increases from TC1 to TC10. Again, dynamics introduced by variable PV output is seen to affect all load busses in the electric network, even load busses that do not have PV generation. Tables 4.8-4.13 show the quantitative analysis of the three-phase voltages at these load busses, by reporting the maximum voltage, minimum voltage, PD and PDR over the varying levels of PV penetration.

Again, it is observed that initially, for low levels of PV penetration, the rate of increase in peak voltage deviations at all load busses is almost proportional to the increase in PV penetration. However, this increase in PDR slows down compared to the increase in PVR as PV output keeps increasing. Figures 4.14, 4.17 and 4.20 plot PDR against PVR over all ten test cases for the three load busses in the system. Peak voltage deviations at all the load busses are seen to be about 6.5-7 times their original values (for TC1) as PV output becomes 10 times its original value (TC1).

4.3.2.1 Load Bus 8 Voltage

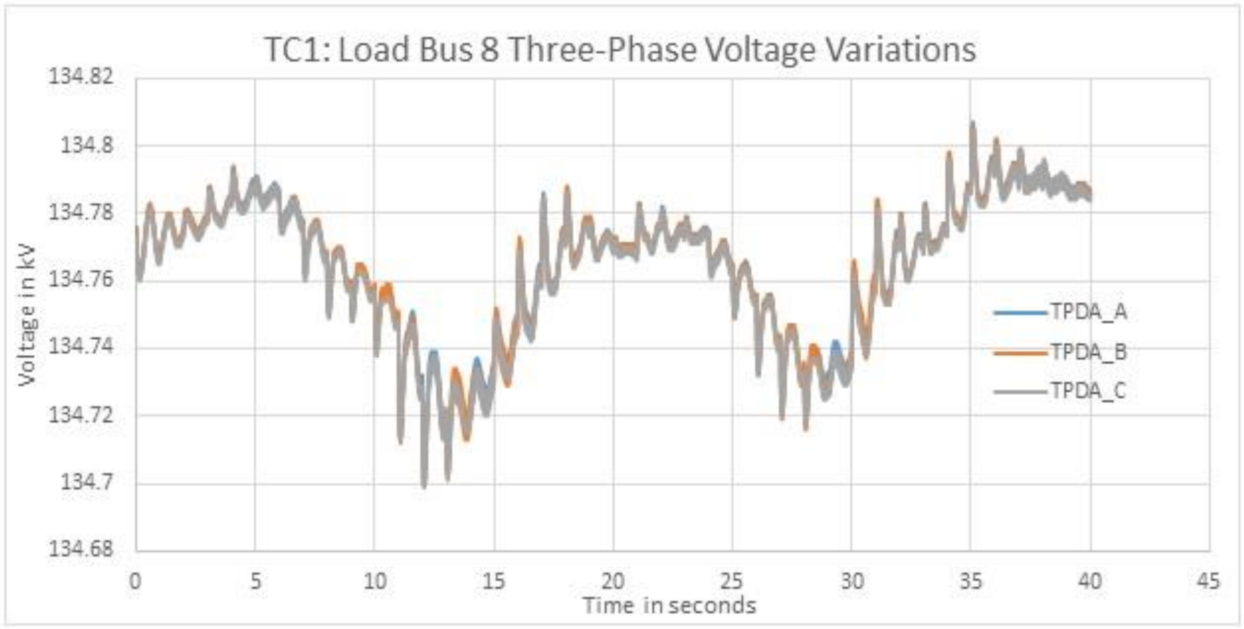


Figure 4.12: Load Bus 8 Three-Phase Voltage for TC1

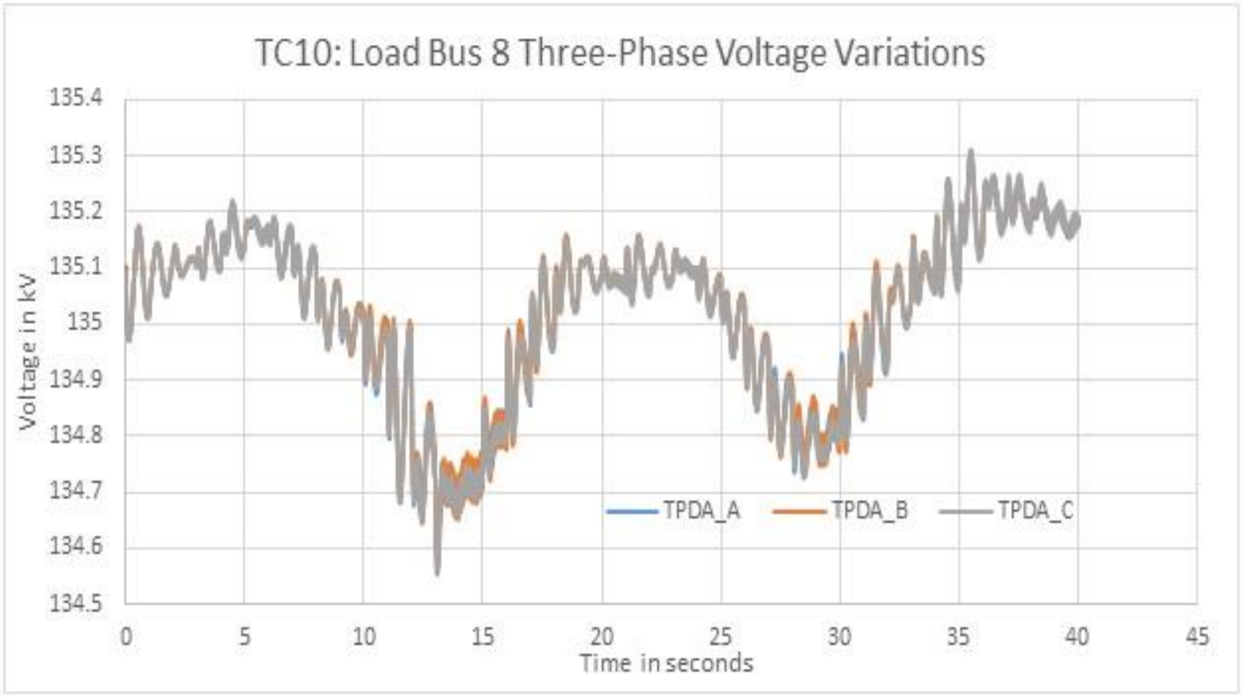


Figure 4.13: Load Bus 8 Three-Phase Voltage for TC10

Table 4.8: Maximum & minimum load bus 8 voltage over all test cases

TC#	Max kV at Load bus 8			Min kV at Load bus 8		
	A	B	C	A	B	C
TC1	134.807	134.806	134.805	134.699	134.7	134.699
TC2	134.88	134.881	134.881	134.681	134.679	134.678
TC3	134.956	134.955	134.956	134.668	134.664	134.667
TC4	135.012	135.009	135.008	134.642	134.643	134.645
TC5	135.063	135.063	135.059	134.635	134.629	134.627
TC6	135.117	135.115	135.113	134.627	134.641	134.628
TC7	135.166	135.166	135.16	134.621	134.621	134.629
TC8	135.201	135.203	135.211	134.625	134.606	134.62
TC9	135.252	135.248	135.257	134.59	134.575	134.579
TC10	135.31	135.308	135.309	134.553	134.565	134.555

Table 4.9: PD and PDR for load bus 8 voltage over all test cases

TC#	Load Bus 8 Voltage Peak Deviation (PD) in kV = Max – Min			$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10$		
	A	B	C	A	B	C
TC1	0.108	0.106	0.106	1	1	1
TC2	0.199	0.202	0.203	1.842593	1.90566	1.915094
TC3	0.288	0.291	0.289	2.666667	2.745283	2.726415
TC4	0.37	0.366	0.363	3.425926	3.45283	3.424528
TC5	0.428	0.434	0.432	3.962963	4.09434	4.075472
TC6	0.49	0.474	0.485	4.537037	4.471698	4.575472
TC7	0.545	0.545	0.531	5.046296	5.141509	5.009434
TC8	0.576	0.597	0.591	5.333333	5.632075	5.575472
TC9	0.662	0.673	0.678	6.12963	6.349057	6.396226
TC10	0.757	0.743	0.754	7.009259	7.009434	7.113208

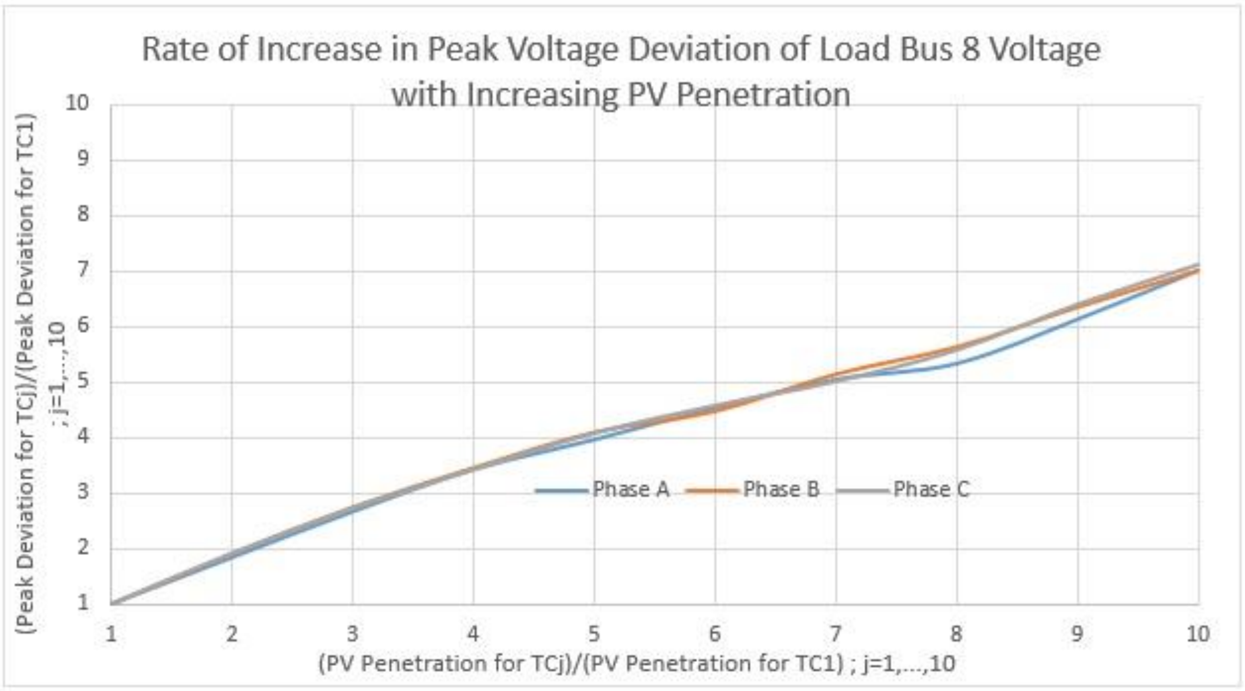


Figure 4.14: Comparing increase in PDR for load bus 8 voltage to the increase in PVR

4.3.2.2 Load Bus 5 Voltage

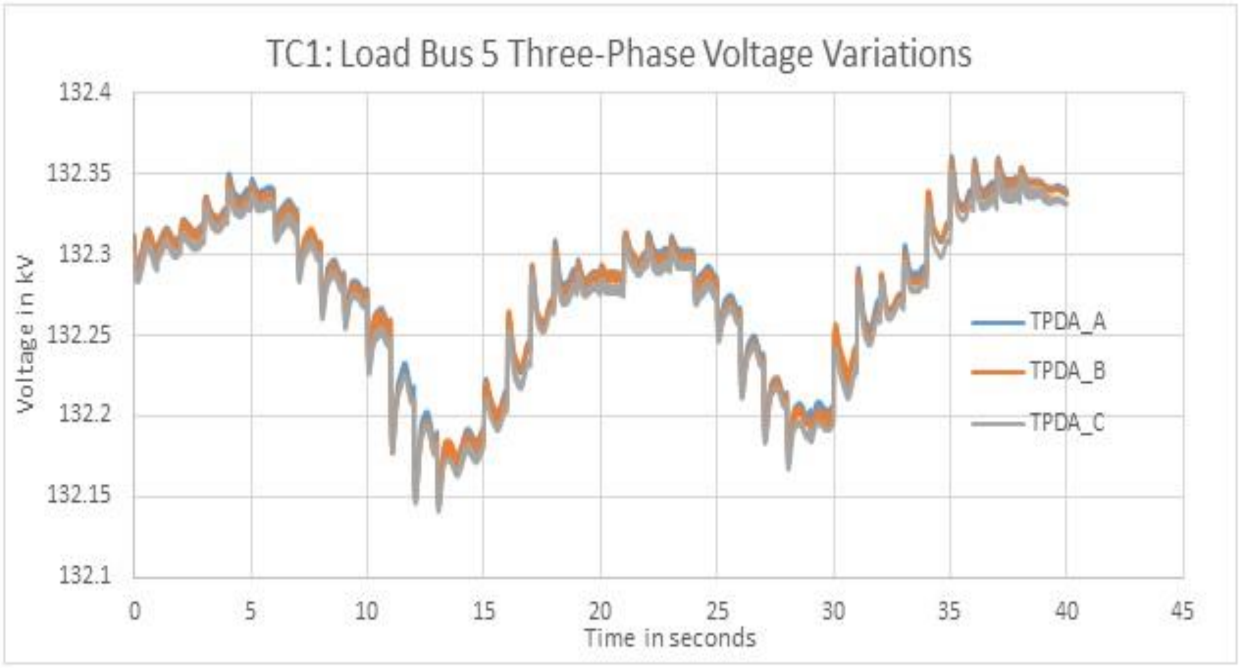


Figure 4.15: Load Bus 5 Three-Phase Voltage for TC1

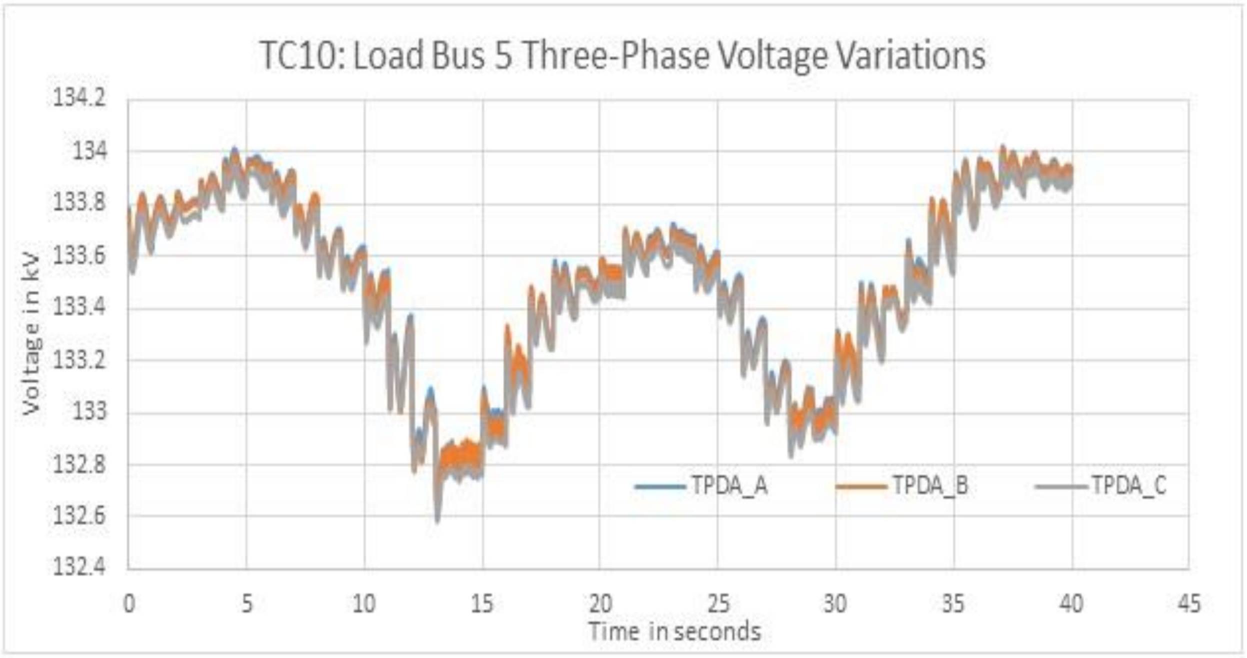


Figure 4.16: Load Bus 5 Three-Phase Voltage for TC10

Table 4.10: Maximum & minimum load bus 5 voltage over all test cases

TC#	Max kV at Load bus 5			Min kV at Load bus 5		
	A	B	C	A	B	C
TC1	132.361	132.358	132.352	132.148	132.143	132.141
TC2	132.596	132.593	132.582	132.185	132.183	132.171
TC3	132.829	132.823	132.809	132.232	132.223	132.207
TC4	133.034	133.032	133.012	132.278	132.252	132.242
TC5	133.227	133.228	133.203	132.316	132.297	132.287
TC6	133.428	133.427	133.394	132.385	132.381	132.35
TC7	133.577	133.565	133.532	132.457	132.434	132.404
TC8	133.729	133.717	133.677	132.532	132.487	132.468
TC9	133.88	133.868	133.828	132.566	132.532	132.518
TC10	134.024	134.014	133.976	132.624	132.606	132.583

Table 4.11: PD and PDR for load bus 5 voltage over all test cases

TC#	Load Bus 5 Voltage Peak Deviation (PD) in kV = Max – Min			$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10$		
	A	B	C	A	B	C
TC1	0.213	0.215	0.211	1	1	1
TC2	0.411	0.41	0.411	1.929577	1.906977	1.947867
TC3	0.597	0.6	0.602	2.802817	2.790698	2.853081
TC4	0.756	0.78	0.77	3.549296	3.627907	3.649289
TC5	0.911	0.931	0.916	4.276995	4.330233	4.341232
TC6	1.043	1.046	1.044	4.896714	4.865116	4.947867
TC7	1.12	1.131	1.128	5.258216	5.260465	5.345972
TC8	1.197	1.23	1.209	5.619718	5.72093	5.729858
TC9	1.314	1.336	1.31	6.169014	6.213953	6.208531
TC10	1.4	1.408	1.393	6.57277	6.548837	6.601896

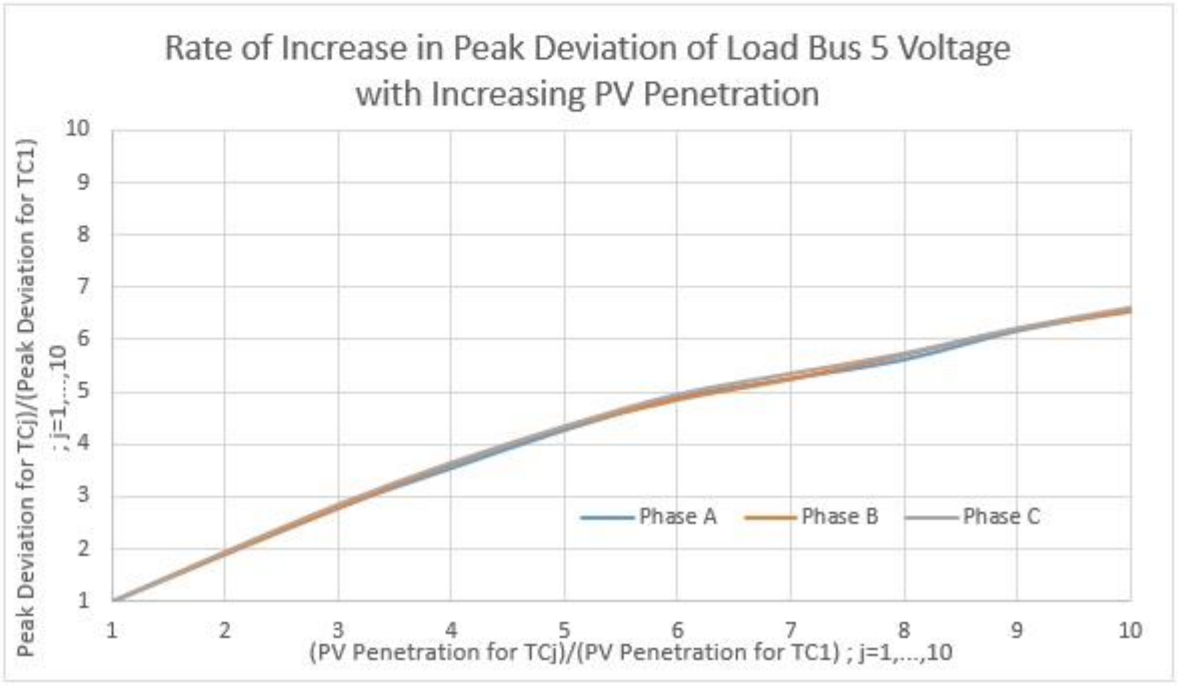


Figure 4.17: Comparing increase in PDR for Load Bus 5 voltage to the increase in PVR

4.3.2.3 Load Bus 6 Voltage

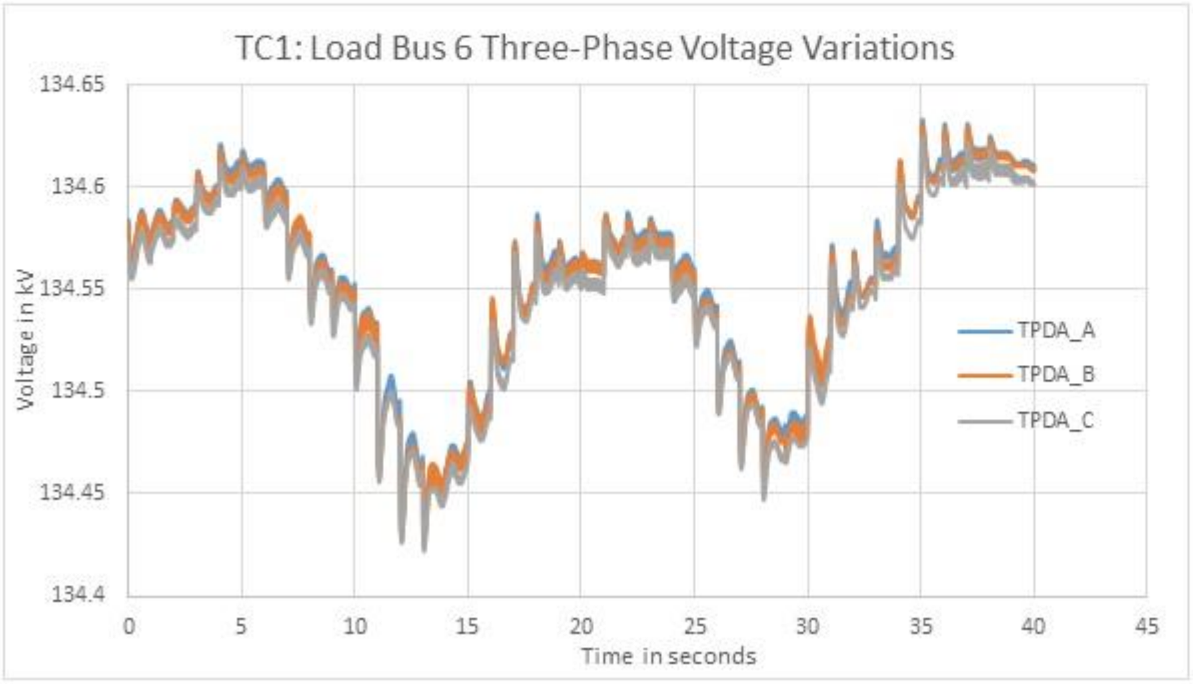


Figure 4.18: Load Bus 6 Three-Phase Voltage for TC1

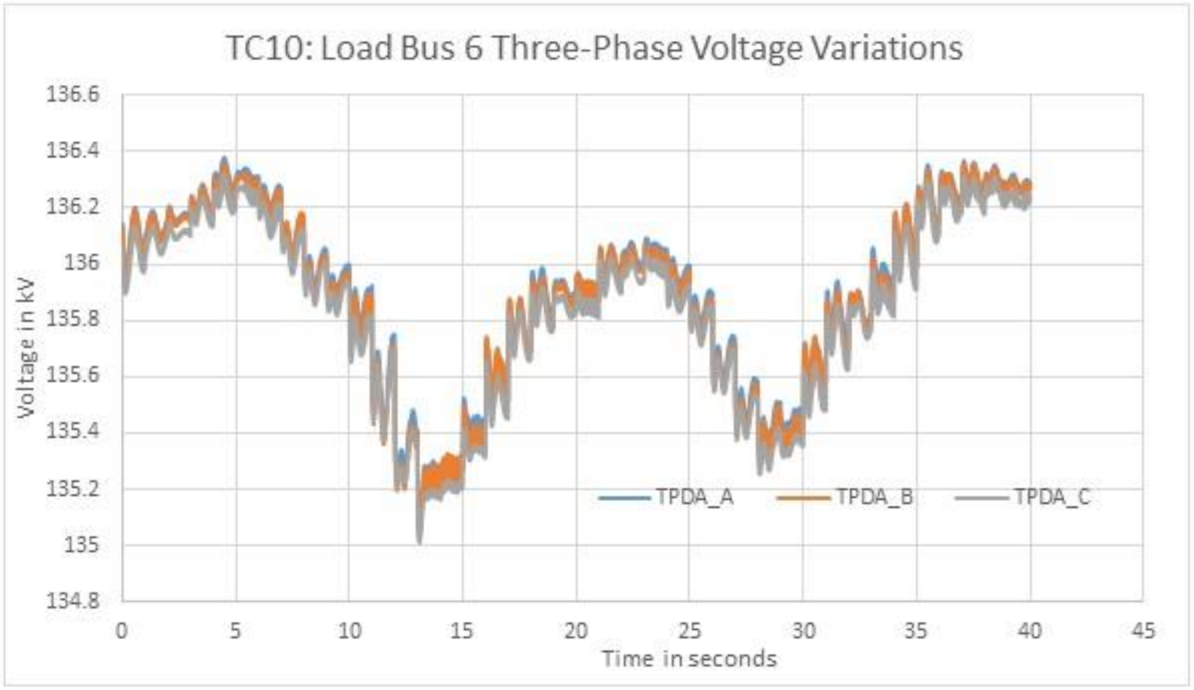


Figure 4.19: Load Bus 6 Three-Phase Voltage for TC10

Table 4.12: Maximum & minimum load bus 6 voltage over all test cases

TC#	Max kV at Load bus 6			Min kV at Load bus 6		
	A	B	C	A	B	C
TC1	134.633	134.631	134.625	134.429	134.425	134.422
TC2	134.876	134.872	134.862	134.484	134.479	134.469
TC3	135.116	135.109	135.094	134.548	134.539	134.522
TC4	135.323	135.322	135.299	134.609	134.585	134.573
TC5	135.525	135.525	135.497	134.665	134.646	134.634
TC6	135.728	135.729	135.693	134.75	134.745	134.713
TC7	135.892	135.881	135.845	134.841	134.815	134.783
TC8	136.055	136.044	136.003	134.928	134.886	134.862
TC9	136.215	136.204	136.16	134.983	134.95	134.928
TC10	136.379	136.359	136.32	135.06	135.04	135.011

Table 4.13: PD and PDR for load bus 6 voltage over all test cases

TC#	Load Bus 6 Voltage Peak Deviation (PD) in kV = Max – Min			$PDR = \frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1,..,10$		
	A	B	C	A	B	C
TC1	0.204	0.206	0.203	1	1	1
TC2	0.392	0.393	0.393	1.921569	1.907767	1.935961
TC3	0.568	0.57	0.572	2.784314	2.76699	2.817734
TC4	0.714	0.737	0.726	3.5	3.57767	3.576355
TC5	0.86	0.879	0.863	4.215686	4.26699	4.251232
TC6	0.978	0.984	0.98	4.794118	4.776699	4.827586
TC7	1.051	1.066	1.062	5.151961	5.174757	5.231527
TC8	1.127	1.158	1.141	5.52451	5.621359	5.62069
TC9	1.232	1.254	1.232	6.039216	6.087379	6.068966
TC10	1.319	1.319	1.309	6.465686	6.402913	6.448276

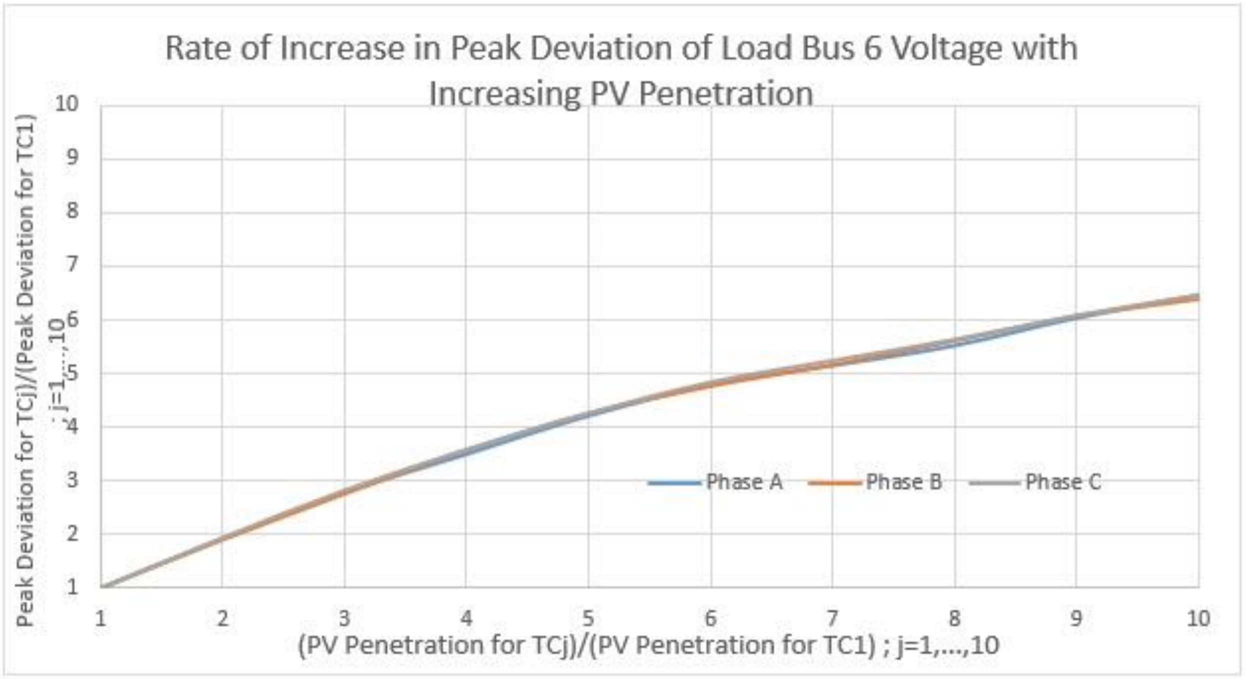


Figure 4.20: Comparing increase in PDR for load bus 6 voltage to the increase in PVR

4.4 Impact of PV Generation on Transmission Network Currents

A similar analysis is carried out for currents leaving the machine terminals in the electric network. PD, PDR and PVR, defined in the previous section, are used for analyzing results for currents in the network as well. Figures 4.21, 4.22, 4.24 and 4.25 show the plots for three-phase currents from generators 1 and 2. Tables 4.14 – 4.17 show that the peak deviations in currents are also seen to increase. This increase is almost linear for generator 1 from TC1 to TC7, after which, as the machine oscillates between motor and generator modes, the increase in PDR is seen to fall (as in the case of voltage). For generator 2, however, PDR is seen to increase proportionally with increase in PVR. Plots comparing the increase in PDR to the increase in PVR are shown in figures 4.23 and 4.26.

4.4.1 Generator 1 Current

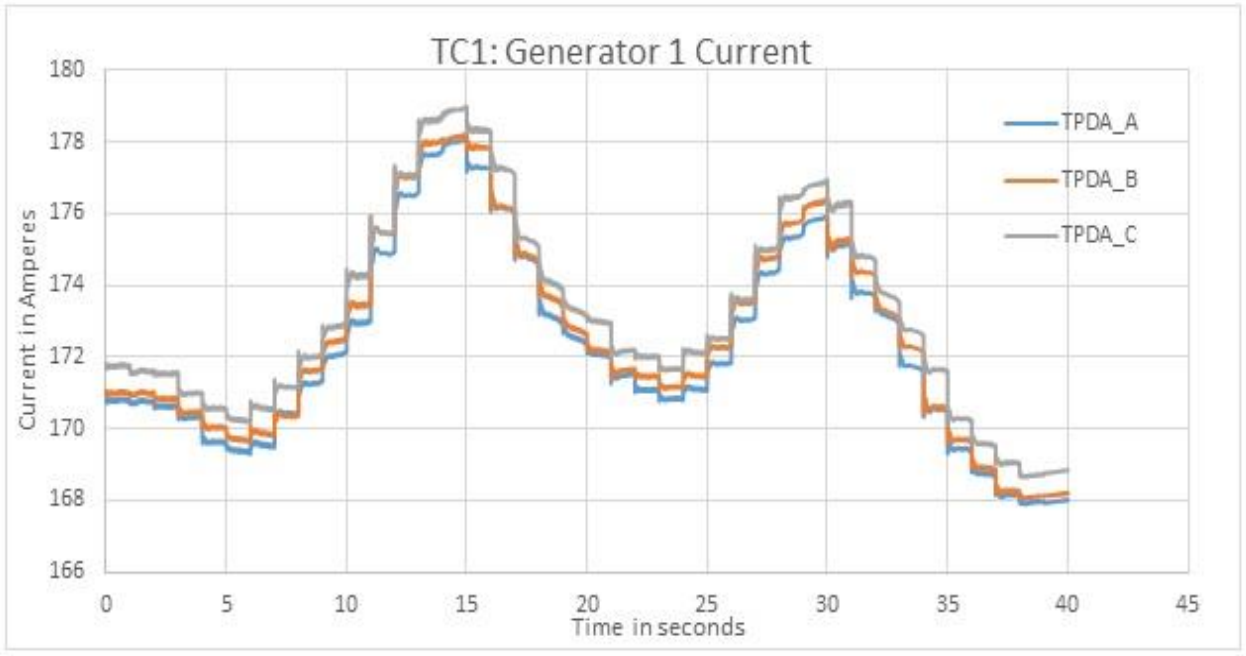


Figure 4.21: Generator 1 Current for TC1

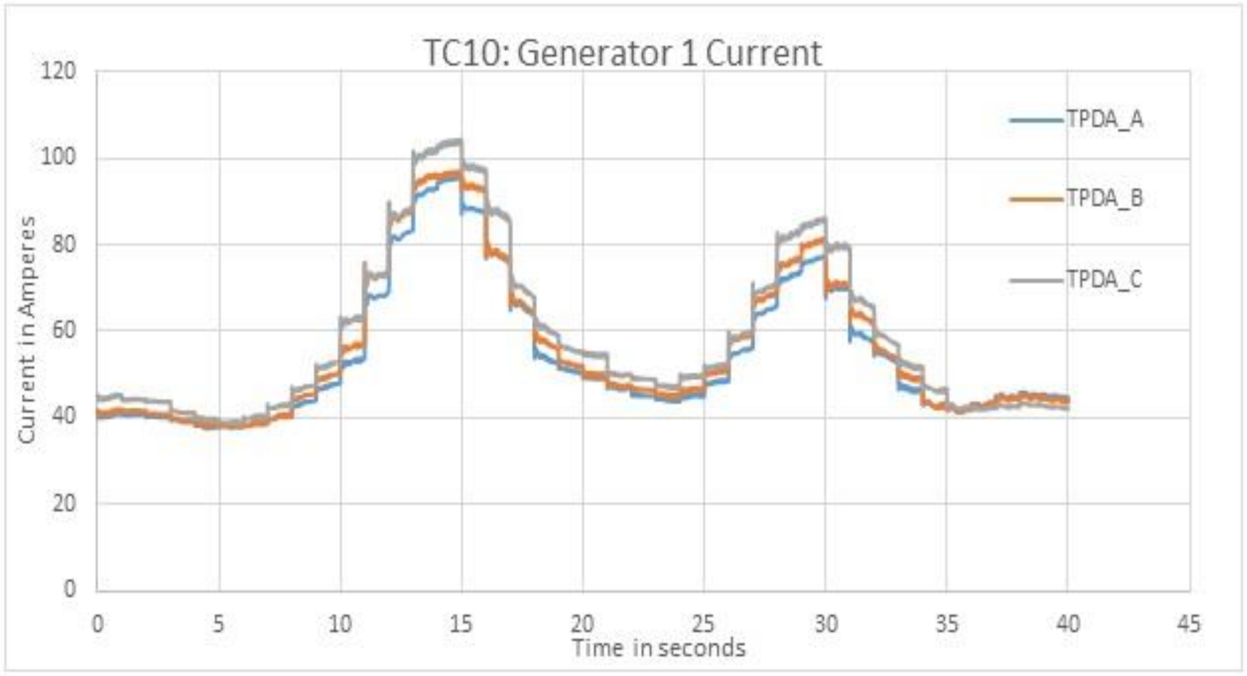


Figure 4.22: Generator 1 Current for TC10

Table 4.14: Maximum & minimum generator 1 current over all test cases

TC#	Max Amps from Gen 1			Min Amps from Gen 1		
	A	B	C	A	B	C
TC1	178.067	178.231	178.985	167.887	168.052	168.643
TC2	168.513	168.781	170.304	148.296	148.57	149.78
TC3	159.021	159.412	161.775	128.889	129.341	131.127
TC4	149.567	150.186	153.236	109.901	110.539	112.824
TC5	140.152	140.94	144.654	91.6619	92.4487	95.1295
TC6	130.993	131.795	136.411	74.3441	75.1732	78.3259
TC7	121.854	122.859	128.173	58.7901	59.6816	62.859
TC8	112.866	114.08	120.013	46.768	47.4949	50.2201
TC9	104.182	105.471	112.108	41.1895	41.3723	42.6127
TC10	95.7033	97.0553	104.388	37.5303	37.8506	38.6401

Table 4.15: PD and PDR for generator 1 current over all test cases

TC#	Gen 1 Current Peak Deviation in Amps = Max – Min			$\frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}$; $i = 1,..,10$		
	A	B	C	A	B	C
TC1	10.18	10.179	10.342	1	1	1
TC2	20.217	20.211	20.524	1.985953	1.985559	1.984529
TC3	30.132	30.071	30.648	2.959921	2.954219	2.96345
TC4	39.666	39.647	40.412	3.896464	3.89498	3.907561
TC5	48.4901	48.4913	49.5245	4.763271	4.763857	4.788677
TC6	56.6489	56.6218	58.0851	5.564725	5.562609	5.616428
TC7	63.0639	63.1774	65.314	6.194882	6.206641	6.315413
TC8	66.098	66.5851	69.7929	6.492927	6.541419	6.748492
TC9	62.9925	64.0987	69.4953	6.187868	6.297151	6.719716
TC10	58.173	59.2047	65.7479	5.71444	5.816357	6.357368

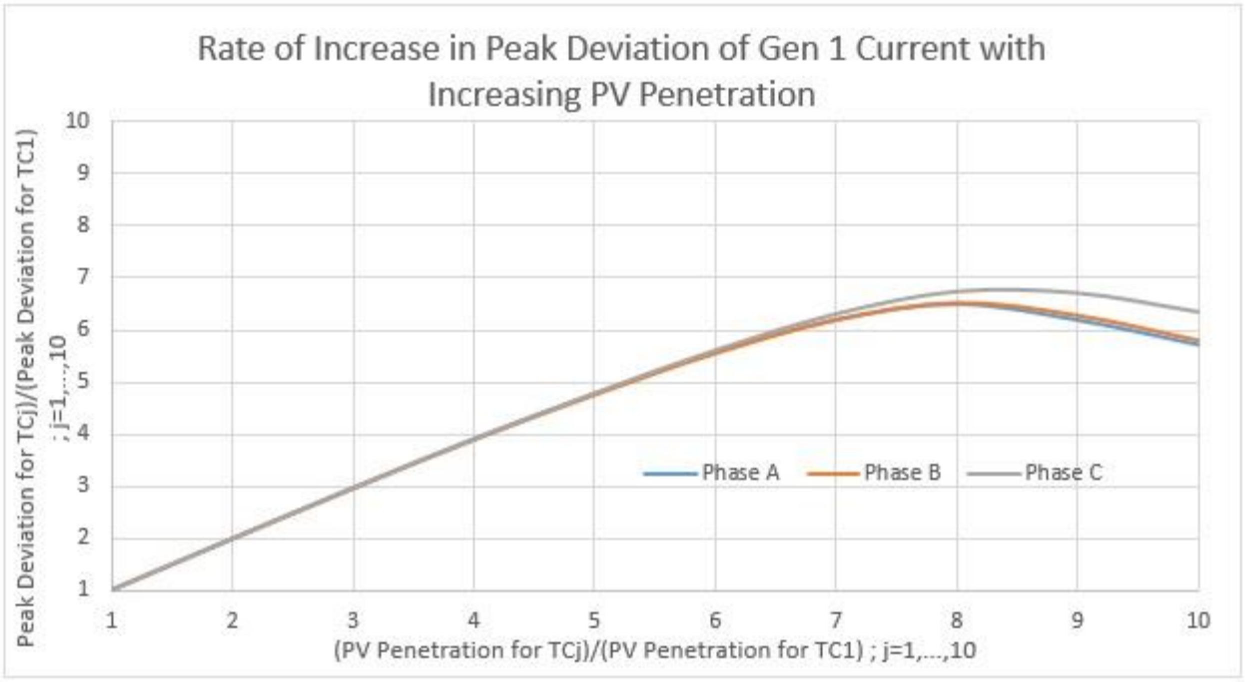


Figure 4.23: Comparing increase in Generator 1 Current to the increase in PVR

4.4.2 Generator 2 Current

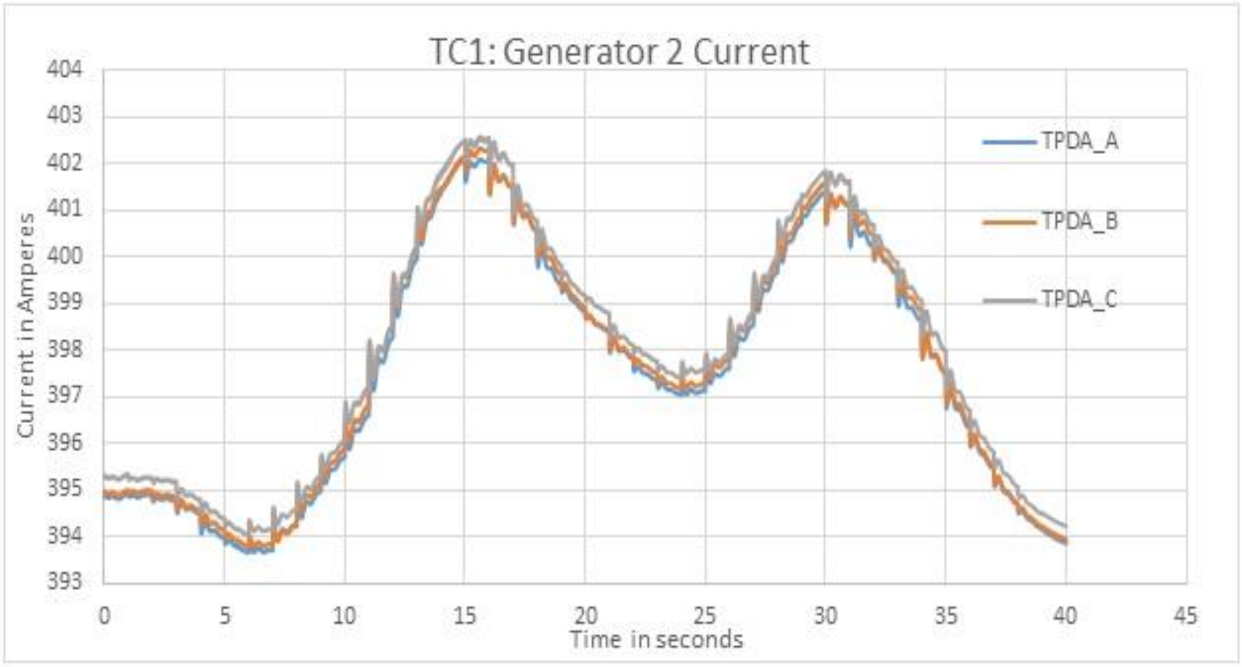


Figure 4.24: Generator 2 Current for TC1

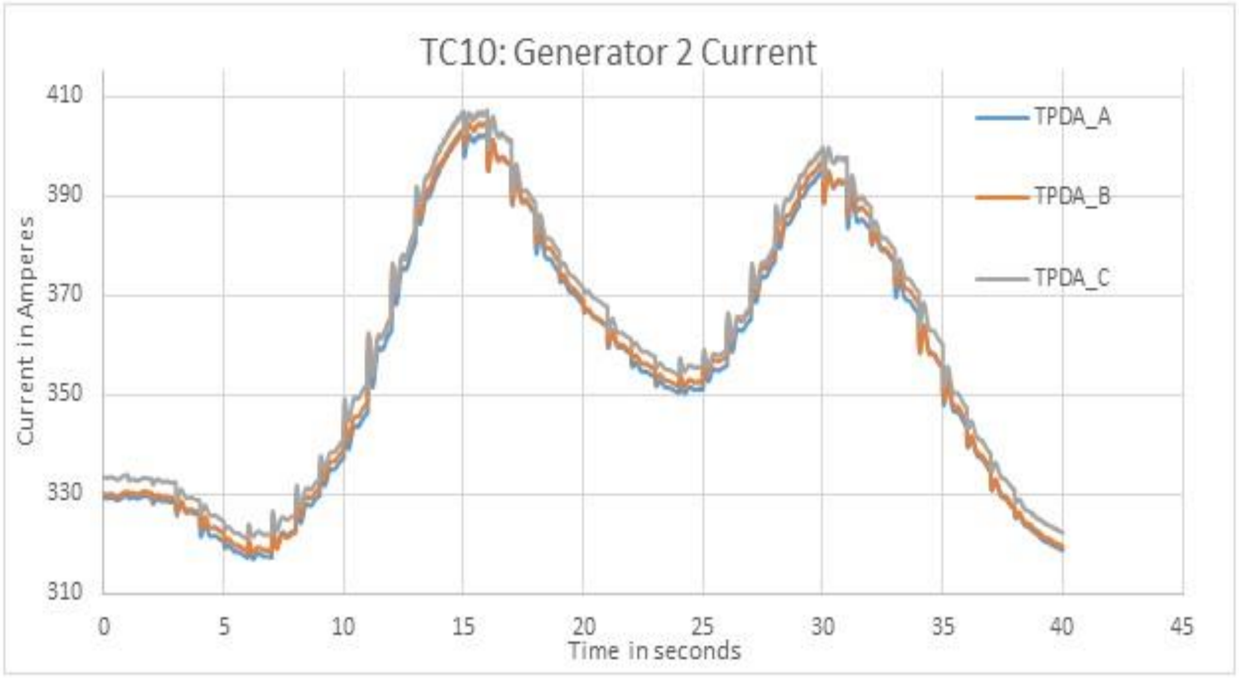


Figure 4.25: Generator 2 Current for TC10

Table 4.16: Maximum & minimum generator 2 current over all test cases

TC#	Max Amps from Gen 2			Min Amps from Gen 2		
	A	B	C	A	B	C
TC1	402.134	402.336	402.579	393.649	393.784	394.032
TC2	402.104	402.496	402.987	385.144	385.418	385.905
TC3	402.039	402.656	403.372	376.723	377.088	377.853
TC4	402.109	402.877	403.805	368.195	368.688	369.737
TC5	402.188	403.085	404.326	359.668	360.305	361.623
TC6	402.274	403.318	404.837	351.23	352.019	353.577
TC7	402.35	403.592	405.291	342.748	343.664	345.488
TC8	402.629	404.064	405.746	334.012	334.958	337.148
TC9	402.84	404.408	406.387	325.406	326.495	328.949
TC10	402.906	404.655	407.13	316.879	318.134	320.852

Table 4.17: PD and PDR for generator 2 current over all test cases

TC#	Gen 2 Current Peak Deviation in Amps = Max – Min			$\frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1};i = 1,..,10$		
	A	B	C	A	B	C
TC1	8.485	8.552	8.547	1	1	1
TC2	16.96	17.078	17.082	1.998821	1.99696	1.998596
TC3	25.316	25.568	25.519	2.983618	2.98971	2.985726
TC4	33.914	34.189	34.068	3.996936	3.997778	3.98596
TC5	42.52	42.78	42.703	5.011196	5.002339	4.996256
TC6	51.044	51.299	51.26	6.015793	5.99848	5.997426
TC7	59.602	59.928	59.803	7.024396	7.007484	6.996958
TC8	68.617	69.106	68.598	8.086859	8.080683	8.025974
TC9	77.434	77.913	77.438	9.125987	9.1105	9.060255
TC10	86.027	86.521	86.278	10.13872	10.11705	10.09454

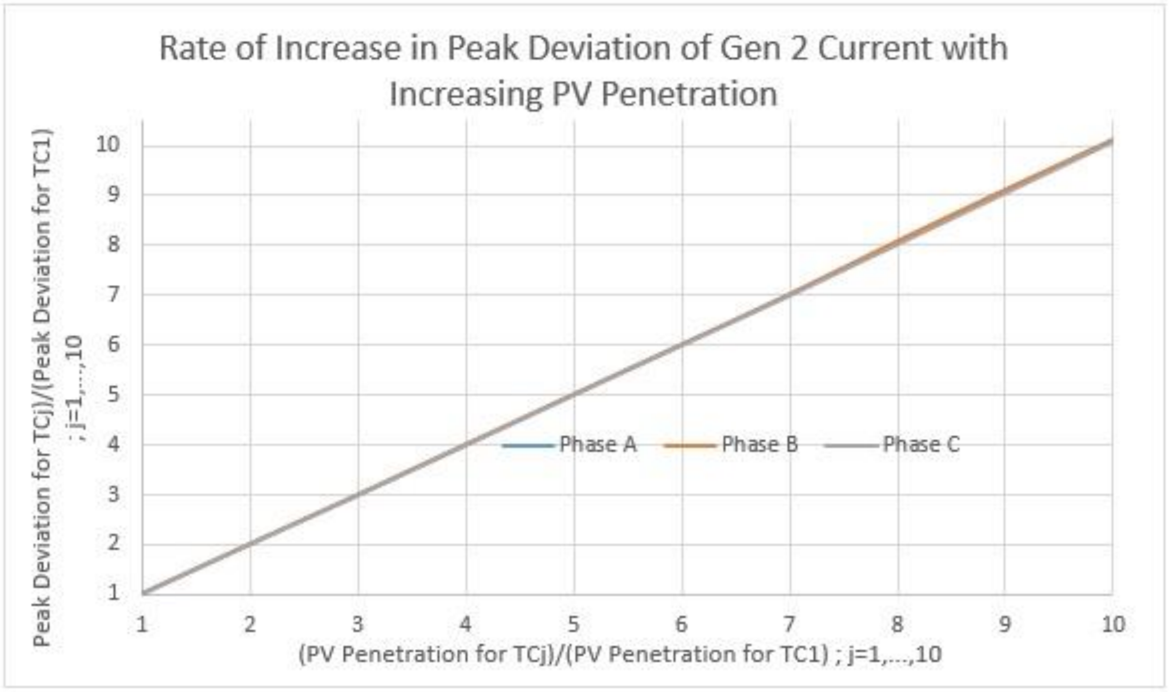


Figure 4.26: Comparing increase in PDR for Generator 2 Current to the increase in PVR

4.5 Impact of PV Generation on Rotor Speed

Figure 4.27 shows the rotor speed plots for machine 1 for all ten test cases: TC1 to TC10. Rotor speed results for all machines are found to be the same indicating that the system synchronism is not lost. Table 4.18 presents the analysis of these rotor speed results as it records the peak rotor speed deviation from its nominal value of 60 Hz. It must be noted that even under the base case solar PV penetration of TC1, the rotor speed is seen to undergo a maximum deviation of 0.13 Hz, which is generally much greater than industry acceptable levels of deviation.

Further, as solar output increases linearly from TC1 to TC10, the rotor speed deviations are also seen to multiply by the same factor. This is evident from figure 4.27, but the exact rates of increase in peak rotor speed deviation are noted in table 4.18. PDR (Peak Deviation Ratio) and PVR (PV penetration Ratio) are again defined as before. Figure 4.28 shows a proportional increase in PDR as PVR increases from 1 in TC1 to 10 in TC10. Thus, as PV penetration increases, rotor speed/system frequency deviations from their nominal value of 60 Hz are seen to increase at the same rate.

Finally, even though the solar measurements used for this study are three-phase solar PV measurements, there exists a small imbalance (in the order of a few KWs) in the three-phase solar outputs. This imbalance causes 120 Hz rotor oscillations, which also keeps increasing with PV penetration and can be easily captured for TC10 as shown in figure 4.29. This sheds light on the possibility of highly significant 120 Hz rotor oscillations damaging the machines in case of a larger imbalance due to reasons like unbalanced PV generation coupled with cloud cover/other factors that may result in unbalanced PV generation.

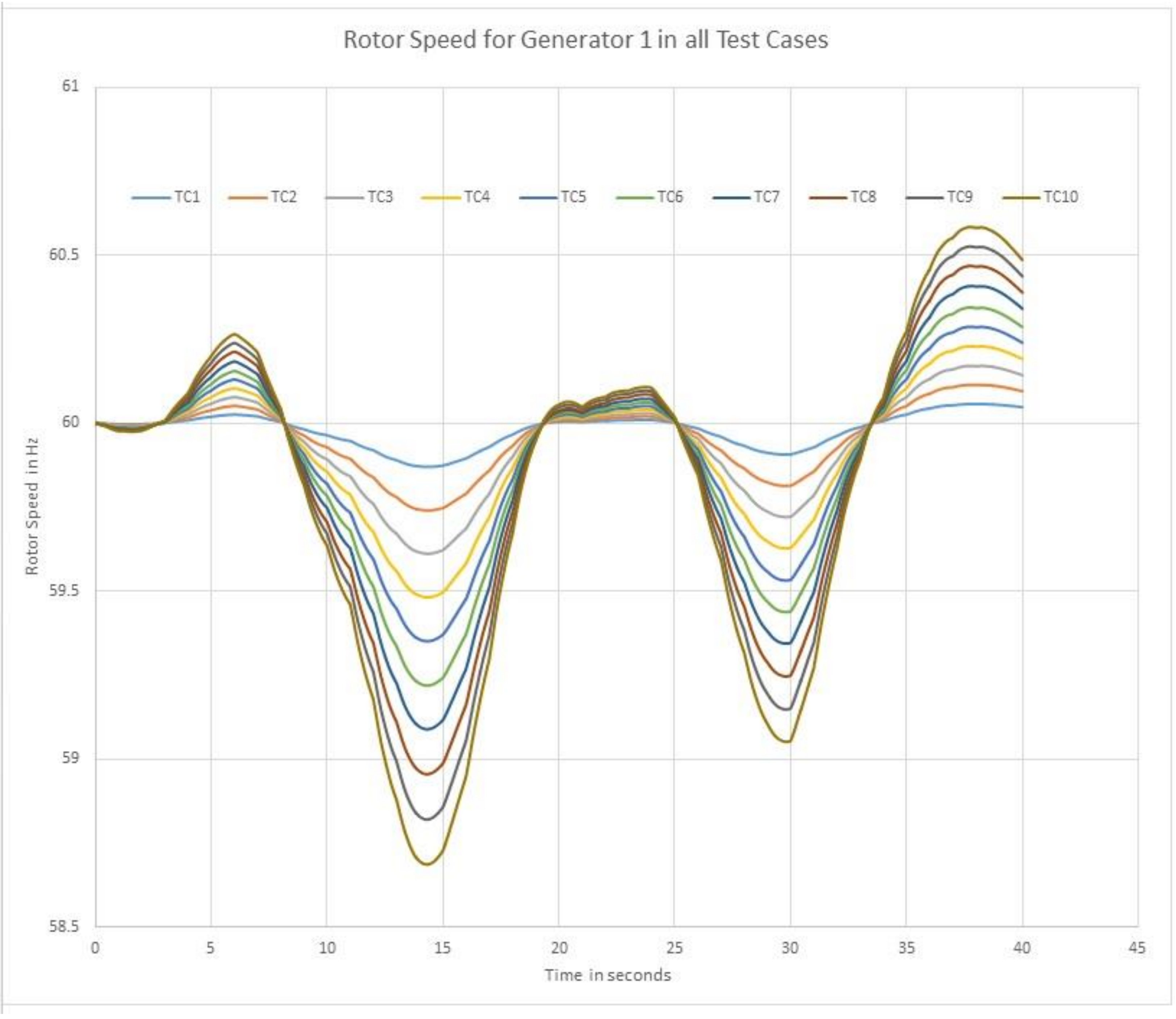


Figure 4.27: Rotor speed comparison for generator 1 over all ten test cases

Table 4.18: PD and PDR in rotor speeds over all test cases

TC#	Peak Rotor Speed Deviation from 60 Hz	$\frac{Peak\ Deviation\ for\ TCi}{Peak\ Deviation\ for\ TC1}\ ; i = 1, \dots ,10$
TC1	0.129953	1
TC2	0.259935	2.000222
TC3	0.388739	2.991382
TC4	0.518689	3.991359
TC5	0.649339	4.996725
TC6	0.781565	6.014216
TC7	0.911483	7.013949
TC8	1.045205	8.042952
TC9	1.180248	9.08212
TC10	1.314098	10.1121

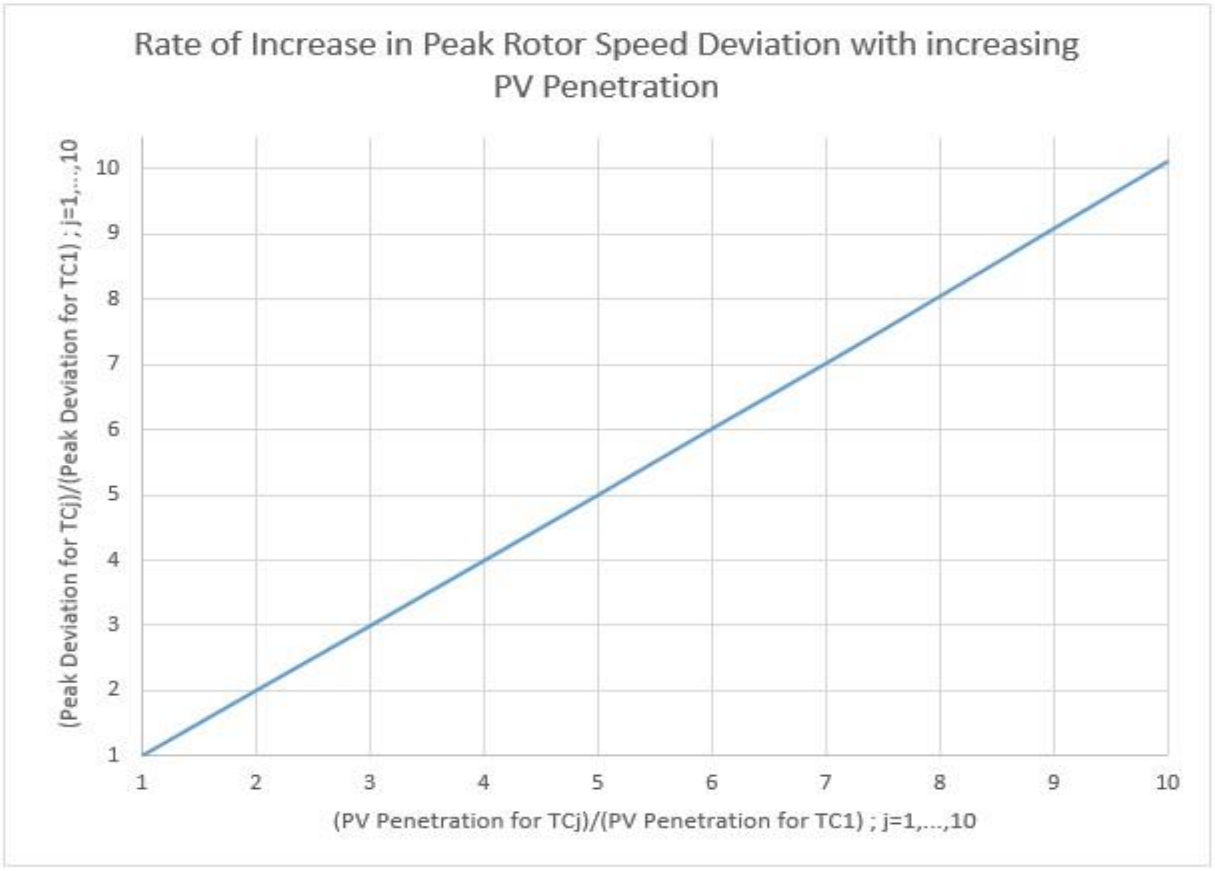


Figure 4.28: Comparing increase in rotor speed deviations to the increase in PVR

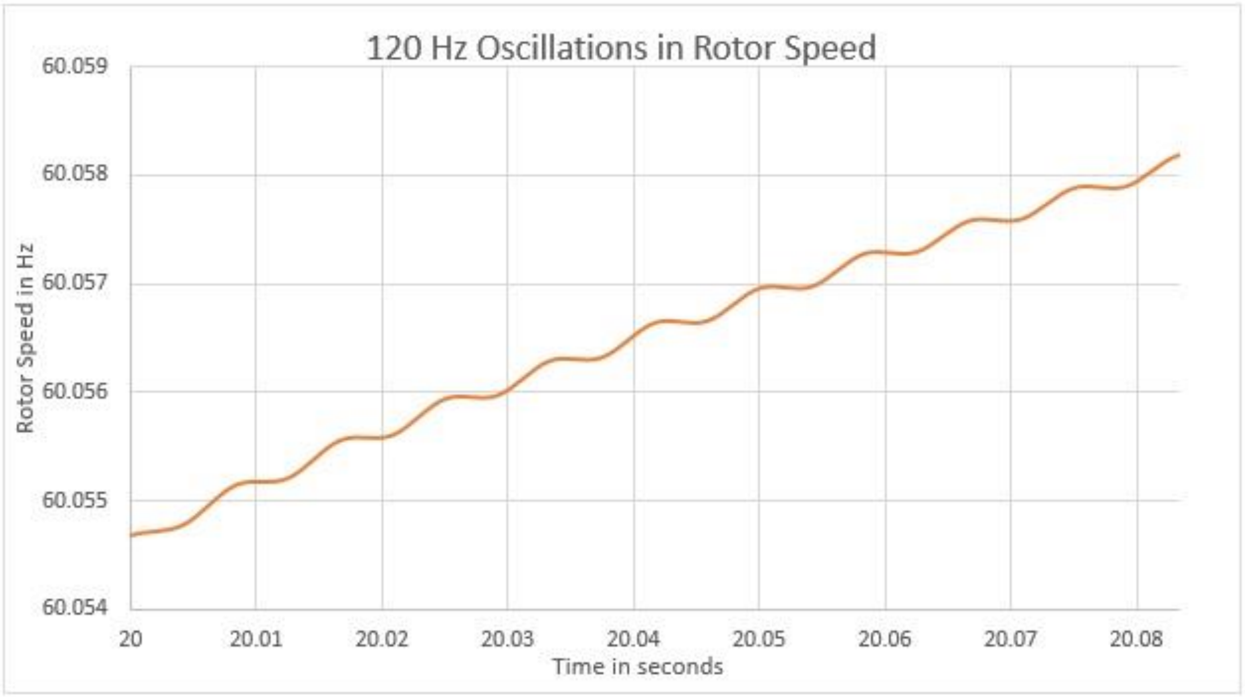


Figure 4.29: 120 Hz rotor oscillations for TC10

To review the results presented above, it is seen that with solar PV installations on load busses 5 and 6, voltage profiles throughout the electric network are seen to undergo fast oscillations (far from the desired flat voltage profiles). Further, as the PV output increases, these oscillations and deviations from steady state values increase. The largest deviations in voltages are seen at busses that are most tightly coupled to the load busses 5 and 6. Understandably, currents leaving generator terminals are also seen to vary in time as the voltages in the electric network keep changing. Rotor speed deviations are seen to increase proportionally with the PV penetration and for the test system under consideration, an unacceptable magnitude of deviation is reported even for the base case. Imbalance in solar output is seen to cause 120 Hz oscillations that magnify as the imbalance increases. These results show that DG generation influences the entire electric grid and that its effects are no longer limited to the distribution system. These effects can be analyzed by modeling the combined T&D system and associated dynamics using TPDA. This work has shown that integration of DG into the network should take into account the impact of rapidly varying DG generation on the transmission system and its conventional generation sources.

Chapter 5: Conclusion

5.1 Summary

This thesis considers a new methodology for three-phase dynamic analysis (TPDA) of large systems. TPDA adopts a mixed time and phasor domain approach. The time domain analysis of electric machines helps capture machine dynamics, while a phasor domain analysis is used to solve the power flow. TPDA is different from EMTP software in that it uses a graph-trace analysis based multi-phase power flow instead of integrating time domain differential equations representing the three-phase electric network. With graph trace analysis distributed computations, a big advantage in terms of speed of solution could result.

Further, the work here also validates the performance of TPDA by comparing results obtained from TPDA to those obtained using GE-PSLF®. Thereafter, results based on a comparative study of TPDA and GE-PSLF® are presented that show the need for three-phase dynamic analysis of large systems. It is shown that as the size of the unbalanced disturbance in the WECC 9-bus system increases, the imbalance and peak deviation in voltages and currents throughout the electric network also increase almost proportionally. The maximum swing in voltages and currents over subsequent iterations of TPDA (excluding fault transients), and the peak rotor speed deviations are shown to increase approximately proportionally to the increase in the size of the unbalanced disturbance. Moreover, TPDA results are shown to capture 120 Hz rotor oscillations due to network imbalance.

Finally, an application of TPDA is presented wherein the effects of increasing distribution level PV penetration on the voltages and currents in the transmission network are analyzed. Rotor speed deviations for generators connected to the transmission system as a result of increasing PV penetration on the distribution circuits are also evaluated. It is shown that as PV generation increases, peak deviations in transmission level voltages and currents (no longer steady) increase as well. The rate of this increase in peak deviations with PV penetration is also analyzed. Rotor speed deviations are once again shown to increase proportionally with increasing PV penetration. Finally, TPDA is shown to be a useful tool for evaluating reverse power flows in the electric network due to DG generation and for testing the dynamic performance of control schemes (that may be developed to allow higher DG penetration) with respect to the health of the combined T&D system.

5.2 Future Work

The algorithm that has been considered in the course of this thesis is a novel approach and provides significant scope for potential enhancements. Currently, TPDA has been developed such that certain specific machine models (specified in chapter 2) can be used in the electric network. The implementation of TPDA is done in such a manner that any number and type of machine models could be added to the network model, without affecting most of the code. So, taking advantage of this generalized implementation, future work entails adding additional models for generators and their controllers; DG inverters; and induction machines and their controllers.

As discussed earlier, TPDA solves machine dynamics corresponding to every machine separately. This allows distributing the numerical integration of differential equations representing these machine dynamics onto several processors. Further, TPDA adopts a multi-phase power flow analysis technique based on graph trace analysis that can also be distributed to attain a much greater speed of solution. So one of the tasks to be accomplished as a follow up to this thesis is to implement TPDA with a multi-processor programming approach and to report the efficiency of such an approach. It is with this vision of possibly using parallel or distributed computation programming to achieve high efficiency, that an object oriented implementation of TPDA in C++/C# was developed in this thesis.

Another area of work is the need for more robust numerical integration methods in the object oriented implementation of TPDA. At present, only Euler based numerical integration is available in the object oriented implementation and MATLAB ODE functions have to be used for solving differential equations if numerical instability is observed with using the Euler method. Thus, future work should include implementation of more robust integration methods in the object oriented implementation as well. This will eliminate the dependency on MATLAB ODE functions.

Another interesting observation made during this thesis paves the way for a possible study in the future. It was found that the voltage at generator terminals was less unbalanced compared to other network busses. It might be interesting to report whether generator’s internal voltage always tries to maintain itself balanced.

Moreover, as an immediate next step, TPDA should be applied to studying electromechanical transients in large T&D models with perhaps 100,000 components or more. Work is in progress in Dr. Broadwater’s research group and a paper will be published soon to discuss the application of TPDA for studying a large real world combined T&D model. Finally, TPDA should also be used for testing existing control strategies that allow high DG penetration or smart grid technologies like Conservation Voltage Reduction (CVR) and Volt-Var Control/Optimization (VVC/VVO). Such studies are key in realizing the full potential of TPDA.

REFERENCES

[1] R. G. Harley, E. B. Makram, E. G. Duran, “The effects of Unbalanced Networks on Synchronous and Asynchronous Machine Transient Stability”, *Electric Power Systems Research,* pp. 119-127, 1987.

[2] A. R. Blaylock, R. Hindmarsh and K.A. Foster, “Some criticial aspects of generator capability under unbalanced operating conditions”, IEEE Trans., PAS-96 (1977) 1470 – 1478.

[3] X. –P. Zhang, P. Ju, E. Handschin, “Continuation Three-Phase Power Flow: A Tool for Voltage Stability Analysis of Unbalanced Three-Phase Power System”, IEEE Trans. on Power Systems, vol. 20, no. 3, August 2005.

[4] X. Bai, T. Jiang, Z. Guo, Z. Yan, and Y. Ni, “A Unified Approach for Processing Unbalanced Conditions in Transient Stability Calculations”, IEEE Trans. on Power Systems, vol. 21, no. 1, February 2006

[5] R. Lasseter, “Dynamic Models of micro-turbines and fuel cells”, in *Proc. 2001 IEEE Power Engineering Society Summer Meeting,* vol. 2, pp. 761-766.

[6] S. A. Papathanassiou, N. D. Hatziargyriou, “Technical Requirements for the connected of dispersed generation to the grid”, in *Proc. 2001 IEEE Power Engineering Society Summer Meeting,* vol. 2, pp. 749-754.

[7] J. A. P. Lopes, “Integration of dispersed generation on distribution networks-impact studies”, in *Proc. 2002 Power Engineering Society Winter Meeting,* vol. 1, pp. 323-328.

[8] N. Jenkins, G. Strbac, “Impact of embedded generation on distribution system voltage stability”, *IEE Colloquium on Voltage Collapse (Digest No: 1997/101),* pp. 9/1-9/4, 1997.

[9] J. G. Slootweg, W. L. Kling, “Impacts of Distributed Generation on Power System Transient Stability”, in *Proc 2002 IEEE Power Engineering Society Summer Meeting.*

[10] N. Jenkins, “Impact of dispersed generation on power systems”, *Invited paper, ELECTRA,* no. 199, Dec. 2001.

[11] J F. de Leon, B. –T. Ooi, “Damping power system oscillations by unidirectional control of alternative power generation plants”, in *Proc. 2001 IEEE Power Engineering Society Winter Meeting.*

[12] M. K. Donnelli, J. E. Dagle, D.J Trudnowski, G. J. Rogers, “ Impacts of the distributed utility on transmission system stability”, IEEE Trans. on Power Systems, vol. 11, pp. 741-746, May 1996.

[13] J. G. Slootweg, S. W. H. de Haan, H. Polinder, W. L. Kling, “Modelling wing turbines in power system dynamics simulations”, in *Proc. 2001 IEEE Power Engineering Society Summer Meeting.*

[14] J. G. Slootweg, W. L. Kling, “Modelling and Analysing Impacts of Wing Power on Transient Stability of Power Systems”, Wing Engineering, v.26, n.1, 2002, pp. 3-20.

[15] M. Reza, J. G. Slootweg, P. H. Schavemaker, W. L. Kling, L. van der Sluis, “Investigating impacts of Distributed Generation on Transmission System Stability”, in *Proc. 2003 IEEE Bologna PowerTech Conference.*

[16] H. Jain, A. Parchure, R. P. Broadwater, M. Dilek, J. Woyak, “Three-Phase Dynamics Simulation of Power Systems Using Combined Transmission and Distribution System Models”, submitted to the IEEE Trans. on Power Systems, **arXiv:1505.06225 [cs.SY]**

[17] M. Dilek, F. de Leon, R. P. Broadwater, S. Lee, "A Robust Multiphase Power Flow for General Distribution Networks", IEEE Trans. on Power Systems, vol. 25, no. 2, pp. 760-768, May 2010.

[18] P. Kundur, *Power System Stability and Control,* New York: McGraw-Hill, 1994, pp. 174-179.

[19] P. W. Sauer, M. A. Pai, *Power System Dynamics and Stability,* New Jersey: Prentice-Hall, 1998.

[20] R. H. Park, "Two-reaction theory of synchronous machines generalized method of analysis-part I", IEEE Trans. of the AIEE, vol. 48, no. 3, pp. 716-727, July 1929.

[21] Components, Circuits, Devices & Systems | Power, Energy, & Industry Applications, "IEEE Standard Definitions for Excitation Systems for Synchronous Machines", IEEE Std. 421.1.2007, July 2007. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=5981340

[22] GE Energy's Positive Sequence Load Flow (PSLF) Software and User Manual, http://site.ge-energy.com/prod_serv/products/utility_software/en/ge_pslf/index.htm

[23] S.A.A. Omran, "Control Applications and Economic Evaluations of Distributed Series Reactors in Unbalanced Electrical Transmission Systems", Ph.D. dissertation, Dept. Elec. & Comp. Eng., Virginia Tech, 2015.

[24] E. N. Azadani, C. A. Canizares, D. E. Olivares, K. Bhattacharya, "Stability Analysis of Unbalanced Distribution Systems With Syncrhonous Machine and DFIG Based Distributed Generators", IEEE Trans. on Smart Grid, vol. 5, no. 5, pp. 2326-2338, September 2014

[25] I. Xyngi, A. Ishchenko, M. Popov, L. Sluis, "Transient Stability Analysis of a Distribution Network with Distributed Generators", IEEE Trans. on Power Systems, vol. 24, no, 29, pp. 1102-1104, May 2009

[26] R. H. Salim, R. A. Ramos, "A Model-Based Approach for Small Signal Stability Assessment of Unbalanced Power Systems", IEEE Trans. on Power Systems, vol. 27, no. 4, pp. 2006-2014, November 2012

[27] G. C. paap, "Symmetrical Components in the Time Domain and Their Applications to Power Network Calculations", IEEE Trans. on Power Systems, vol. 15, no. 2, pp. 522-528, May 2000

[28] E. B. Makram, V. O. Zamrano, R. G. Harley, J. C. Balda, "Three-Phase Modeling for Transient Stability of Large Scale Unbalanced Distribution Systems", IEEE Trans. on Power Systems, vol. 4, no. 2, pp. 487-493, May 1989

[29] N. Hadjsaid, J. –F, Canard, F. Dumas, "Dispersed generation impact on distribution networks", *IEEE Computer Applications in Power,* vol. 12, pp. 22-28, April 1999.

[30] X. Bai, T. Jiang, Z. Guo, Z. Yan, and Y. Ni, "A Unified Approach for Processing Unbalanced Conditions in Transient Stability Calculations", IEEE Trans. on Power Systems, vol. 21, no. 1, February 2006.

[31] R. H. Salim, R. A. Ramos, N. G. Bretas, "Analysis of the Small Signal Dynamic Performance of Synchronous Generators under Unbalanced Operating Conditions", *Proc. IEEE PES General Meeting,* pp. 1-6, 2010.

[32] E. N. Azadani, C. Canizares, K. Bhattacharya, "Modeling and Stability Analysis of Distributed Generation", *IEEE PES General Meeting,* vol. 1, no. 8, pp. 22-26, July 2010.

Appendix A: Definition of Symbols Used in Chapter 2

Symbol	Definition
ψ_d, ψ_q, ψ_0	$d, q,$ 0 axis fluxes
v_d, v_q, v_0	$d, q,$ 0 axis voltages
I_d, I_q, I_0	$d, q,$ 0 axis currents
E'_q, ψ_{1d}	State variables associated with d-axis field winding and damper winding, respectively
E'_d, ψ_{2q}	State variables associated with q-axis slow and fast time constant damper windings, respectively
R_s	Stator resistance
X_d, X'_d, X''_d	d axis self, transient and sub-transient reactance, respectively
X_q, X'_q, X''_q	q axis self, transient and sub-transient reactance, respectively
T'_{d0}, T''_{d0}	Time constant (seconds) associated with d-axis field winding and damper winding, respectively
T'_{q0}, T''_{q0}	Time constant (seconds) associated with q-axis slow and fast time constant damper windings, respectively
H	Inertia Constant (seconds)
$\delta; \delta_0$	Rotor angle w.r.t synchronously rotating reference frame; initial rotor angle
T_M	Mechanical Torque
E_{fd}	Field Voltage
ω	Rotor angular speed (radians/second)
ω_s	Synchronous rotor speed (radians/second)
T_{FW}	Friction Windage Torque
X_{ls}	Leakage Reactance

Appendix B: Exciter & Governor Dynamic Model Input Parameters

IEEE (2005) type AC7B Excitation System Input Parameters [22]:

Tr	0.0	Filter time constant, sec.
Kpr	3.89	Regulator proportional gain, p.u. (> 0. if Kir = 0.)
Kir	3.89	Regulator integral gain, p.u.
Kdr	0.0	Regulator derivative gain, p.u.
Tdr	0.0	Derivative gain washout time constant, sec.
Vrmax	6.74	Maximum regulator output, p.u.
Vrmin	-6.74	Minimum regulator output, p.u.
Kpa	117.7	Amplifier proportional gain. (> 0. if Kia = 0.)
Kia	26.8	Amplifier integral gain, p.u.
Vamax	1.0	Maximum amplifier output, p.u.
Vamin	-0.95	Minimum amplifier output, p.u.
Kp	12.08	Exciter field voltage source gain, p.u.
Kl	10.0	Exciter field voltage lower limit parameter, p.u.
Te	3.0	Exciter time constant, sec. (> 0.)
Vfemax	15.2	Exciter field current limit parameter, p.u. Efd (note e)
Vemin	0.0	Minimum exciter ouput voltage, p.u. Efd
Ke	1.0	Exciter field resistance constant, p.u.
Kc	0.13	Rectifier regulation factor, p.u.
Kd	1.14	Exciter internal reactance, p.u.
Kf1	0.194	Field voltage feedback gain, p.u.
Kf2	0.0	Exciter field current feedback gain, p.u.
Kf3	0.0	Rate feedback gain, p.u.
Tf	1.0	Rate feedback time constant, sec. (> 0.)
E1	6.67	Field voltage value 1, p.u. (note d)
S(E1)	1.951	Saturation factor at E1 (note d)
E2	5.0	Field voltage value 2, p.u. (note d)
S(E2)	0.156	Saturation factor at E2 (note d)
spdmlt	0	If = 1, multiply output (Efd) by generator speed (note f)

General Governor Model (GGOV1) Input Parameters [22]:

r	0.04	Permanent droop, p.u.
rselect	1.0	Feedback signal for droop
		= 1 selected electrical power
		= 0 none (isochronous governor)
		= -1 fuel valve stroke (true stroke)
		= -2 governor output (requested stroke)
Tpelec	1.0	Electrical power transducer time constant, sec. (>0.)
maxerr	0.05	Maximum value for speed error signal

minerr	-0.05	Minimum value for speed error signal
Kpgov	10.0	Governor proportional gain
Kigov	2.0	Governor integral gain
Kdgov	0.0	Governor derivative gain
Tdgov	1.0	Governor derivative controller time constant, sec.
vmax	1.0	Maximum valve position limit
vmin	0.15	Minimum valve position limit
Tact	0.5	Actuator time constant
Kturb	1.5	Turbine gain (>0.)
wfnl	0.2	No load fuel flow, p.u
Tb	0.5	Turbine lag time constant, sec. (>0.)
Tc	0.0	Turbine lead time constant, sec.
Flag	1.0	Switch for fuel source characteristic
		= 0 for fuel flow independent of speed
		= 1 fuel flow proportional to speed
Teng	0.0	Transport lag time constant for diesel engine
Tfload	3.0	Load Limiter time constant, sec. (>0.)
Kpload	2.0	Load limiter proportional gain for PI controller
Kiload	0.67	Load limiter integral gain for PI controller
Ldref	1.0	Load limiter reference value p.u.
Dm	0.0	Speed sensitivity coefficient, p.u.
ropen	.10	Maximum valve opening rate, p.u./sec.
rclose	-0.1	Minimum valve closing rate, p.u./sec.
Kimw	0.002	Power controller (reset) gain
Pmwset	80.0	Power controller setpoint, MW
aset	0.01	Acceleration limiter setpoint, p.u./sec.
Ka	10.0	Acceleration limiter Gain
Ta	0.1	Acceleration limiter time constant, sec. (>0.)
db	0.0	Speed governor dead band
Tsa	4.0	Temperature detection lead time constant, sec.
Tsb	5.0	Temperature detection lag time constant, sec.
rup	99.0	Maximum rate of load limit increase
rdown	-99.0	Maximum rate of load limit decrease

Appendix C: Code

```
#define _USE_MATH_DEFINES
#define _CRTDBG_MAP_ALLOC

#include <sstream>
#include "Eigen/Dense"
#include "passValues.h"
#include <iomanip>

#include "DynamicSimulationApp.h"
#include "PartClasses.h"
#include "LoadData.h"
#include "CComplex.h"
#include "time.h"
#include "CDouble3Ph.h"

#include "iostream"
#include "string.h"
#include "engine.h"
#include <cmath>
#include <conio.h>

#include "crtdbg.h"
#include <vector>
#include "extInit.h"
#include "genInit.h"
#include "govInit.h"
#include "dqParams.h"
#include "dewParams.h"
#include <tuple>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

using namespace Eigen;
using namespace std;
using std::endl;

#pragma comment (lib, "libmat.lib")
#pragma comment (lib, "libeng.lib")
#pragma comment (lib, "libmx.lib")
#pragma comment (lib, "libmex.lib")




void mainfunc(passValues oldValues, vector<genInit>& initialized, vector<extInit>&
extInitialized, vector<govInit>& govInitialized, dewParams pass, int samples,
vector<dqParams>& setParams,vector<vector<double>>& vMagNewLL, vector<vector<double>>&
vAngNewLL){

	int totalGen = 3;

	int count = 0;
	while (count < totalGen){

		//---------------------------------------------------------------
---------------------------------------------------------------------
----------------------------------------------------------------
		//Section 1: Setting up variables to be used within one iteration. Here, we
also extract all the previously calculated variables
		//---------------------------------------------------------------
---------------------------------------------------------------------
----------------------------------------------------------------
		vector<vector<double>>* opt = pass.getOpt();

		double genNum = (*opt)[0][count];//machine.opt[0];
		double extNum = (*opt)[1][count];//machine.opt[1];
		double pssNum = (*opt)[2][count];//machine.opt[2];
		double govNum = (*opt)[3][count];//machine.opt[3];

		//setting vt
		vector<vector<double>>* vMagOld  = pass.getVMagOld();
		vector<double> vt(totalGen);
		for(int i = 0; i<totalGen; i++){
			vt[i] = ((*vMagOld)[0][i] + (*vMagOld)[1][i] +
(*vMagOld)[2][i])/(3*pass.getVLNBase());
			//vt[i] = 138.1022;
		}

		pass.setVt(&vt);

		int cycle = pass.getCycle(); //this is the iteration number and not the
cycle
		double vLNBase = pass.getVLNBase();

		double efd = (*(oldValues.getEfd()))[count];
		double ifd = (*(oldValues.getIfd()))[count];
		double vpss = (*(oldValues.getVpss()))[count];

		double w0 = (*(oldValues.getAngSpeed()))[count]; //not really required as
this value is always equal to the synchronous rotor speed = 376.991 rad/sec
		double genRating = (*(oldValues.getGenRating()))[count];
		double govRating = (*(oldValues.getGovRating()))[count];
		double deltaOld = (*(oldValues.getDeltaOld()))[count];
		vector<vector<double>>* edqOld = oldValues.getEDQOld();
		vector<vector<double>>* idold = oldValues.getIDQOld();

		double check1 = (*idold)[count][0];
		double check2 = (*idold)[count][1];

		double ed0 = (*edqOld)[count][0];
		double eq0 = (*edqOld)[count][1];
		double e00 = (*edqOld)[count][2];

		double pelec = (*(oldValues.getPElec()))[count]; //for governor and PSS
		double pmech = (*(oldValues.getPMech()))[count];
		double pmech_gen = (*(oldValues.getPMechGen()))[count]; //for generator

		double Ts = 0.016666666666666666666666666666666666667;
		double tspan1 = tspan1 = (cycle-1)*Ts/4;;
		double tspan2 = tspan2 = (cycle-1)*Ts/4+Ts/4;
		static vector<std::ofstream> oldValFID(totalGen);
		static vector<std::ofstream> genFID(totalGen);
		static vector<std::ofstream> extFID(totalGen);
		static vector<std::ofstream> govFID(totalGen);
		static vector<std::ofstream> pssFID(totalGen);
		static vector<std::ofstream> voltageFID(totalGen);
		static vector<std::string> num(totalGen);

		//---------------------------------------------------------------------
-------------------------------------------------------------------------------------
------------------------------------------------------------------------------
		//Section 2: Setting up text files to output results
		//---------------------------------------------------------------------
-------------------------------------------------------------------------------------
------------------------------------------------------------------------------

		if(cycle == 1){



			num[count] = std::to_string((long double)(count+1));
			oldValFID[count].open("Old
Values_"+num[count]+".txt",std::ios_base::app);

	genFID[count].open("generator_"+num[count]+".txt",std::ios_base::app);
			pssFID[count].open("pss_"+num[count]+".txt",std::ios_base::app);

	govFID[count].open("governor_"+num[count]+".txt",std::ios_base::app);
			extFID[count].open("exciter_"+num[count]+".txt",std::ios_base::app);

	voltageFID[count].open("voltage_"+num[count]+".txt",std::ios_base::app);

			oldValFID[count]<<setw(16)<<"Gen Rating"<<setw(16)<<"Gov
Rating"<<setw(16)<<"Delta Old"<<setw(16)<<"P-electrical"<<setw(16)<<"P-
mechanical"<<setw(16)<<"P-Mech-Gen"<<setw(16)<<"Ang
Speed"<<setw(16)<<"Ed0"<<setw(16)<<"Eq0"<<setw(16)<<"E00"<<setw(16)<<"id"<<setw(16)<<"iq"
<<setw(16)<<endl;

	voltageFID[count]<<setw(16)<<"Mag(A)"<<setw(16)<<"Mag(B)"<<setw(16)<<"Mag(C)"<<set
w(16)<<"Ang(A)"<<setw(16)<<"Ang(B)"<<setw(16)<<"Ang(C)"<<endl;

			//-----------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
			//Initialize machine parametes
			//-----------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------

			if (genNum == 1){
				double a[17];
				//	if(count == 0){

				std::ifstream file("gen"+num[count]+".txt");

				for(int i=0; i<17;i++){
```

```
                                        file>>a[i];
                                }


        genFID[count]<<setw(16)<<"State1"<<setw(16)<<"State2"<<setw(16)<<"State3"<<setw(16
)<<"State4"<<setw(16)<<"State5"<<setw(16)<<"State6"<<endl;

                                Genrou gen1;

        gen1.setParams(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12]
,a[13],a[14],a[15],a[16]);
                                gen1 = 
initialized[count].initializeNoSat(pass,totalGen,gen1,count);

                                genRating = gen1.getRating();
                                w0 = gen1.getSpeed();
                                deltaOld = gen1.getCalcInit();
                                efd = gen1.getEfd();
                                ifd = gen1.getIfd();


        //System::Windows::Forms::MessageBox::Show(gen1.getIfd().ToString());

                                //pass.setExt(gen1.getIfd(), gen1.getEfd());
                                pass.setGov(gen1.getPelec(), a[0]);

                                pmech = gen1.getPMech();
                                initialized[count].setGenParams(gen1);

                        }

                        else if (genNum == 2){
                        }

                        //myfile >> extType;
                        if (extNum == 2){

                                double a[28];

                                std::ifstream file("ext"+num[count]+".txt");
                                for(int i = 0;i<28;i++){
                                        file>>a[i];
                                }


        extFID[count]<<setw(16)<<"State1"<<setw(16)<<"State2"<<setw(16)<<"State3"<<setw(16
)<<"State4"<<setw(16)<<"State5"<<setw(16)<<"ifd"<<endl;

                                AC7B ext1;

        ext1.setParams(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12]
,a[13],a[14],a[15],a[16],a[17],a[18],a[19],a[20],a[21],a[22],a[23],a[24],a[25],a[26],a[27
]);

                                ext1 = extInitialized[count].initialize(efd, ifd, vt[count],
ext1, count);
                                extInitialized[count].setExtParams(ext1);
```

```
                            //              std::cout<<"Exciter Parameters: "<<"
"<<ext1.getVc()<<"  "<<ext1.getU1avr()<<"  "<<ext1.getYpid1()<<"  "<<ext1.getYpid3()<<"
"<<ext1.getVe()<<endl;
                            //              std::cout<<"Extra Param: Vref
"<<ext1.getVref()<<endl;

                     }

                     if(pssNum == 0){
                            //pass.setVpss(0);
                            vpss = 0;
                     }
                     //myfile >> pssType;
                     //myfile >> govType;

                     if (govNum == 1){

                            double a[36];

                            std::ifstream file("gov"+num[count]+".txt");
                            for(int i = 0;i<36;i++){
                                   file>>a[i];
                            }


       govFID[count]<<setw(16)<<"State1"<<setw(16)<<"State2"<<setw(16)<<"State3"<<setw(16
)<<"State4"<<setw(16)<<"State5"<<setw(16)<<"State6"<<endl;

                            Ggov1 gov1;

       gov1.setParams(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12]
,a[13],a[14],a[15],a[16],a[17],a[18],a[19],a[20],a[21],a[22],a[23],a[24],a[25],a[26],a[27
],a[28],a[29],a[30],a[31],a[32],a[33],a[34],a[35]);

                            govRating = gov1.getMwcap();

                            gov1 = govInitialized[count].initialize(pass,gov1);

                            pmech = gov1.getPmech();
                            //std::cout<<"Governer Params here "<<gov1.getUpeload()<<"
"<<gov1.getYgov1()<<"  "<<gov1.getX()<<"  "<<gov1.getYsup1()<<"  "<<gov1.getYstroke()<<"
"<<gov1.getU1()<<endl;
                            //std::cout<<"Extra Params: Pref & Pmech: "<<gov1.getPref()<<"
"<<gov1.getPmech()<<endl;


                     }

              }

              //----------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
              //setup dq0 values
              //----------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
              setParams[count].setdqParams(tspan1, tspan2, Ts, samples, pass, deltaOld,
genRating, govRating, pmech, pelec, pmech_gen, idold, ed0, eq0, e00,w0,count);
              //std::cout<<"Check idold: "<<(*idold)[count][0]<<"
"<<(*idold)[count][1]<<endl;
              //pelec=(ed0*id0+eq0*iq0)*(genRating)/(govRating);
              //pmech_gen= pmech*govRating/genRating;

              //---------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------
              //solve differential equations
              //---------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------

              if(genNum == 1){

                     //System::Windows::Forms::MessageBox::Show(ed0.ToString());
                     double id0 = (*idold)[count][0];
                     double iq0 = (*idold)[count][1];
                     //System::Windows::Forms::MessageBox::Show("id:");
                     //System::Windows::Forms::MessageBox::Show(id0.ToString());

                     //setting the efd value calculated by the exciter during the last
iteration into the generator object.
                     Genrou newParams = initialized[count].getGenParams();
                     newParams.setEfd(efd);
                     initialized[count].setGenParams(newParams);
                     //System::Windows::Forms::MessageBox::Show(efd.ToString());
                     //Genrou newParams;

                     newParams =  initialized[count].gianeraNoSat(cycle,
initialized[count], pmech_gen, id0, iq0, ed0, eq0, w0, count, totalGen);


       genFID[count]<<setw(16)<<newParams.getEq1()<<setw(16)<<newParams.getPsikd()<<setw(
16)<<newParams.getPsiq1()<<setw(16)<<newParams.getPsiq2()<<setw(16)<<newParams.getSpeed()
<<setw(16)<<newParams.getCalcInit()<<endl;


                     //setting new ifd into the dewParams object-to be used in the extDyn
function
                     //pass.setIfd(newParams.getIfd());
                     ifd = newParams.getIfd();

                     ed0 = newParams.getEd();
                     eq0 = newParams.getEq();
                     (*edqOld)[count][0] = ed0;
                     (*edqOld)[count][1] = eq0;
                     (*edqOld)[count][2] = e00;

                     initialized[count].setGenParams(newParams);
                     deltaOld = newParams.getCalcInit();

              }

              if(extNum == 2){
                    AC7B newExtParams;
                    newExtParams = extInitialized[count].extDyn(extInitialized[count],
efd, ifd, vt[count], vpss, cycle, count, totalGen);


       extFID[count]<<setw(16)<<newExtParams.getVc()<<setw(16)<<newExtParams.getYpid1()<<
setw(16)<<newExtParams.getYpid3()<<setw(16)<<newExtParams.getU1avr()<<setw(16)<<newExtPar
ams.getVe()<<endl;
                    extInitialized[count].setExtParams(newExtParams);

              }

              //-------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
              //create waveforms
              //-------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------


              setParams[count].createWaveform(pass,initialized[count],tspan1, tspan2,
samples,ed0,eq0,e00,w0);

              //System::Windows::Forms::MessageBox::Show(ed0.ToString());
              std::complex<double>** vCplxNewLN = setParams[count].getVCplxNewLN();

              //Althought the variable name reads LL at the end, these are actually just
the L-N voltages
              vMagNewLL[count][0] = (double)abs(vCplxNewLN[0][0])*(vLNBase);
              vMagNewLL[count][1] = (double)abs(vCplxNewLN[0][1])*(vLNBase);
              vMagNewLL[count][2] = (double)abs(vCplxNewLN[0][2])*(vLNBase);

              /* This part used to compute L-L values from L-N values
              std::complex<double>** vCplxNewLL;
              vCplxNewLL = new std::complex<double>*[1];
              for(int i = 0; i<1; i++){
              vCplxNewLL[i] = new std::complex<double>[3];
              }
              for(int i = 0; i<1; i++){
              for(int j =0; j<3; j++){
              vCplxNewLL[i][j]= 0;
              }
              }
              vCplxNewLL[0][0] = (vCplxNewLN[0][0] - vCplxNewLN[0][1])*vLNBase;
              vCplxNewLL[0][1] = (vCplxNewLN[0][1] - vCplxNewLN[0][2])*vLNBase;
              vCplxNewLL[0][2] = (vCplxNewLN[0][2] - vCplxNewLN[0][0])*vLNBase;



              vMagNewLL[0][0] = (double)abs(vCplxNewLL[0][0]);
              vMagNewLL[1][0] = (double)abs(vCplxNewLL[0][1]);
              vMagNewLL[2][0] = (double)abs(vCplxNewLL[0][2]);
              std::cout<<"VMAGNEWLL: \n"<<vMagNewLL[0][0]<<"  "<<vMagNewLL[1][0]<<"
"<<vMagNewLL[2][0]<<endl;
              */
              vector<vector<double>>* vAngLN = setParams[count].getVAng();
```

```
		//Again, athough the variable name reads LL at the end, these are actually 
the L-N angles
		vAngNewLL[count][0] = (*vAngLN)[0][0];
		vAngNewLL[count][1] = (*vAngLN)[0][1];
		vAngNewLL[count][2] = (*vAngLN)[0][2];


	voltageFID[count]<<setw(16)<<vMagNewLL[count][0]<<setw(16)<<vMagNewLL[count][1]<<s
etw(16)<<vMagNewLL[count][2]<<setw(16)<<vAngNewLL[count][0]<<setw(16)<<vAngNewLL[count][1
]<<setw(16)<<vAngNewLL[count][2]<<endl;

		//this part was calculating L-L angles
		//vAngNewLL[0][0] = 
(atan2(vCplxNewLL[0][0].imag(),vCplxNewLL[0][0].real()))*180/M_PI;
		//vAngNewLL[1][0] = 
(atan2(vCplxNewLL[0][1].imag(),vCplxNewLL[0][1].real()))*180/M_PI;
		//vAngNewLL[2][0] = 
(atan2(vCplxNewLL[0][2].imag(),vCplxNewLL[0][2].real()))*180/M_PI;


		double check3 = (*idold)[count][0];
		double check4 = (*idold)[count][1];


	oldValFID[count]<<setw(16)<<genRating<<setw(16)<<govRating<<setw(16)<<deltaOld<<se
tw(16)<<pelec<<setw(16)<<pmech<<setw(16)<<pmech_gen<<setw(16)<<w0<<setw(16)<<ed0<<setw(16
)<<eq0<<setw(16)<<e00<<setw(16)<<check3<<setw(16)<<check4<<endl;

		//---------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-
		//Setting the new values in old values object
		//---------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-

		(*(oldValues.getEfd()))[count] = efd;
		(*(oldValues.getIfd()))[count] = ifd;
		(*(oldValues.getVpss()))[count] = vpss;
		(*(oldValues.getAngSpeed()))[count] = w0; //Don't really need to pass this 
value as this is the reference angular speed always equal to 376.991 rad/sec.
		(*(oldValues.getGenRating()))[count] = genRating;
		(*(oldValues.getGovRating()))[count] = govRating;
		(*(oldValues.getDeltaOld()))[count] = deltaOld;
		//vector<vector<double>>* edqOld = oldValues.getEDQOld() = edqOld;
		//vector<vector<double>>* idold = oldValues.getIDQOld();

		(*(oldValues.getPElec()))[count] = pelec; //for governor and PSS
		(*(oldValues.getPMech()))[count] = pmech;
		(*(oldValues.getPMechGen()))[count] = pmech_gen; //for generator


		count = count+1;
		//if (cycle == 12 || cycle == 5000 || cycle == 8000 || cycle == 10000){
		//System::Windows::Forms::MessageBox::Show(vMagNewLL[0].ToString());
		//}

	}
```

```
}

       void ManagedTutorialApp::runSystem(PCMPSYS pSys, DateTime analysisTime, bool
suppressDialogs, bool stopAfterFirstFailedCkt)
       {
              IPowerFlowApp^ powerFlow =
(IPowerFlowApp^)DEWAppManager::getGUIApp(IPowerFlowApp::APP_NAME, pSys);

              powerFlow->runSystem(pSys, analysisTime, true, false);
              PSYSwrapper^ dewSys = PSYSwrapper::getSysWrapper(pSys);
              PCMPCKT pFirstCircuit = dewSys->getCktWrapList(true, true, true)[0]->pCkt;
              IResultsExchange^ resultsExch = powerFlow-
>getCircuitExchange(pFirstCircuit, analysisTime);

              ICMPFilter^ sourceCmpFilter = gcnew ICMPFilter(ICMP_IPS_CKTFDR);
              getch();
              // AP Modifications Begin

              double *matPtr=nullptr; // this pointer will be used to access the real
data returned by MATLAB as mxArray data type
              double *matPtr1=nullptr; // this pointer will also be used to access the
real data returned by MATLAB as mxArray data type
              double *matPtr2=nullptr; // this pointer will also be used to access the
real data returned by MATLAB as mxArray data type
              double totalCycles=7200;
              const int SAMPLES=7200; //this value should be the same as totalCycles
              double totalGen2=3; // we had to define a double totalGen with same value
as consta int totalGen1 because int values cannot be passed to matlab using mxArray
              const int totalGen1=3; //this number will determine the size of voltage and
current arrays
              const int totalFileNames=36; //this number should be calculated as
toatlGenX4(number of components for each plant)X3(this array will hold function
handles,parameter filenames and output filenames)

              vector<vector<double>> option(4,vector<double>(totalGen1));
              vector<vector<double>> iAbsOld(3,vector<double>(totalGen1));
              vector<vector<double>> iAngleOld(3,vector<double>(totalGen1));
              vector<vector<double>> vAbsOld(3,vector<double>(totalGen1));
              vector<vector<double>> vAngleOld(3,vector<double>(totalGen1));
              vector<vector<double>> vMagNew(totalGen1, vector<double>(3));
              vector<vector<double>> vAngNew(totalGen1, vector<double>(3));

              //later we need to read option values from the DEW model - separatelty for
each machine and assign that option vector to that machine structure.
              for(int j = 0; j<totalGen1; j++){
                     option[0][j] = 1;
                     option[1][j] = 2;
                     option[2][j] = 0;
                     option[3][j] = 1;
              }

              CComplex3Ph oldLoadKVACplx;
              CDouble3Ph loadReal, loadImag; // Variables to store the real and imaginary
parts of modified loads
```

```
		char IPSUID1[][5]={"1","2","3"}; //this list will ensure that power supply
is associated with correct dynamic models. This list will be in the same order as in the
gen_spec file. We could modify this approach later to avoid hard coding it.
		char *addr=IPSUID1[0];
		String^ IPSUID=gcnew String(addr);
		double vBase=132.7906;
		double systemMVABase=100; // Since DEW is sending actual voltages to
MATLAB, this value is not very important but must be given nonetheless
		int counter=0;
		int i=0;

		CDouble3Ph vMagKvNew[totalGen1];
		CDouble3Ph vAngDegNew[totalGen1];
		CDouble3Ph newLoadKVAReal;
		CDouble3Ph newLoadKVAImag;

		//setting OldValues to be passed:
		//Trying a single oldValues
		vector<double> genRating(totalGen1);
		vector<double> govRating(totalGen1);
		vector<double> deltaOld(totalGen1);
		vector<vector<double>> edOld(totalGen1, vector<double>(3));
		vector<vector<double>> idOld(totalGen1, vector<double>(3));
		vector<double> pElec(totalGen1);
		vector<double> pMech(totalGen1);
		vector<double> pMechGen(totalGen1);
		vector<double> w0(totalGen1);
		vector<double> efd(totalGen1);
		vector<double> ifd(totalGen1);
		vector<double> vpss(totalGen1);

		passValues oldies(&genRating, &govRating, &deltaOld, &edOld, &idOld,
&pElec, &pMech, &pMechGen, &w0, &efd, &ifd, &vpss);

		int sampleNum = 0;
		div_t divresult;
		dewParams pass;
		vector<dqParams> setParams(totalGen1);
		vector<genInit> genInitialize(totalGen1);
		vector<extInit> extInitialize(totalGen1);
		vector<govInit> govInitialize(totalGen1);
		//vector<genStates> genStateCalc(totalGen1);

		for (int iter=1;iter<totalCycles+1;iter++)
		{
			divresult = div (iter,4);
			sampleNum = divresult.rem;
			if (sampleNum == 0){
				sampleNum = 4;
			}
			// what we have done below is to pass the handles, and all relevant
filenames once and for all to Matlab using the copying_filenames function. This way we
did not have to copy the strings into C++
			// Also, since the extraParam parameter in the main function needed
to sent only once and is not accessed after that, copying_filenames function does not
need to return the list of filenames and handles
			if (iter==1)
			{
```

```
                            //for each(PCMPwrapper^ powerSourceCmp in gcnew
CktCmpWrapIterator(dewSys->getCircuitIterator(true, true, true), sourceCmpFilter))
                            for (counter=0; counter<totalGen1;counter++)
                            {
                                   char *addr=IPSUID1[counter];
                                   IPSUID=gcnew String(addr);
                                   PCMPwrapper^ powerSourceCmp = (PCMPwrapper^)dewSys-
>findComponentWithUID(IPSUID,true);
                                   if (powerSourceCmp != nullptr)
                                   {
                                          Double3Ph iMagAmps = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_IMAGA);
                                          Double3Ph iAngDeg = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_IANGDG);
                                          Double3Ph vMagKv = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_VMAGKV);
                                          Double3Ph vAngDeg = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_VANGDG);
                                          iAbsOld[0][counter] = iMagAmps[A_PH]; //A_PH is
a pre-processor directive
                                          iAbsOld[1][counter] = iMagAmps[B_PH];
                                          iAbsOld[2][counter] = iMagAmps[C_PH];
                                          iAngleOld[0][counter] = iAngDeg[A_PH];
                                          iAngleOld[1][counter] = iAngDeg[B_PH];
                                          iAngleOld[2][counter] = iAngDeg[C_PH];

                                          vAbsOld[0][counter] = vMagKv[A_PH]; //A_PH is a
pre-processor directive
                                          vAbsOld[1][counter] = vMagKv[B_PH];
                                          vAbsOld[2][counter] = vMagKv[C_PH];
                                          vAngleOld[0][counter] = vAngDeg[A_PH];
                                          vAngleOld[1][counter] = vAngDeg[B_PH];
                                          vAngleOld[2][counter] = vAngDeg[C_PH];

                                          genInit initialized;
                                          extInit extInitialized;
                                          govInit govInitialized;
                                          dqParams setParam;

                                          genInitialize[counter] = initialized;
                                          extInitialize[counter] = extInitialized;
                                          govInitialize[counter] = govInitialized;
                                          setParams[counter] = setParam;

                                   }
                                   else
                                   {

       System::Windows::Forms::MessageBox::Show("Gen_spec_file doesn't match with Power
Supply UIDs in DEW");
                                   }
                            }
                            counter=0;
                            //set dew parameters:
                            pass.setDewParams(&option, iter, &vAbsOld, &vAngleOld,
&iAbsOld, &iAngleOld, vBase, systemMVABase);
```

```
                            mainfunc(oldies, genInitialize, extInitialize, govInitialize,
pass, sampleNum, setParams, vMagNew, vAngNew);


                            for (int j=0;j<totalGen1;j++)
                            {

                                   vMagKvNew[j][A_PH] = vMagNew[j][0];
                                   vMagKvNew[j][B_PH] = vMagNew[j][1];
                                   vMagKvNew[j][C_PH] = vMagNew[j][2];

                                   vAngDegNew[j][A_PH] = vAngNew[j][0];
                                   vAngDegNew[j][B_PH] = vAngNew[j][1];
                                   vAngDegNew[j][C_PH] = vAngNew[j][2];

                            }

                            //for each(PCMPwrapper^ powerSourceCmp in gcnew
CktCmpWrapIterator(dewSys->getCircuitIterator(true, true, true), sourceCmpFilter))
                            for (counter=0; counter<totalGen1;counter++)
                            {
                                   char *addr=IPSUID1[counter];
                                   IPSUID=gcnew String(addr);
                                   //System::Windows::Forms::MessageBox::Show(IPSUID);
                                   PCMPwrapper^ powerSourceCmp = (PCMPwrapper^)dewSys-
>findComponentWithUID(IPSUID,true);
                                   if (powerSourceCmp != nullptr)
                                   {

       IndependentPowerSourcePart::setSourceVpu(powerSourceCmp->pCmp, vMagKvNew[counter],
vAngDegNew[counter]);
                                   }
                            }

                            UnRegisterApp(-1, TRUE, NULL);
                            powerFlow->runSystem(pSys, analysisTime, true, false); // run
powerflow after an iteration is complete in MATLAB
                            resultsExch = powerFlow->getCircuitExchange(pFirstCircuit,
analysisTime);
                     }
                     else
                     {
                            for (counter=0; counter<totalGen1;counter++)
                            {
                                   char *addr=IPSUID1[counter];
                                   IPSUID=gcnew String(addr);
                                   PCMPwrapper^ powerSourceCmp = (PCMPwrapper^)dewSys-
>findComponentWithUID(IPSUID,true);
                                   if (powerSourceCmp != nullptr)
                                   {
                                          Double3Ph iMagAmps = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_IMAGA);
                                          Double3Ph iAngDeg = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_IANGDG);
                                          Double3Ph vMagKv = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_VMAGKV);
                                          Double3Ph vAngDeg = resultsExch-
>getValues(powerSourceCmp->pCmp, IPowerFlowApp::IVAR_VANGDG);
```

```
                                          iAbsOld[0][counter] = iMagAmps[A_PH]; //A_PH is 
a pre-processor directive
                                          iAbsOld[1][counter] = iMagAmps[B_PH];
                                          iAbsOld[2][counter] = iMagAmps[C_PH];
                                          iAngleOld[0][counter] = iAngDeg[A_PH];
                                          iAngleOld[1][counter] = iAngDeg[B_PH];
                                          iAngleOld[2][counter] = iAngDeg[C_PH];

                                          vAbsOld[0][counter] = vMagKv[A_PH]; //A_PH is a 
pre-processor directive
                                          vAbsOld[1][counter] = vMagKv[B_PH];
                                          vAbsOld[2][counter] = vMagKv[C_PH];
                                          vAngleOld[0][counter] = vAngDeg[A_PH];
                                          vAngleOld[1][counter] = vAngDeg[B_PH];
                                          vAngleOld[2][counter] = vAngDeg[C_PH];

                                   }
                                   else
                                   {
       System::Windows::Forms::MessageBox::Show("Gen_spec_file doesn't match with Power 
Supply UIDs in DEW");
                                   }
                            }

                            PCMPwrapper^ anotherSourceCmp = (PCMPwrapper^)dewSys-
>findComponentWithUID("ld6",true); // get load pointer
                            if (iter==1000000) //creating a disturbance
                            {
       oldLoadKVACplx=LoadData::getSpotLoadKVA(anotherSourceCmp->pCmp); // we could use 
the class name "LoadData" to access the getSpotLoad and setSpot Load functions as these 
are static member functions and can be accessed even if no object of the class is 
created.
                                   CDouble3Ph loadMag=oldLoadKVACplx.getMagnitudes(); 
//get A,B and C phase load magnitudes
                                   CDouble3Ph loadAng=oldLoadKVACplx.getAnglesRad(); //get 
A,B and C phase load angles

                                   loadReal[A_PH]=loadMag[A_PH]*cos(loadAng[A_PH])+1500; 
// Adding 4.5 to ld6's real power and 1.5 to the reactive power increases total KVA by 5% 
from the base value while keeping the power factor same.
                                   loadReal[B_PH]=loadMag[B_PH]*cos(loadAng[B_PH])+1500;
                                   loadReal[C_PH]=loadMag[C_PH]*cos(loadAng[C_PH])+1500;

                                   loadImag[A_PH]=loadMag[A_PH]*sin(loadAng[A_PH])+500;
                                   loadImag[B_PH]=loadMag[B_PH]*sin(loadAng[B_PH])+500;
                                   loadImag[C_PH]=loadMag[C_PH]*sin(loadAng[C_PH])+500;



                                   CComplex3Ph newLoadKVACplx(loadReal,loadImag);
                                   LoadData::setSpotLoadKVA(anotherSourceCmp-
>pCmp,newLoadKVACplx);


                            }
                            if (iter==24800) //restoring load
                            {

                                   CDouble3Ph loadMag=oldLoadKVACplx.getMagnitudes();
//get A,B and C phase load magnitudes
                                   CDouble3Ph loadAng=oldLoadKVACplx.getAnglesRad(); //get
A,B and C phase load angles
                                   CDouble3Ph loadReal, loadImag; // Variables to store
the real and imaginary parts of modified loads

                                   LoadData::setSpotLoadKVA(anotherSourceCmp-
>pCmp,oldLoadKVACplx);
                            }



                            pass.setDewParams(&option, iter, &vAbsOld, &vAngleOld,
&iAbsOld, &iAngleOld, vBase, systemMVABase);

                            mainfunc(oldies, genInitialize, extInitialize, govInitialize,
pass, sampleNum, setParams, vMagNew, vAngNew);

                            for (int j=0;j<totalGen1;j++)
                            {

                                   vMagKvNew[j][A_PH] = vMagNew[j][0];
                                   vMagKvNew[j][B_PH] = vMagNew[j][1];
                                   vMagKvNew[j][C_PH] = vMagNew[j][2];

                                   vAngDegNew[j][A_PH] = vAngNew[j][0];
                                   vAngDegNew[j][B_PH] = vAngNew[j][1];
                                   vAngDegNew[j][C_PH] = vAngNew[j][2];
                            }

                            for (counter=0; counter<totalGen1;counter++)
                            {
                                   char *addr=IPSUID1[counter];
                                   IPSUID=gcnew String(addr);
                                   PCMPwrapper^ powerSourceCmp = (PCMPwrapper^)dewSys-
>findComponentWithUID(IPSUID,true);
                                   if (powerSourceCmp != nullptr)
                                   {

       IndependentPowerSourcePart::setSourceVpu(powerSourceCmp->pCmp, vMagKvNew[counter],
vAngDegNew[counter]);
                                   }
                            }

                            UnRegisterApp(-1, TRUE, NULL);
                            powerFlow->runSystem(pSys, analysisTime, true, false); // run
powerflow after an iteration is complete in MATLAB
                            resultsExch = powerFlow->getCircuitExchange(pFirstCircuit,
analysisTime);

                     }
              }
```

```
	}//end of runSystem()
```

Header files for individual functions:

dewParams.h

```
#ifndef DEW_PARAMS_H
#define DEW_PARAMS_H

#include <vector>

using namespace std;


class dewParams{
private:
	vector<vector<double>>* opt;
	int cycle;
	vector<vector<double>>* iAngOld;
	vector<vector<double>>* iMagOld;
	vector<vector<double>>* vAngOld;
	vector<vector<double>>* vMagOld;
	double sysMVABase;
	double vLNBase;
	vector<double>* vt;
	double pelec;
	double sbgen;


public:
	dewParams();
	void setVt(vector<double>* vt);
	void setDewParams(vector<vector<double>>* opt,int cycle,vector<vector<double>>*
vMagOld,vector<vector<double>>* vAngOld,vector<vector<double>>*
iMagOld,vector<vector<double>>* iAngOld,double vLNBase,double sysMVABase);
	vector<vector<double>>* getOpt();
	int getCycle();
	vector<vector<double>>* getVMagOld();
	vector<vector<double>>* getVAngOld();
	vector<vector<double>>* getIMagOld();
	vector<vector<double>>* getIAngOld();
	double getVLNBase();
	double getSysMVABase();
	vector<double>* getVt();
	void setGov(double pelec, double sbgen);
	double getPelec();
	double getSbgen();

};

#endif
```

dqParams.h

```
#ifndef DQPARAMS_H
```

```
#define DQPARAMS_H
#define _USE_MATH_DEFINES
#define NUM 6;

#include <complex>
#include <string>
#include "dewParams.h"
#include <iostream>
#include <cmath>
#include<stdio.h>
#include<stdlib.h>
#include <iomanip>

using namespace std;

class dqParams{
private:
vector<vector<double>> X1;
vector<vector<double>>* X2ptr;
vector<vector<double>> X2;
vector<vector<double>> vABCNew;
vector<vector<double>>* vABCNewptr;
vector<vector<double>> vAngNew;
vector<vector<double>>* vAngNewptr;
std::complex<double>** vCplxNewLN;

public:
void setdqParams(double tspan1, double tspan2, double Ts, double samples, dewParams
params, double deltaOld, double genRating, double govRating, double pmech, double& pelec,
double& pmech_gen, vector<vector<double>>* idold, double& ed0, double& eq0, double& e00,
double w0, int count);
void createWaveform(dewParams params, double calcInit, double speed, double ed, double
eq, double tspan1, double tspan2, double samples, double& ed0, double& eq0, double& e00,
double w0);
vector<vector<double>>* getVMag();
vector<vector<double>>* getVAng();
complex<double>** getVCplxNewLN();

};
#endif
```

passValues.h

```
#ifndef PASS_VALUES_H
#define PASS_VALUES_H

#include <vector>

using namespace std;

class passValues{

private:
	vector<double>* genRating;
	vector<double>* govRating;
	vector<double>* deltaOld;
	vector<vector<double>>* edOld;
	vector<vector<double>>* idOld;
	vector<double>* pElec;
	vector<double>* pMech;
	vector<double>* pMechGen;
	vector<double>* w0;
	vector<double>* efd;
	vector<double>* ifd;
	vector<double>* vpss;


public:
	passValues();
	passValues(vector<double>* genRate, vector<double>* govRate, vector<double>*
delta, vector<vector<double>>* edold, vector<vector<double>>* idold, vector<double>*
pelec, vector<double>* pmech, vector<double>* pmech_gen, vector<double>* speed,
vector<double>* efd, vector<double>* ifd, vector<double>* vpss);
	void setPassValues(vector<double>* genRate, vector<double>* govRate,
vector<double>* delta, vector<vector<double>>* edold, vector<vector<double>>* idold,
vector<double>* pelec, vector<double>* pmech, vector<double>* pmech_gen, vector<double>*
speed, vector<double>* efd, vector<double>* ifd, vector<double>* vpss);
	vector<double>* getGenRating();
	vector<double>* getGovRating();
	vector<double>* getDeltaOld();
	vector<double>* getPElec();
	vector<double>* getPMech();
	vector<double>* getPMechGen();
	vector<double>* getAngSpeed();
	vector<vector<double>>* getEDQOld();
	vector<vector<double>>* getIDQOld();
	vector<double>* getEfd();
	vector<double>* getIfd();
	vector<double>* getVpss();

};
#endif
```

numInit.h

```
#ifndef NUM_INT_H
#define NUM_INT_H

#include <vector>
#include <cmath>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

using namespace std;

void numInt(vector<double>* dx, double tstep, vector<double>* x01, int size);

#endif
```

Genrou.h

```
#ifndef GENROU_H
#define GENROU_H
#define _USE_MATH_DEFINES
```

```
#include <cmath>
#include <string>
#include <iostream>

using std::endl;

class Genrou
{
private:
	double eq1;
	double psikd;
	double psiq1;
	double psiq2;
	double w0;
	double calcInit;
	double rating;
	double T1d0;
	double T2d0;
	double T1q0;
	double T2q0;
	double H;
	double D;
	double Xd;
	double Xq;
	double X1d;
	double X1q;
	double X2d;
	double X2q;
	double Xl;
	double S1;
	double S2;
	double Ra;
	double A;
	double B;
	double ifd;
	double efd;
	double pelec;
	double pmech;
	double ed;
	double eq;

public:
	Genrou();
	~Genrou();
	Genrou(double rat, double T1d0, double T2d0, double T1q0, double T2q0, double H, 
double D, double Xd, double Xq, double X1d, double X1q, double X2d, double X2q, double 
Xl, double S1, double S2, double Ra);
	void setParams(double rat, double T1d0, double T2d0, double T1q0, double T2q0, 
double H, double D, double Xd, double Xq, double X1d, double X1q, double X2d, double X2q, 
double Xl, double S1, double S2, double Ra);
	double getRating();
	double getA();
	double getXq();
	double getX1d();
	double getX1q();
	double getX2d();
	double getX2q();
```

```
        double getXl();
        double getB();
        double getRa();
        double getXd();
        void setA(double newA);
        void setB(double newB);
        void setEq1(double eq);
        void setPsikd(double pkd);
        void setPsiq1(double pq1);
        void setPsiq2(double pq2);
        void setW0(double w0);
        void setCalcInit(double y);
        void setIfd(double ifd);
        void setEfd(double efd);
        void setPelec(double pelec);
        void setPMech(double pmech);
        double getSpeed();
        double getEq1();
        double getPsikd();
        double getPsiq1();
        double getPsiq2();
        double getCalcInit();
        double getIfd();
        double getEfd();
        double getPelec();
        double getPMech();
        double getT1d0();
        double getT2d0();
        double getT1q0();
        double getT2q0();
        double getH();
        double getD();
        double getS1();
        double getS2();
        double getEd();
        double getEq();
        void setEd(double Ed);
        void setEq(double Eq);

};

#endif
```

genInit.h

```
#ifndef GEN_INIT_H
#define GEN_INIT_H
#define _USE_MATH_DEFINES

#include "dewParams.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <conio.h>
#include "Genrou.h"
#include "genStates.h"
```

```
#include <cmath>
#include "numInt.h"
#include <iomanip>

using std::endl;

class genInit:public Genrou{

private:
	double eq1;
	double psikd;
	double psiq1;
	double psiq2;
	double w0;
	double calcInit;
	double ifd;
	double efd;
	double pelec;
	double rating;
	double ed;
	double eq;
	Genrou generatoryo;

public:
	genInit();
	~genInit();
	Genrou initialize(dewParams params,double totalGen, Genrou generator, int count);
	Genrou initializeNoSat(dewParams params,double totalGen, Genrou generator, int
count);
	Genrou getGenParams();
	void setGenParams(Genrou gen);
	genStates gianera(double cycle, genInit initialized, double pmech, double id,
double iq, double ed, double eq);
	Genrou gianeraNoSat(double cycle, genInit initialized, double pmech, double id,
double iq, double ed, double eq, double& wr, int count, int totalGen);

};

#endif
```

AC7B.h

```
#ifndef AC7B_H
#define AC7B_H
#define _USE_MATH_DEFINES

#include <iostream>
#include <string>
#include <cmath>

using std::string;

class AC7B
{
private:
	double vc;
	double ypid1;
	double ypid3;
	double u1avr;
	double ve;
	double Tr;
	double Kpr;
	double Kir;
	double Kdr;
	double Tdr;
	double Vrmax;
	double Vrmin;
	double Kpa;
	double Kia;
	double Vamax;
	double Vamin;
	double Kp;
	double Kl;
	double Te;
	double Vfemax;
	double Vemin;
	double Ke;
	double Kc;
	double Kd;
	double Kf1;
	double Kf2;
	double Kf3;
	double Tf;
	double E1;
	double SE1;
	double E2;
	double SE2;
	double Spdmlt;
	double vref;
	double A;
	double B;


public:
	AC7B();
	~AC7B();
	void setParams(double tr, double kpr, double kir, double kdr, double tdr, double 
vrmax, double vrmin, double kpa, double kia, double vamax, double vamin, double kp, 
double kl, double te, double vfemax, double vemin, double ke, double kc, double kd, 
double kf1, double kf2, double kf3, double tf, double e1, double se1, double e2, double 
se2, double spdmlt);

	double getVc();
	double getYpid1();
	double getYpid3();
	double getU1avr();
	double getVe();
	double getVref();

	void setVe(double ve);
	void setVc(double vc);
	void setYpid1(double ypid1);
	void setYpid3(double ypid3);
```

```
        void setU1avr(double u1avr);
        void setVref(double vref);

        //void setInitParams(extInit initialized);
        double getTr();
        double getKpr();
        double getKir();
        double getKdr();
        double getTdr();
        double getKpa();
        double getKia();
        double getKp();
        double getKl();
        double getTe();
        double getKe();
        double getKc();
        double getKd();
        double getKf1();
        double getKf2();
        double getKf3();
        double getTf();
        double getE1();
        double getSe1();
        double getE2();
        double getSe2();
        double getA();
        double getB();
        double getVrmax();
        double getVrmin();
        double getVamax();
        double getVamin();
        double getVfemax();
        double getVemin();
};

#endif
```

extInit.h

```
#ifndef EXT_INIT_H
#define EXT_INIT_H
#include "dewParams.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <conio.h>
#include "AC7B.h"

using std::endl;

class extInit:public AC7B{

private:

        double vc;
        double ypid1;
```

```
	double ypid3;
	double u1avr;
	double ve;
	double vref;
	AC7B exciter;

public:
	extInit();
	AC7B initialize(double efd, double ifd, double vt, AC7B e1, int count);
	void setExtParams(AC7B ext);
	AC7B getExtParams();
	double getVc();
	double getYpid1();
	double getYpid3();
	double getU1avr();
	double getVe();
	double getVref();
	AC7B extDyn(extInit initialized, double& efd, double ifd, double vt, double vpss,
int cycle, int count, int totalGen);

};

#endif
```

Ggov1.h

```
#ifndef GGOV1_H
#define GGOV1_H
#define _USE_MATH_DEFINES

#include <iostream>
#include <string>
#include <cmath>

using std::string;

class Ggov1
{
private:

	double Upeload;
	double Ygov1;
	double x;
	double Ysup1;
	double Ystroke;
	double U1;
	double Mwcap;
	double R;
	double Rselect;
	double Tpelec;
	double Maxerr;
	double Minerr;
	double Kpgov;
	double Kigov;
	double Kdgov;
	double Tdgov;
	double Vmax;
```

```
        double Vmin;
        double Tact;
        double Kturb;
        double Wfnl;
        double Tb;
        double Tc;
        double Flag;
        double Teng;
        double Tfload;
        double Kpload;
        double Kiload;
        double Ldref;
        double Dm;
        double Ropen;
        double Rclose;
        double Kimw;
        double Pmwset;
        double Aset;
        double Ka;
        double Ta;
        double Db;
        double Tsa;
        double Tsb;
        double Rup;
        double Rdown;
        double Pref;
        double Pmech;

public:
        Ggov1();
        ~Ggov1();
        void setParams(double mwcap, double r, double rselect, double tpelec, double
maxerr, double minerr, double kpgov, double kigov, double kdgov, double tdgov, double
vmax, double vmin, double tact, double kturb, double wfnl, double tb, double tc, double
flag, double teng, double tfload, double kpload, double kiload, double ldref, double dm,
double ropen, double rclose, double kimw, double pmwset, double aset, double ka, double
ta, double db, double tsa, double tsb, double rup, double rdown);
        double getR();
        double getMwcap();
        double getTpelec();
        double getWfnl();
        double getTb();
        double getTc();
        double getFlag();
        double getKturb();
        double getSbturb();
        double getRselect();
        double getMaxerr();
        double getMinerr();
        double getKpgov();
        double getKigov();
        double getKdgov();
        double getVmax();
        double getVmin();
        double getTdgov();
        double getTact();
        double getTeng();
        double getTfload();
        double getKpload();
        double getKiload();
        double getLdref();
        double getDm();
        double getRopen();
        double getRclose();
        double getKimw();
        double getPmwset();
        double getAset();
        double getKa();
        double getTa();
        double getDb();
        double getTsa();
        double getTsb();
        double getRup();
        double getRdown();


        double getUpeload();
        void setUpeload(double upeload);
        double getYgov1();
        void setYgov1(double ygov1);
        double getX();
        void setX(double x);
        double getYsup1();
        void setYsup1(double ysup1);
        double getYstroke();
        void setYstroke(double ystroke);
        double getU1();
        void setU1(double u1);
        double getPref();
        void setPref(double pref);
        double getPmech();
        void setPmech(double pmech);

};

#endif
```

govInit.h

```
#ifndef GOV_INIT_H
#define GOV_INIT_H
#include "dewParams.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <conio.h>
#include "AC7B.h"
#include "Ggov1.h"
#include "numInt.h"

#using <system.dll>
#using <system.xml.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>
```

```
using namespace System;
using namespace System::Xml;
using namespace System::Collections::Generic;
using namespace System::Drawing;
using namespace System::Windows::Forms;

using std::endl;

class govInit:public Ggov1{

private:

	double Upeload;
	double Ygov1;
	double x;
	double Ysup1;
	double Ystroke;
	double U1;
	double Pref;
	double Pmech;
	Ggov1 governor;

public:
	govInit();
	~govInit();
	Ggov1 initialize(dewParams params, Ggov1 governer);
	void setGovParams(Ggov1 gov);
	Ggov1 getGovParams();
	double getUpeload();
	double getYgov1();
	double getX();
	double getYsup1();
	double getYstroke();
	double getU1();
	double getPref();
	double getPmech();
	Ggov1 govDyn(govInit initialized, double& pmech, double pelec, double wr, int
cycle, int count, int totalGen);
};

#endif
```