Cyprus University of Technology
Department of Electrical Engineering and Computer Engineering and Informatics

Doctoral Dissertation

Advanced Computational Methods for Digital Controller Integration in Power System Dynamic Simulations

Mehran Jafari

Limassol, July 2025

CYPRUS UNIVERSITY OF TECHNOLOGY

Faculty of Engineering and Technology

Department of Electrical Engineering, Computer Engineering, and Informatics

Doctoral Dissertation

Advanced Computational Methods for Digital Controller Integration in Power System Dynamic Simulations

Mehran Jafari

Advisor:

Dr Petros Aristidou

Limassol, July 2025

Advisory Committee

Doctoral Dissertation

Advanced Computational Methods for Digital Controller Integration in Power System Dynamic Simulations

Presented by

Mehran Jafari

Supervisor: Department of Electrical Engineering, Computer Engineering and Informatics, Petros Aristidou, Assistant Professor
Member of the committee: Name Surname and position
Member of the committee: Name, Surname and position

Cyprus University of Technology
Limassol, July 2025

Copyrights

Acknowledgements

We humans rely on words to communicate meaning to one another. However, there are situations or experiences in life that their description is beyond common words. The last four years of my life working as a PhD student under the supervision of Dr. Petros Aristidou is a perfect example of such an experience. He patiently provided invaluable suggestions, motivation, and training to remove all the obstacles in my path. I am deeply thankful for the countless hours of meetings, discussing technical research-related and even life-related issues of the day. Overall, He is a brilliant advisor, and I express my sincerest gratitude to him for his extraordinary support during my studies.

I would like to thank my co-advisors at RTE, G. Bureau, M. Chiaramello, A. Guironnet, and P. Panciatici for their kind support and help. Their critical insights have improved and led my research continuously over the years. I also thank RTE for funding my work, which made this research possible.

Last but not least, I express my special thanks to my beloved parents, whose great endeavors and sacrifices throughout my life brought me this far. Therefore, I humbly dedicate this thesis to them for selflessly supporting me for a lifetime. I also would like to thank my brother, who is filling my absence at home with his warm, kind heart.

Finally, I am grateful to the kind people of Cyprus who treated me with generosity, warmth, and genuine hospitality throughout my studies.

ABSTRACT

Dynamic simulation of power systems provides the system's response to a disturbance, which is a critical tool for optimally operating it and ensuring security. However, with increasing technological advancements and interconnecting neighbor networks, today's power systems have become a complex interconnected network of heterogeneous devices. Some traditional devices, such as synchronous machines, have slower responses, and some others, such as electronic-based devices, have faster responses. This leads to a system modeled with numerically stiff differential-algebraic equations. Furthermore, the existence of different kinds of components in the system, such as breakers and digital controllers, makes the system hybrid, i.e., discontinuities arise during the simulation that must be handled carefully. Thus, dynamic power system simulation requires solving numerous sets of nonlinear, stiff, hybrid Differential-Algebraic Equations (DAEs).

In the case of hybrid systems, the real challenge is the integration over discontinuities since the solver must reduce the time step and land on the discontinuity, ensuring an accurate solution. Digital controllers are a classic source of discontinuities, introducing one discontinuity per sampling action. Therefore, the simulation of power systems with multiple digital controllers causes numerous discontinuities over the simulation horizon. Stopping at each sampling action and reducing the time steps constantly is challenging, leading to a slow and computationally inefficient simulation.

In this thesis, first, the modeling of digital controllers and its challenges is discussed, then, the methods capable of simulating them are reviewed. A family of methods based on interpolation is proposed to tackle the issue of the simulation of power systems with digital controllers. The proposed methods are compared to already existing methods in terms of accuracy and performance. It is shown that, for example, the interpolation-based method is more than 16 times faster than the conventional step-reduction method for a system with 9 digital controllers. This performance gap between the methods expands as the number of controllers increases, since the conventional method's time steps become more limited while the proposed method takes relatively large time steps.

Furthermore, Many variations of the proposed methods are also developed, suited for different situations. For example, a light version of the interpolation-based method is devised for the simulation of computationally heavy controllers, which is about 40 percent faster than the original interpolation-based approach. Also, a novel approach for simplified simulation of power systems with digital controllers is proposed, which improves the accuracy of the traditional simplified simulations. Last but not least, a variation of the proposed method is also developed, which is based on solving the controller inputs instead of its outputs. The simulations revealed that this approach improves the performance over the original method by about 18 percent. Finally, an implementation of the interpolation-based approach using the Modelica language is provided to make it accessible.

It should be noted that all the proposed interpolation-based methods have the advantage of being able to solve systems with non-equation-based digital controllers, such as machine-learning-based controllers with a variable time step approach, which is not possible with any conventional methods. In other words, the conventional methods cannot simulate the non-equation-based digital controllers quickly and accurately, as either they have to reduce the time steps constantly to catch the sampling actions of the

controllers, or ignore some events to speed up. Furthermore, conventional variable-step solvers that rely on modeling the controllers with continuous equations will fail to model and simulate those types of controllers, as the y cannot be described with DAEs.

Keywords: Digital controller, discontinuity handling, interpolation methods, power system simulation, differential-algebraic equations, Jacobian approximation, hybrid systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SRM	Step Reduction Method
ATM	Analog Treatment Method
SSM	Simplified Simulation Method
IBM	Interpolation-Based Method
DAE	Differential-Algebraic Equation
SEXS	Simplified Excitation System model
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
ZOH	Zero-Order Hold
AW	Anti-Windup
IQI	Inverse Quadratic Interpolation
IVP	Initial Value Problem
ODE	Ordinary Differential Equation
HVDC	High-Voltage Direct Current
AVR	Automatic Voltage Regulator
EQAVR	Equation-based Automatic Voltage Regulator
FAVR	Fuzzy logic based Automatic Voltage Regulator
MLAVR	Machine-Learning based Automatic Voltage Regulator
DIBM	Decoupled Interpolation-Based Method
LIBM	Light Interpolation-Based Method
SIBM	Simplified Interpolation-Based Method
MJIBM	Moved Jacobian Interpolation-Based Method
SMIB	Single-Machine Infinite-Bus
PV	Photovoltaic
EMT	Electromagnetic Transient

1 Introduction

1.1 Motivation

Dynamic simulations under the phasor approximation are a cornerstone of modern power system analysis, used globally to evaluate system response to large-scale disturbances such as faults, line outages, or sudden generation loss. Over the past few decades, these simulations have evolved into indispensable tools for a wide range of stakeholders involved in the planning, design, operation, and protection of electric power systems. They provide a practical and computationally efficient means to study electromechanical dynamics over time scales ranging from seconds to minutes, making them particularly well-suited for transient stability studies and control system evaluation. Power system operators depend on high-fidelity, real-time simulations not only for analyzing contingencies and assessing the dynamic security of the grid, but also for training control room personnel and conducting comprehensive what-if scenario analyses. These simulations play a critical role in the reliable scheduling of day-ahead operations by helping operators understand how the system will behave under varying load and generation conditions. On the planning side, engineers and researchers use dynamic simulations to investigate the long-term impacts of proposed infrastructure developments, including the addition of new transmission corridors, the integration of intermittent renewable energy sources, the deployment of advanced grid-forming inverters, and the retirement of legacy thermal plants. As power systems become increasingly complex and decentralized, the role of dynamic simulation is only set to grow, supporting the transition to a more resilient, sustainable, and digitally controlled energy grid.

Dynamic simulations of electric power systems require the numerical solution of large sets of nonlinear, stiff, and hybrid Differential-Algebraic Equations (DAEs) that capture the system's physical dynamics, component interactions, and embedded control schemes. These DAEs represent the behavior of synchronous machines, power electronic converters, loads, protection systems, and controllers, among others. In large-scale interconnected transmission networks, the system can easily involve hundreds of thousands of equations. These equations span a wide range of time scales, from sub-millisecond electromagnetic transients to slower electromechanical oscillations lasting several seconds. Additionally, the presence of discrete events, triggered by protection devices, limiters, tap changers, switching actions, or digital controllers, introduces hybrid behavior that further complicates the simulation process. The resulting numerical problem is not only highly stiff and nonlinear but also subject to frequent discontinuities, which makes robust and accurate integration particularly challenging. As a result, dynamic simulations are computationally demanding, often requiring sophisticated solvers, adaptive time-stepping techniques, and significant computing resources. For large networks or high-fidelity models, simulations can easily push the limits of standard computing platforms, especially when real-time or near-real-time performance is needed for control room applications or hardware-in-the-loop testing.

With the ongoing modernization of power systems, virtually all contemporary control and protection devices are implemented digitally, introducing new layers of complexity into dynamic simulation studies. These devices range from classical control schemes, such as proportional-integral (PI) or proportional-integral-derivative (PID) controllers, to more advanced and adaptive strategies based on artificial intelli-

gence (AI), optimization algorithms, and machine learning techniques. Their integration into large-scale dynamic simulations has become increasingly important, not only to validate the performance of individual devices but also to analyze their interactions with the broader power system under various operating conditions and disturbance scenarios. However, incorporating digital controllers into traditional simulation frameworks is far from straightforward. Most power system dynamics are modeled using continuous DAEs, which describe the continuous evolution of voltages, currents, rotor angles, and other system states. In contrast, digital controllers operate in discrete time, executing control logic at specific sampling intervals. This mismatch in time representation creates a hybrid system dynamic that is difficult to model and simulate accurately. Embedding discrete-time logic within continuous DAE-based solvers requires careful treatment of signal sampling, event detection, and synchronization between continuous and discrete components. These challenges are further amplified when multiple digital devices are involved, each with different sampling rates, delays, and logic paths. As such, accurate simulation of modern power systems demands advanced modeling techniques and hybrid numerical solvers that can seamlessly handle both continuous dynamics and discrete events.

Digital controllers operate by sampling signals at fixed intervals, processing them, and updating control outputs in discrete time steps. The digital sampling actions of controllers introduce discontinuities into the otherwise smooth trajectory of the simulation. Each time a controller samples and a signal updates its output, a discrete event occurs, potentially causing abrupt changes in control signals or triggering switching actions. In systems with many digital controllers, such as wide-area protection schemes, distributed energy resource inverters, or modern grid-forming controls, these discrete actions can number in the hundreds or even thousands during a single simulation window. Each of these events must be precisely detected and handled to maintain numerical stability and ensure the accuracy of the results. Failure to properly account for these discontinuities can lead to missed events, incorrect system behavior, or convergence issues in the numerical solver. Consequently, simulating such systems requires hybrid simulation techniques with robust event-handling mechanisms and tight synchronization between discrete-time control logic and continuous-time system dynamics.

1.2 Aims and Objectives

This work begins by investigating the modeling and integration of digital controllers within power system dynamic simulations. It addresses the unique challenges that arise when simulating digital control systems, characterized by their discrete-time behavior, alongside the continuous-time dynamics of power system components. Particular emphasis is placed on understanding how these challenges affect the accuracy, stability, and computational performance of the overall simulation. The study then delves into the critical issue of handling discontinuities, which are introduced primarily by the sampling actions and logic switching inherent in digital control. Various types of discontinuities are identified and categorized, along with a discussion of the specific numerical and modeling difficulties they introduce. Existing approaches in the literature for managing such hybrid dynamics are reviewed and assessed in terms of their effectiveness and limitations. Building on this foundation, the work proposes novel methods to address the identified challenges, aiming to enhance both simulation accuracy and computational efficiency.

First, a novel interpolation-based method is proposed for handling numerous discontinuities arising

from the operation of digital controllers. The proposed approach is devised to increase the performance of the simulation of power systems under control by digital controllers by taking relatively large time steps and integrating over multiple discontinuities. This is achieved by interpolating the feedback of digital controllers at sampling points of time and refining the controller outputs alongside the system's state variables in the Newton solver.

A comprehensive Jacobian investigation is performed on the proposed method to identify the best performance choice. Furthermore, a decoupled version of the proposed method is introduced with a simplified Jacobian built, and consequently, improved performance.

A light version of the proposed approach is proposed for handling the computationally demanding controllers during the simulation. These types of controllers are expensive to call, and a light approach that calls them the minimum number of times in each time step can lead to a significant performance increase. This light method covers the overcalling issue of the proposed method, which leads to a slow simulation while facing computationally demanding controllers.

A simplified simulation method designed specifically for the simulation of digital controllers is also proposed. This method covers the accuracy issues that the traditional simplified method suffers from when facing multiple digital controllers.

Finally, another variation of the interpolation-based method that moves the Jacobian construction to the controller input instead of its output is proposed. This new approach has a few advantages, such as not introducing any new type of variables to the solver since the controller inputs are the system's state variables only at different points in time.

Finally, an open-source Modelica implementation of the proposed interpolation-based method is devised to make it accessible and evaluate its application in simulation software packages.

1.3 Structure of the Thesis

Chapter 2 In this chapter, various modeling approaches for digital controllers are examined, with a focus on their impact on the accuracy and performance of power system dynamic simulations. The trade-offs between model complexity, computational burden, and simulation fidelity are discussed in detail. To illustrate the practical implications of these modeling choices, two case studies are presented. In each case, the dynamic response of the system is simulated using both analog (continuous-time) and digital (discrete-time) controllers under identical operating conditions. The results are then compared to highlight key differences in system behavior, numerical stability, and computational performance. These case studies provide valuable insight into how the choice of controller modeling strategy can influence the simulation outcomes and why it is vital to stay loyal to the real controller's nature while modeling it.

Chapter 3 Handling discontinuities in power system dynamic simulations presents significant challenges, both in terms of computational complexity and numerical accuracy. Discontinuities must be reliably detected at each simulation time step. Once identified, their precise locations must be determined with high accuracy to ensure the fidelity of the simulation results. Moreover, appropriate numerical techniques must be employed to bridge the solution across the discontinuity, preserving the physical integrity and stability of the simulated system response. This chapter presents a comprehensive review of various

methodologies for discontinuity detection, location, and treatment as reported in the existing literature. Additionally, complex and often non-trivial scenarios that may emerge during each stage of discontinuity handling, such as simultaneous events, rapid sequences of discontinuities, and numerical instability, are thoroughly discussed. Potential strategies and algorithmic enhancements proposed in prior research to mitigate these challenges are also presented.

Chapter 4 In this chapter, classical methods for simulating digital controllers within power system dynamic simulations are first categorized and systematically reviewed. While effective in some contexts, these methods often impose limitations on efficiency and accuracy, particularly when dealing with large-scale systems featuring numerous digital controllers. To address these challenges, a novel method is proposed for efficiently handling time events in systems with a high density of digital control elements. The proposed approach introduces an interpolation-based treatment of discrete events within each simulation time step. Crucially, this method allows for the accurate modeling of digital controller actions without interrupting the ongoing simulation or requiring a reduction in the time step. As a result, it preserves the advantages of variable time-step integration techniques and avoids the performance degradation typically associated with fine-grained time-step enforcement. This approach focuses specifically on the first stage of discontinuity treatment by embedding event information seamlessly into the simulation process. The second stage, which involves bridging the solution across the event, remains dependent on the specific numerical integration method employed.

Chapter 5 This chapter extends the proposed method to a variety of methods, each suitable for specific simulations. In summary, a simplified simulation approach appropriate for the simplified simulation of power systems with digital controllers is proposed. In addition, a light version of the proposed approach is discussed that is able to quickly solve systems with computationally demanding digital controllers. Furthermore, a modified version of the proposed method is devised that is based on solving the controller inputs instead of the outputs, which offers several numerical advantages.

Chapter 6 This chapter provides an implementation of the proposed approach using Modelica, an open-source simulation platform. The Modelica version of the proposed method is a fixed-step approach that integrates the proposed method inside the controller blocks. In this way, only the controller blocks are replaced with new custom-made blocks and the rest of the system and solvers remain the same.

Chapter 7 In this chapter, the main contributions of the thesis are summarized, highlighting the key findings and advancements made in the area of power system dynamic simulation and digital controller modeling. The effectiveness and applicability of the proposed methods are reviewed in light of the challenges addressed throughout the work. Additionally, several potential directions for future research are outlined.

Overall, this thesis builds upon the following material, which has been published, submitted, or is currently under preparation for various journals and conferences:

Published:

- [7] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Modeling of Digital Controllers in Electric Power System Dynamic Simulations*, in 2023 IEEE Belgrade PowerTech, 2023.
- [8] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Methods for*

Incorporating Digital Controllers in Power System Dynamic Simulations, Electric Power Systems Research, 2024.

- [9] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Decoupled Interpolation-based Method for Numerical simulation of Digital Controllers*, in 2024 IEEE SysCon, Montreal, 2024.
- [10] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *A Moelica IBM Implementation for Fast Simulation of Digital Controllers in Power Systems*, in 2024 IEEE OSMSES, 2024.
- [11] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Handling of Computationally Demanding Digital Controllers in Power System Dynamic Simulations*, in 2024 IEEE SyNERGY MED, 2024.
- [12] **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Fast and Accurate Simulation of Smart Digital Controllers in Power System Dynamic Studies*, Sustainable Energy, Grids, and Networks, 2025.

Under preparation:

- **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Moved Jacobian Interpolation-based Method for Numerical Simulation of Digital Controllers in Power System Dynamic Studies*.
- **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Dynamic Simulation of Power Systems with Digital Controllers: A Review of Challenges and Methods*.

Under review:

- **M.Jafari**, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, P. Aristidou. *Simplified Simulation of Digital Controllers in Power System Dynamic Studies*, Submitted to 2026 Power Systems Computation Conference.

The code files related to the published papers can be found in a GitHub repository [1]. The reader can refer to Appendix II for more information.

[1] https://github.com/SPS-L/Discrete-controller-handling

2 Digital Controller Modeling in Power System Dynamic Simulations

Digital controllers are a serious source of discontinuities in power system dynamic simulations. Due to their sampling, they introduce numerous time events, each needing to be handled effectively to ensure accuracy.

Traditionally, controllers are modeled using differential-algebraic equations (DAEs), just like other continuous components of a system. These equations are combined with the system's DAEs and solved together. While this method works well for analog controllers, it does not accurately represent digital controllers because it overlooks the effects of sampling. As a result, simulations involving digital controllers may produce inaccurate results. Despite this limitation, the method remains popular in large-scale simulations due to its simplicity and computational efficiency.

In the rest of this section, different approaches for modeling controllers and their impact on accuracy and performance are investigated. Furthermore, two case studies are utilized to compare the simulation results of a system with analog and digital controllers.

2.1 Controller modeling

The modeling of digital controllers heavily affects the results of the dynamic simulation of the system under control. The modeling choice of the controller changes both the accuracy and performance of the simulation. Therefore, modeling the controller effectively and as closely as possible to the real component is vital. This section describes the methods used for modeling controllers in power system time-domain simulations and investigates the impact of modeling on the simulation accuracy and performance [13].

2.1.1 Block diagrams and transfer functions

Block diagrams are the most commonly used representation in power systems literature and standards to illustrate controller structures, primarily due to their simplicity and clarity (e.g., [2, 14]). For example, Fig. 2.1 shows a simplified excitation system model (SEXS), where the controller's inputs, intermediate states, and their respective functions can be easily identified.

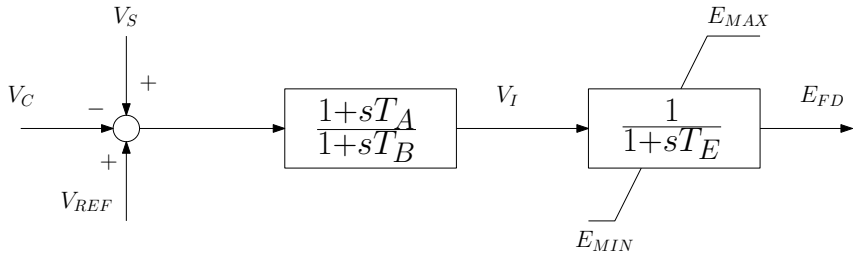

Figure 2.1: Block diagram of SEXS controller that can be used for controlling a synchronous generator voltage output [2]

Despite their clarity, this simplicity has often led to variations in how models are implemented across

different simulation software, resulting in notable discrepancies in system responses. This is largely because block diagrams seldom specify the detailed implementation of certain features, such as anti-windup integrator limits, selection flags, or the handling of violations. To address these inconsistencies, several technical reports and standards have been updated to provide clearer guidance on implementing even basic and widely used control blocks, such as the IEEE anti-windup integrator [15].

This form of representation is inherently linked to the s-domain transfer functions of the individual components of the controller. For example, based on the block diagram shown in Fig. 2.1, the transfer function equations corresponding to the SEXS controller can be derived—under the assumption that the limits (E_{MIN}, E_{MAX}) are not active—as follows:

$$E_{FD}(s) = \left(\frac{1 + sT_A}{1 + sT_B}\right)\left(\frac{1}{1 + sT_E}\right) e(s) \tag{2.1}$$

where $e(s)$ denotes the input and $E_{FD}(s)$ the output.

Both block diagram and transfer function representations are based on an input-output framework, in which outputs are explicitly defined as functions of the inputs. These models adhere to a causal modeling approach, and the block-oriented paradigm they embody has served as the foundation for the vast majority of general-purpose modeling languages used in physical system modeling.

2.1.2 Differential-algebraic equations

To circumvent certain limitations of causal modeling and enable integration of controllers into large-scale power system dynamic simulation tools, an acausal modeling approach can be adopted. In this framework, models are represented using undirected differential-algebraic equations (DAEs), which enhances their reusability and composability, which are key benefits for tackling the complexities of large-scale simulation. This acausal paradigm is embraced by modern modeling languages such as Modelica [16]. For example, equation (2.1) can be reformulated as:

$$\begin{aligned} T_E\dot{E}_{FD}(t) &= V_I(t) - E_{FD}(t) \\ T_B\dot{V}_t(t) &= e(t) - V_t(t) \\ 0 &= T_A e(t) + (T_B - T_A)V_t(t) - T_B V_I(t) \end{aligned} \tag{2.2}$$

where $V_t(t)$ is a temporary variable.

It can be seen that the notion of input-output has been removed from this representation. However, the controller equations can now be used in a simulation software package based on solving DAEs.

2.1.3 Discrete domain and difference equations

Most controllers in use today are digital. To enable model-based control design or evaluate system performance, security, and stability, a hybrid analysis (such as the one illustrated in Fig. 2.2) is required. In this setup, the analog-to-digital converter (ADC) samples selected variables from the controlled system, denoted as $y(t)$, at a fixed sampling interval T_s, producing discrete-time signals $y(k)$ that are fed into the digital controller. The controller generates a discrete output $u(k)$, which is passed through a digital-

to-analog converter (DAC) to produce a continuous signal that actuates the system. In most practical applications, the DAC is usually a Zero-order Hold (ZOH) model [17]. The ADC and DAC blocks also perform a quantization on the amplitude of the sampled signal. The range of the amplitude of the signal is divided into a discrete number of smaller parts with specific different values. This allows for converting a continuous signal into a discrete one with a precision specified by the number of bits that determines the number of small intervals [18], consequently, the converted signal's precision.

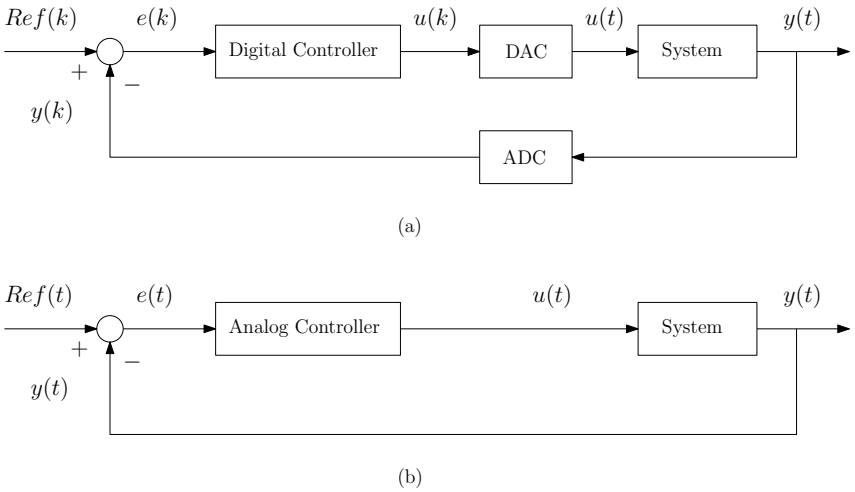

Figure 2.2: Interface of controllers in a (a) hybrid and (b) continuous system analysis

A discrete-time representation of a controller can be readily derived from its continuous-time transfer function using various discretization techniques, such as Zero-Order Hold (ZOH), First-Order Hold, Forward or Backward Difference, and the Trapezoidal method (also known as Tustin's approximation). For example, applying the Backward Difference method to equation (2.1) yields:

$$E_{FD(z)} = \frac{zT_s(z(T_s+T_A)-T_A)}{(z(T_s+T_B)-T_B)\,(z(T_s+T_E)-T_E)}e_{(z)} \tag{2.3}$$

Consequently, difference equations are more useful to implement the controller in simulation software packages. Therefore, the following equation will be derived:

$$\begin{aligned} E_{FD(k+1)} =& \frac{1}{(T_s+T_B)(T_s+T_E)}[E_{FD(k)}(T_E(T_s+2T_B) \\ &+ T_sT_B) - E_{FD(k-1)}(T_BT_E) \\ &+ e_{(k+1)}(T_s(1+T_A)) - e_{(k)}(T_sT_A)] \end{aligned} \tag{2.4}$$

The output of the formula at time $t = t_k = kT_s$ is held constant over the period $t \in [kT_s, (k+1)T_s)$ (assuming a ZOH DAC). These equations are then finally implemented in the control loop of the digital controller hardware.

2.2 Digital controller simulation challenges

Over the past few decades, power system controls, whether locally deployed on individual components or implemented as wide-area solutions, have largely transitioned to digital platforms, with all newly

installed controllers now being digital by default. This widespread adoption of digital control introduces considerable challenges for power system modeling and simulation. In particular, the discrete nature of digital control actions, stemming from their formulation using difference equations and governed by their sampling intervals, leads to the injection of thousands of discrete events into the system simulation[19]. These time events can modify the DAEs (state events) too. Figure 2.3 illustrates the discrete-time events occurring in a small system equipped with three digital controllers operating at different sampling rates. Each bar in the figure marks an instance when at least one controller samples its input. In this example, the controllers have sampling intervals of 0.1 s, 0.12 s, and 0.15 s. When such a system is simulated using a standard variable time-step solver, these discrete events force the simulation to frequently reduce the time-step and momentarily halt before resuming, thereby increasing the overall computational burden. They also make the simulation significantly slow due to the constantly reduced time steps. Managing the large number of discontinuities introduced by digital controllers remains an ongoing challenge and often requires specialized, time-intensive techniques.

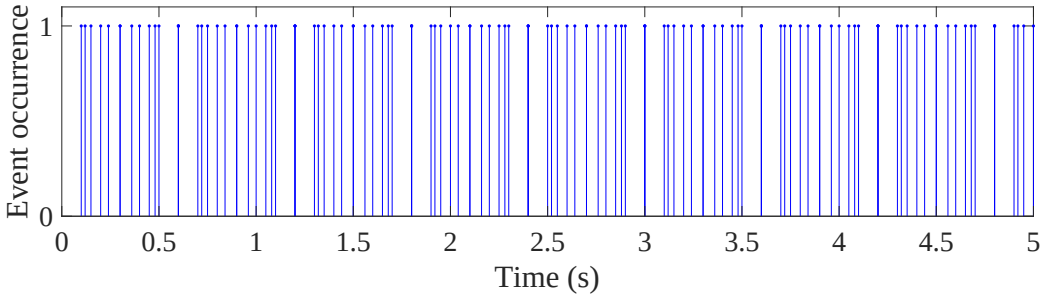

Figure 2.3: Discontinuities occurrence in a system controlled by three digital controllers with 0.1, 0.12, and 0.15 seconds of sampling times, during 5 seconds of simulation

A commonly adopted strategy to overcome this challenge is to represent digital controllers using their continuous-time equivalents [7]. This approach helps eliminate discrete time events, enabling the use of traditional analysis techniques, such as variable time-step solvers, that can significantly improve simulation efficiency. However, accurately replicating the behavior of digital controllers, along with their associated ADC and DAC converters, within a continuous DAE framework is challenging. The continuous approximation may fail to capture key characteristics of the actual digital implementation. Moreover, this analog representation can introduce artificial issues not present in the real system, particularly due to time delays introduced by ZOH mechanisms in digital controllers. For example, simulations may experience deadlocks caused by limits on controller variables, as demonstrated in the IEEE Anti-Windup (AW) controller case studies in [20, 21], which is not the case in digital controller implementation. Additionally, some types of digital controllers (such as those based on optimization algorithms or artificial intelligence) cannot be modeled with analog equivalents.

2.3 Simulation results to showcase the problem

To illustrate the problem more clearly, several case studies are presented in the following sections. The first one examines the simulation of an IEEE 421.5-2016 AW PI controller in both open-loop and closed-loop configurations, comparing its analog implementation with digital versions using various sampling

intervals. Next, a 3-bus power system equipped with an Automatic Voltage Regulator (AVR) and a governor is analyzed, again contrasting analog and digital control implementations under similar conditions.

All simulations shown below employ a predictor-corrector integration scheme that combines a second-order Adams-Bashforth (explicit) method with a second-order Adams-Moulton (implicit) method. The Milne's estimate is used for computing the local truncation error, and the Newton formula is utilized for solving DAEs [22]. Unless stated otherwise, the integration step size is fixed at 1 ms across all simulations, and the sampling intervals of the digital controllers are selected to be integer multiples of the simulation time step. Additionally, both the ADC and the ZOH DAC use a uniform 16-bit quantization.

2.3.1 PI controller formulation

PI controllers are widely used in power systems for many applications due to their simple structure and tuning process. The IEEE recommended 421.5-2016 AW PI controller shown in Figure 2.4, formulation as a DAE model [15], including the limits, is given by:

$$\begin{aligned} &\text{If} \quad y \geq w_{max} : w = w_{max} \quad \text{and} \quad \dot{x} = 0 \\ &\text{If} \quad y \leq w_{max} : w = w_{min} \quad \text{and} \quad \dot{x} = 0 \\ &\text{Otherwise} \quad \dot{x} = k_i u \quad \text{and} \quad w = y = k_p u + x \end{aligned} \tag{2.5}$$

where y and w are the controller outputs before and after the limit, respectively. Also, x denotes the internal state variable of the controller. k_i and k_p are the controller blocks' parameter and u is its input. Finally, the AW maximum and minimum limits are defined using w_{max} and w_{min}.

Denoting sampling time by T, and for simplicity, discretizing the controller using the Forward Difference method results:

$$\begin{aligned} &\text{If} \quad y_k \geq w_{max} : w_{k+1} = w_{max} \quad \text{and} \quad x_{k+1} = x_k \\ &\text{If} \quad y_k \leq w_{max} : w_{k+1} = w_{min} \quad \text{and} \quad x_{k+1} = x_k \\ &\text{Otherwise} \quad x_{k+1} = k_i u_k T + x_k \quad \text{and} \\ &\qquad\qquad w_{k+1} = y_{k+1} = k_p u_{k+1} + x_{k+1} \end{aligned} \tag{2.6}$$

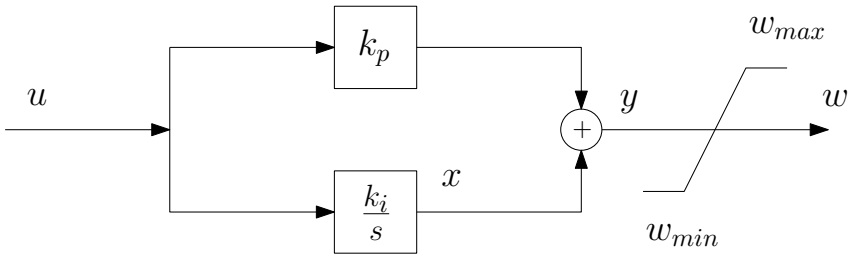

Figure 2.4: Block diagram representation of IEEE 421.5-2016 AW PI controller

It should be noted that the equation set (2.5) will be used for analog simulations of the PI controller, while (2.6) will be used for its digital version.

2.3.2 Open-loop PI controller test system model

First, a simple open-loop example is considered in which the input u is varied over a 6.5 seconds horizon as (2.7). The parameters of the controllers are set to $k_i = 3$ and $k_p = 1$ and the limits are set to $w_{min} = -1.2$ and $w_{max} = 1.2$.

$$\text{If} \quad t < 3: \quad \dot{u}(t) = 1; \qquad \text{Else}: \quad \dot{u}(t) = -1 \tag{2.7}$$

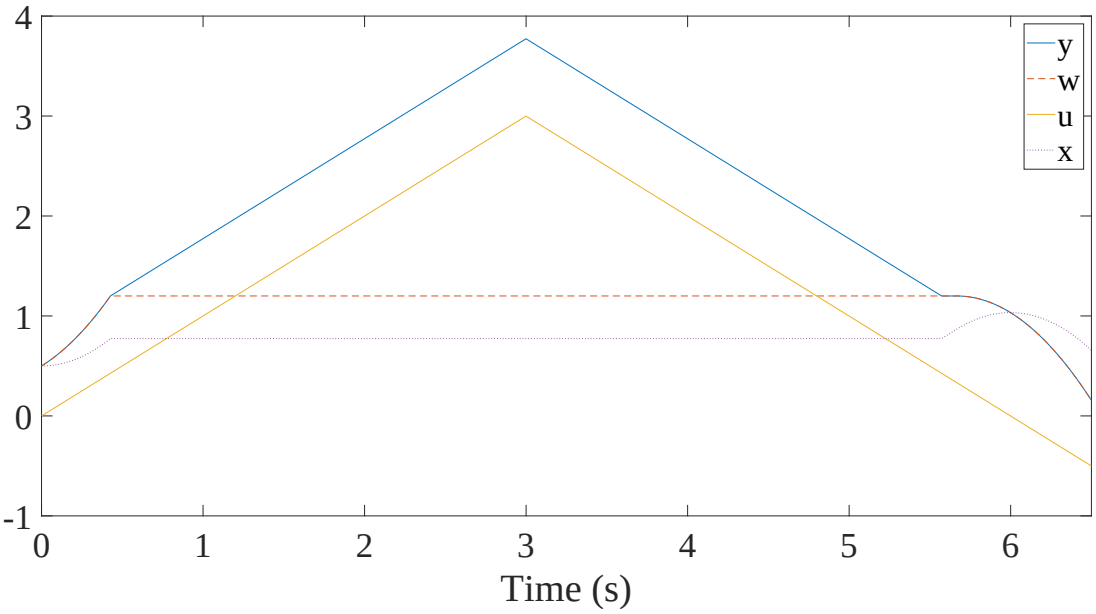

Figure 2.5: Results of simulation of the analog AW PI controller in the open-loop test system

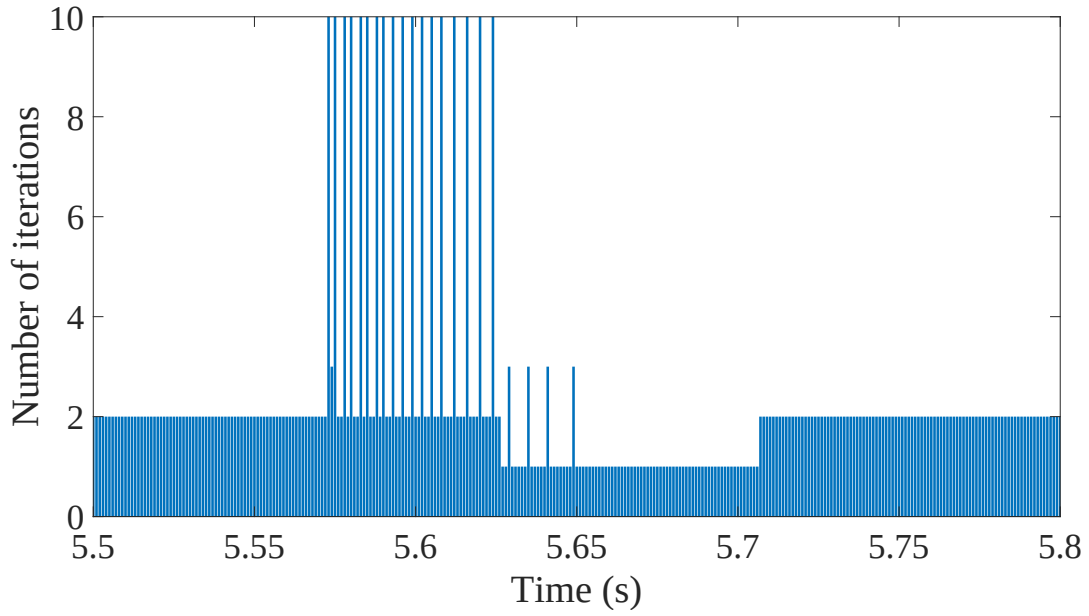

Figure 2.6: Number of Newton iterations for each time step between the times 5.56 and 5.66 s for the simulation of the analog AW PI controller in the open-loop test system

The results of different variables of the controller for a 6.5-second simulation are illustrated in Fig 2.5. The input initially drives the controller output y beyond its upper limit, then it returns and remains exactly on the limit for some time. This numerical issue, known as deadlock [15], causes the controller output to fluctuate as the controller switches back and forth between two states. This happens since around $t = 5.57$, y goes below the maximum limit w_{max} and tends to decrease while the input u is still positive, pushing the controller output to increase. This leads to locking and unlocking x in each Newton iteration of the solver as shown in Fig 2.6, which lists the number of Newton iterations for each time step. It should be noted that the maximum number of Newton iterations allowed is set to 10 before the solver is forced to move to the next time step or repeat the current one based on the value of the estimated error.

The same simulation is repeated with the digital version of the controller with different sampling times. The controllers' output y are illustrated in Fig 2.7, and its zoomed version in Fig 2.8. When the sampling time is equal to 1 ms, which is also equal to the time step size, the digital controller output resembles the analog one. This happens since the equations of the analog controllers (2.5) are discretized by the integration method, which has the time steps landing on the controller's sampling times. However, as the sampling time changes, the output varies. This is due to the fact that the equation set of the digital controller updates at each $T > h$ instead of every time step for the analog controller.

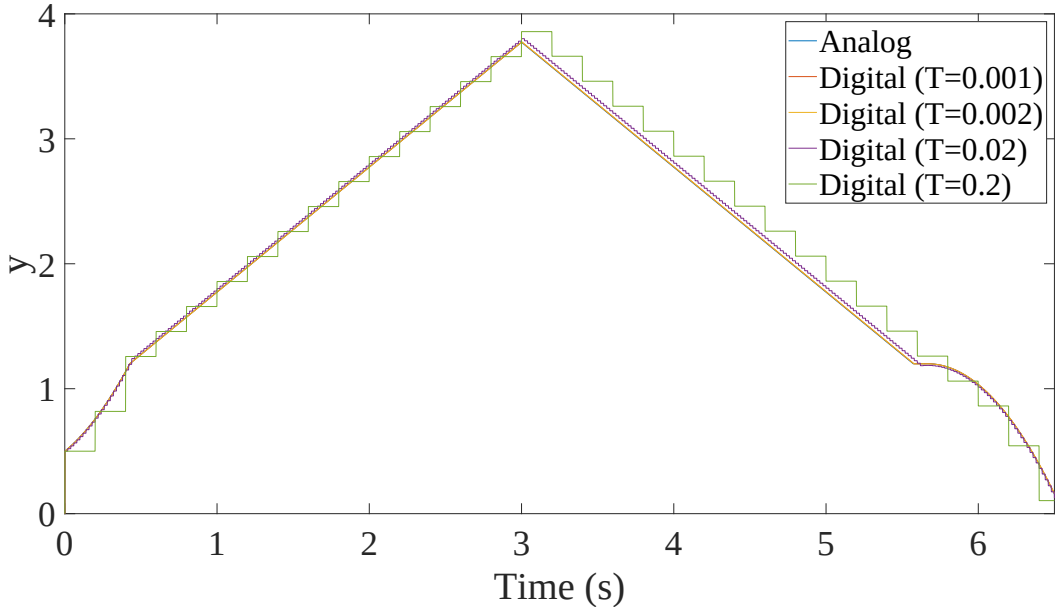

Figure 2.7: Results of simulation of the digital AW PI controller with different sampling times in the open-loop test system

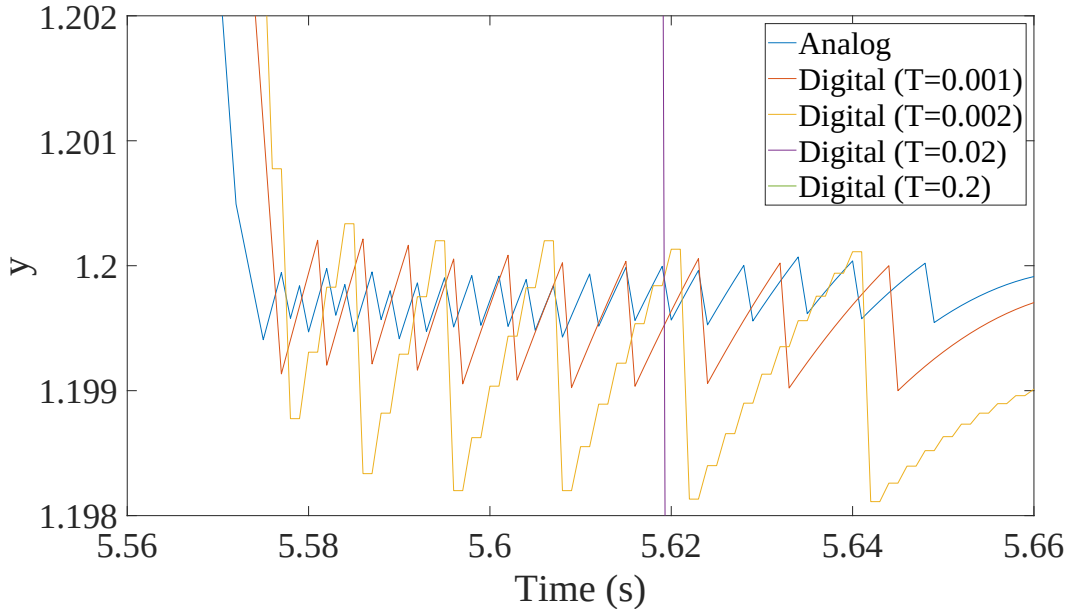

Figure 2.8: Results of simulation of the digital AW PI controller with different sampling times between 5.56 and 5.66 s in the open-loop test system

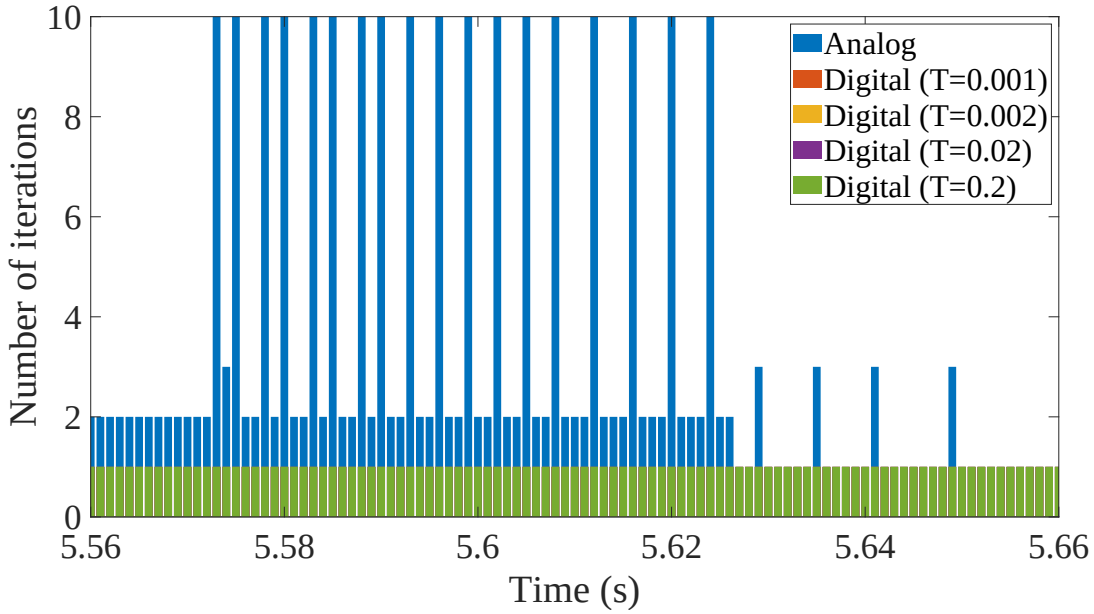

Figure 2.9: Number of Newton iterations for the continuous and digital AW PI controller with different sampling times between 5.56 and 5.66 s in the open-loop test system (all digital controllers have only one iteration)

Furthermore, chattering is observed across all models, with its amplitude varying depending on the sampling time. Nevertheless, as illustrated in Fig. 2.9, which presents the number of Newton iterations required for both the continuous and digital controllers, the digital controllers demonstrate significantly better computational performance. Additionally, the likelihood of a deadlock occurring in the digital controller models is eliminated. This is attributed to the use of a zero-order hold (ZOH) approach for updating the limits, which inherently introduces a time delay.

2.3.3 Closed-loop PI controller test system model

In this case study, the PI controller is utilized for driving a simple continuous system with two differential equations as shown in Fig. 2.10:

$$\begin{bmatrix} \dot{z_1} \\ \dot{z_2} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} -b \\ 0 \end{bmatrix} w \tag{2.8}$$

The parameters of the controller are kept the same as the previous case study, and $k_f = 0.1$. In addition, the parameters of the continuous system are set equal to $a = -0.1$ and $b = 0.9$.

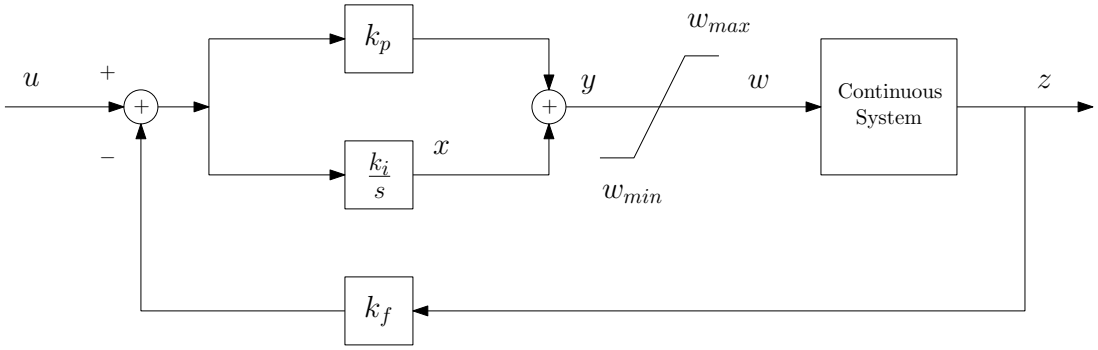

Figure 2.10: A simple continuous system controlled by the digital PI controller

First, the same input variation defined in (2.7) is applied. The output z_2 of the system is depicted in Fig. 2.11 for analog and digital controllers with different sampling times. It can be seen again that for $T \leq h$, the responses of the digital controller and the analog one are identical. However, a significant variation is observed for $T > h$. Therefore, simulating the systems that are under control by digital devices must be modeled and simulated numerically using the digital controller's model defined by difference equations and not the continuous one. Otherwise, the effect of the sampling rate of the digital controller will be overlooked, and the wrong analog output will be obtained.

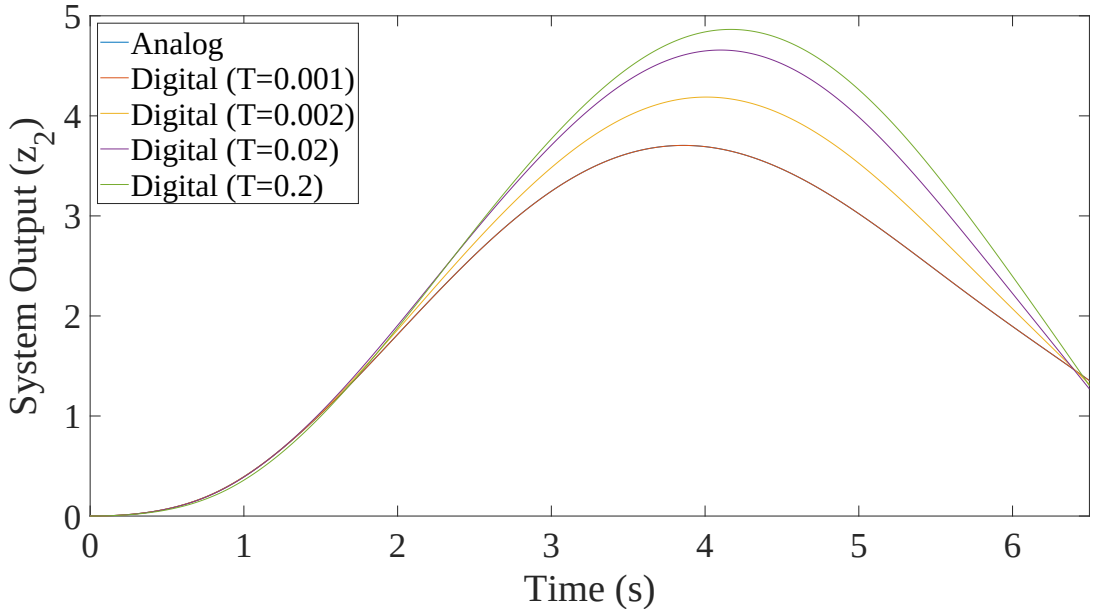

Figure 2.11: System's output z_2 for the analog and digital controllers with different sampling times in the closed-loop test system (first scenario)

In the second scenario, instead of (2.7), a unit step change is applied to $u = 0 \rightarrow 1$ with $k_p = 2$, $k_i = 1$, $k_f = 1$, $a = -0.2$, $b = 0.9$, $w_{min} = -1.1$, and $w_{max} = 1.1$. The response is illustrated in Fig. 2.12 and the controllers' output in Fig. 2.13, respectively. It can be seen that even in this simplified LTI ODE system, a change in the sampling time T can provide different simulation results. In addition, the activation and deactivation of the limits are shifted according to the sampling time.

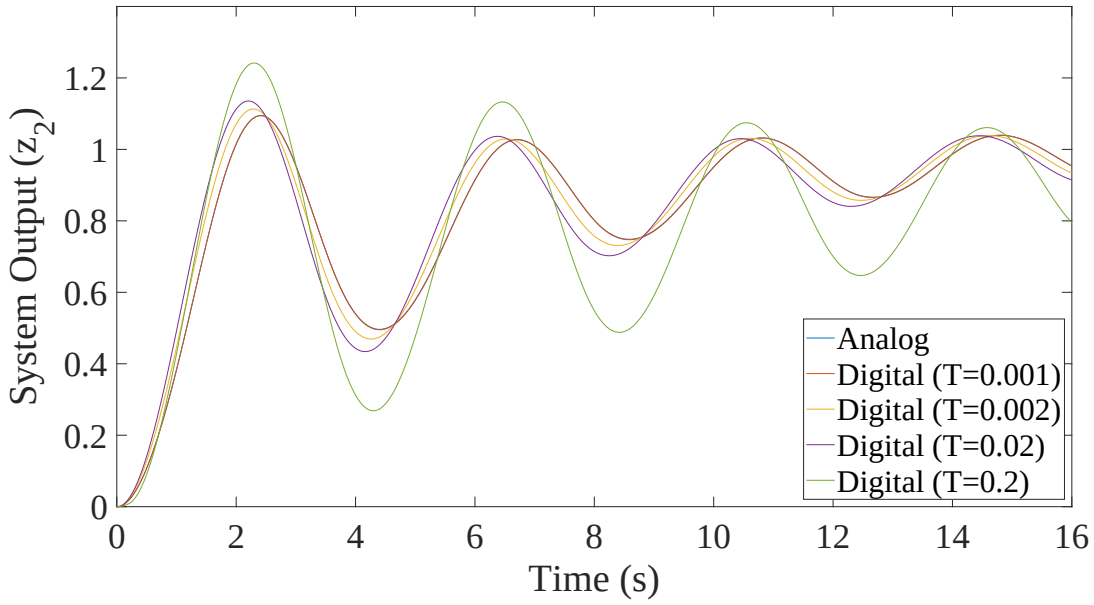

Figure 2.12: System's output z_2 for the analog and digital controllers with different sampling times in the closed-loop test system (second scenario)

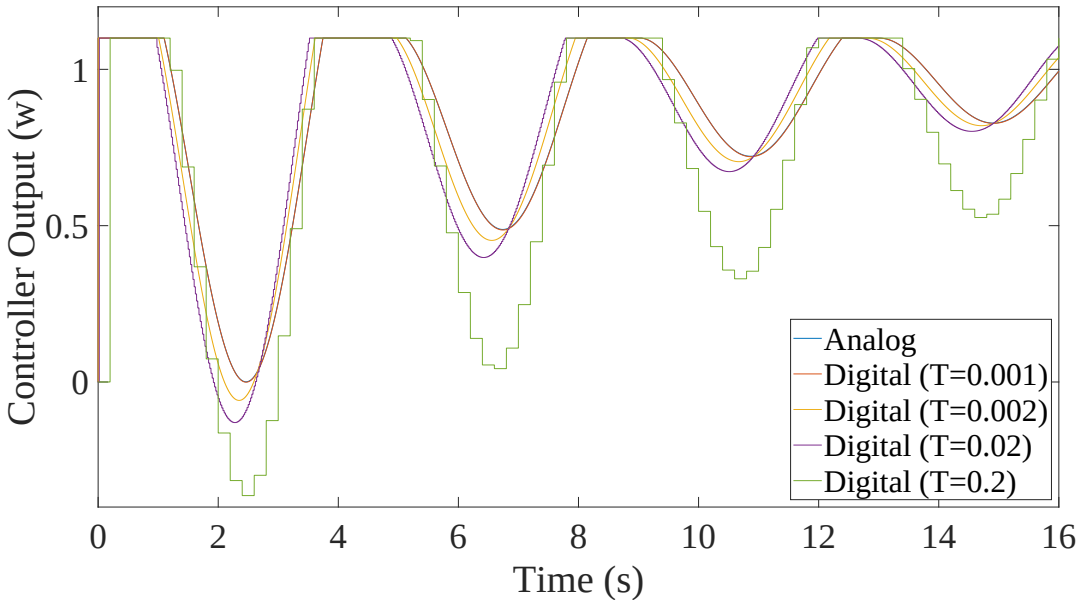

Figure 2.13: Controller's output w for the analog and digital controllers with different sampling times in the closed-loop test system (second scenario)

The same simulation for a variation of different numbers of quantization bits is repeated while keeping the sampling rate of the controller fixed at 1 ms. The system's response is illustrated in Fig. 2.14 and the controllers' output in Fig. 2.15, respectively. Furthermore, a zoomed version of parts of the same figures is provided in Fig. 2.16 and Fig. 2.17, respectively. It can be seen that the number of quantization bits also heavily impacts the accuracy. This emphasizes another reason that estimating a digital controller with a continuous equivalent may produce different results than the real controller may behave in reality.

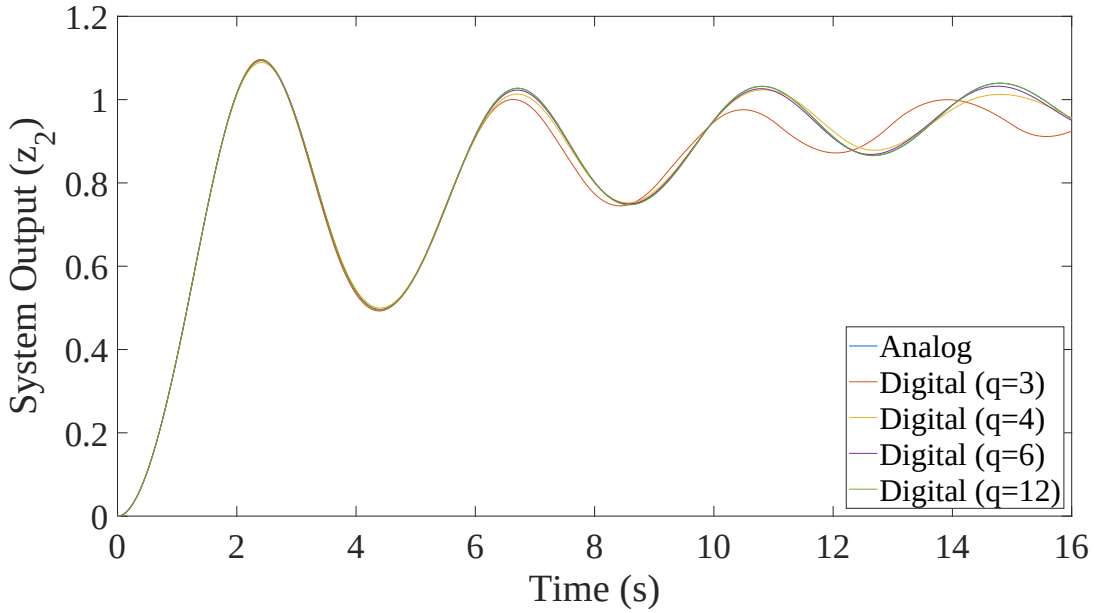

Figure 2.14: System's output z_2 for the analog and digital controllers with different numbers of quantization bits in the closed-loop test system (second scenario)

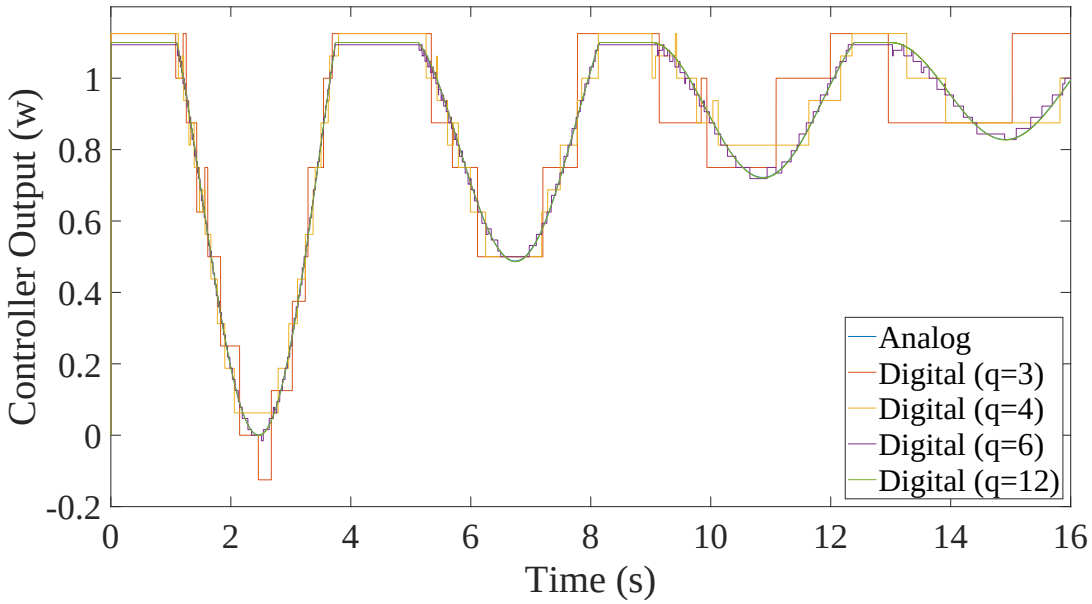

Figure 2.15: System's output z_2 for the analog and digital controllers with different numbers of quantization bits in the closed-loop test system (second scenario)

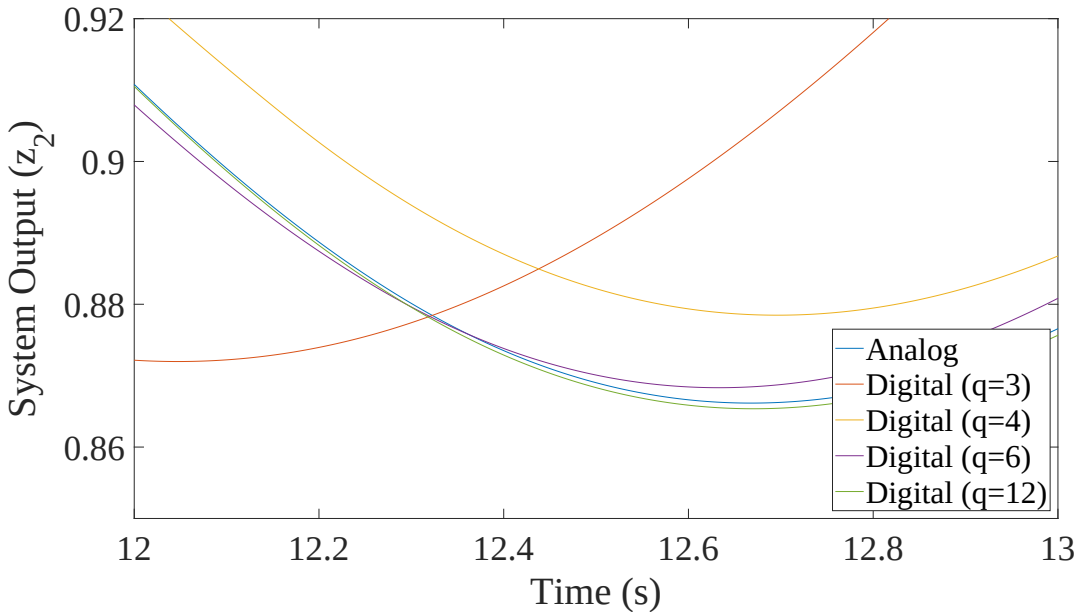

Figure 2.16: System's output z_2 for the analog and digital controllers with different numbers of quantization bits between 12 and 13 s in the closed-loop test system (second scenario)

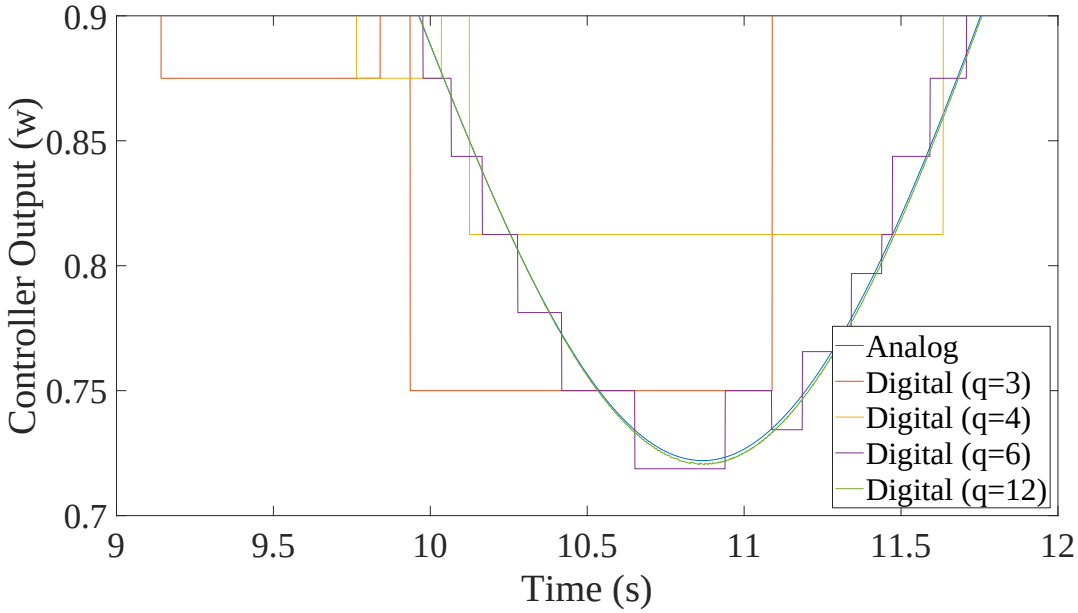

Figure 2.17: System's output z_2 for the analog and digital controllers with different numbers of quantization bits between 9 and 12 s in the closed-loop test system (second scenario)

2.3.4 Three-bus test system model

In this case study, a 3-bus network as depicted in Fig. 2.18 is considered. It comprises a synchronous generator (SG), a photovoltaic (PV) source, and a load, interconnected by two transmission lines. The generator dynamics are governed by an automatic voltage regulator (AVR, IEEE DC2A) [2] and a governor (type HYGOV) [14], operating with respective sampling times T_G and T_E. Typical sampling times for digital AVRs range from 5 to 50 ms, while governors generally operate with sampling times between 5 and 250 ms, depending on the specific controller and system characteristics. The equations describing the synchronous generator, the controllers' block diagrams, and their parameters are defined in Appendix I.

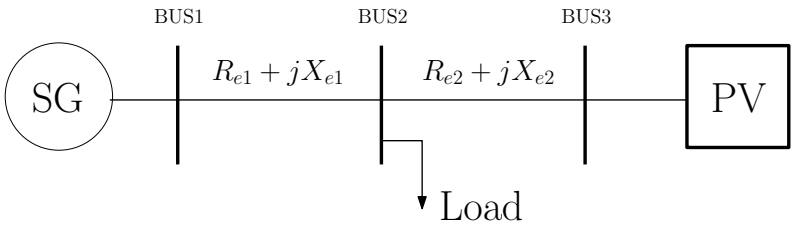

Figure 2.18: The schematic of the 3-bus network

The system is simulated for 16 seconds, and a step change in the PV's active power production from 3 to 2 MW is applied between seconds 3 and 6. The system is first simulated using the analog controllers, with both a fixed time-step ($h = 1$ ms) and a variable time-step ($h_{min} = 1$ ms and $h_{max} = 500$ ms). Subsequently, the simulations are repeated using the digital controllers replacing their analog equivalents.

Voltage of Bus 2, the generator speed deviation, the governor, and the exciter output are shown in Figs. 2.19-2.22. The simulations with the analog controller variants give an identical response, both for the variable time-step and the constant time-step versions. The hybrid simulation deviates from the

analog ones, with the ZOH shown more prominent in Fig. 2.22 since the effect of the digital controllers' sampling rate is considered.

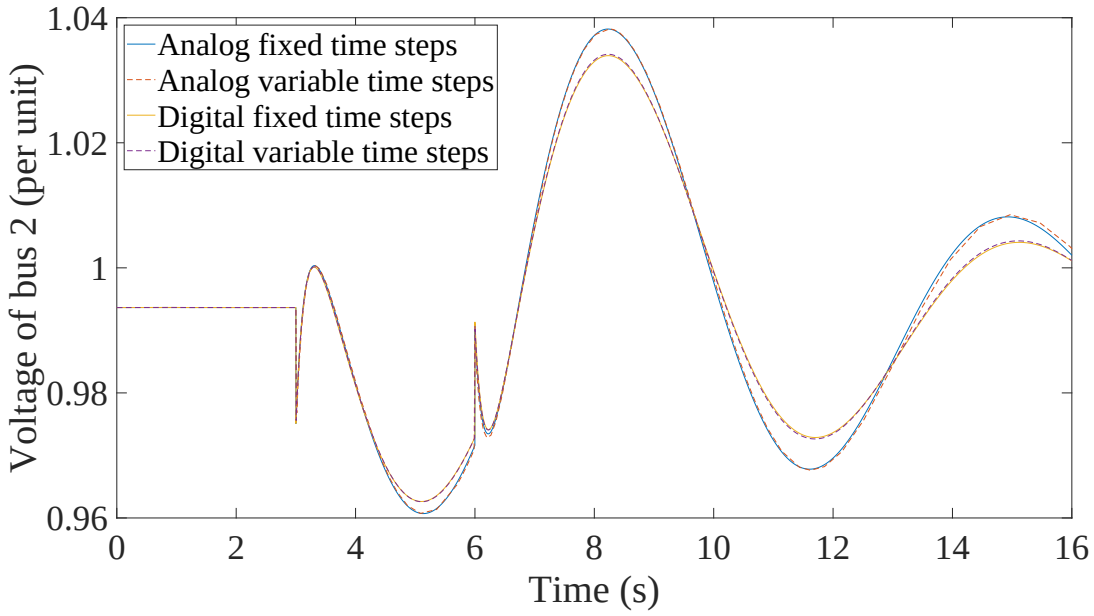

Figure 2.19: Simulation results for the voltage of the load on Bus 2 of the 3-bus test system

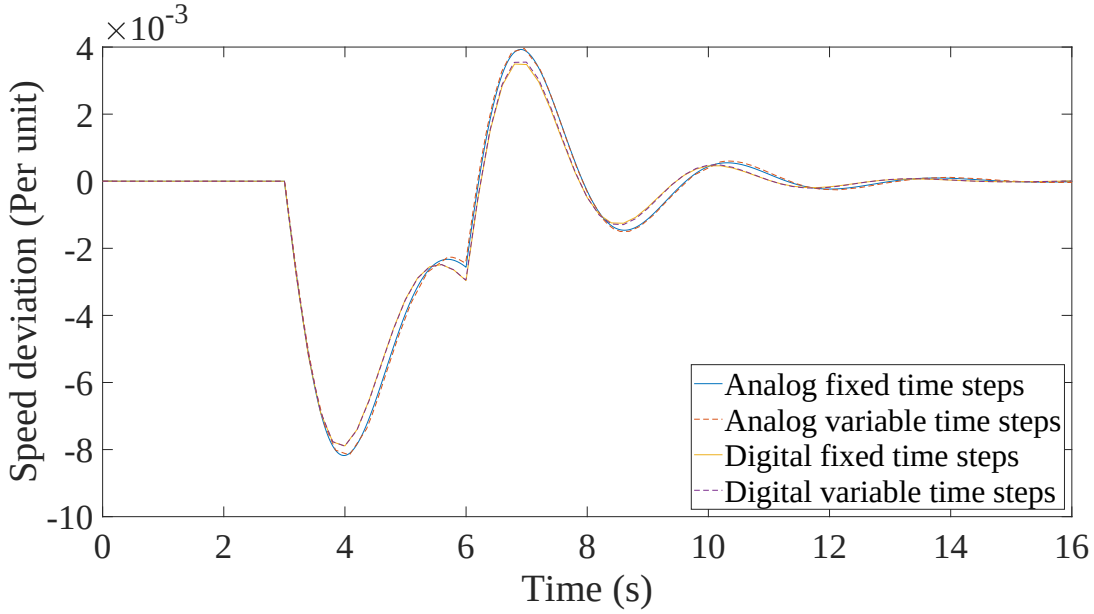

Figure 2.20: Simulation results for the speed deviation of the generator of the 3-bus test system

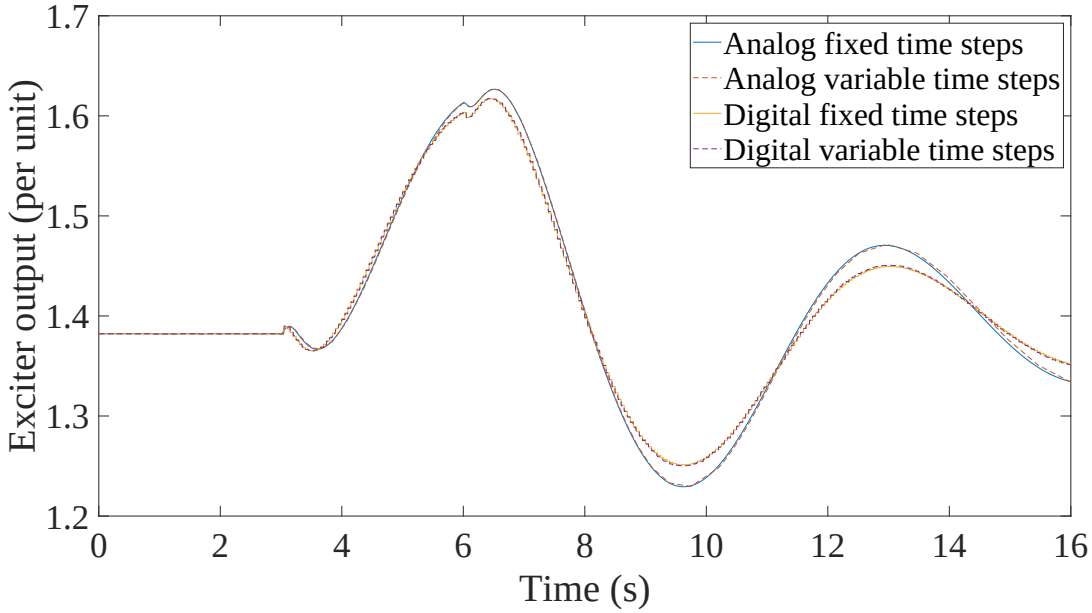

Figure 2.21: Simulation results for the generator's exciter output of the 3-bus test system

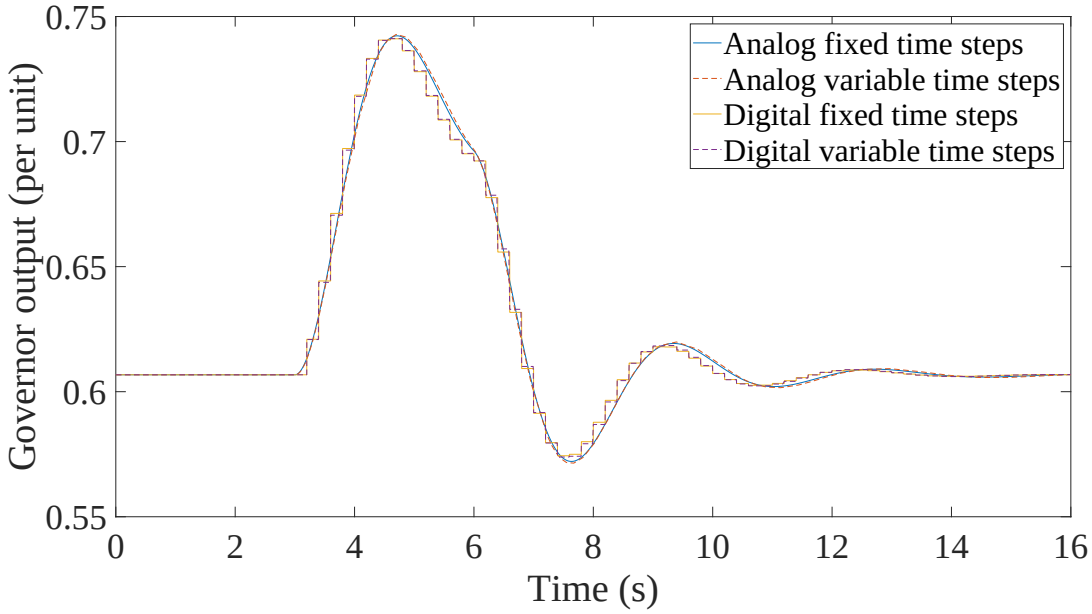

Figure 2.22: Simulation results for the generator's governor output of the 3-bus test system

The performance results of the simulations are listed in Table 2.1. It can be seen that the performance of the analog controller simulation with the variable time step integration is better than the digital controllers due to the larger time steps taken. This is one of the main reasons that analog controllers are preferred in dynamic simulations. Nevertheless, it can be seen that using an analog controller instead of digital ones brings the wrong result in case the real controller is digital. So, the decisions made over the simulation errors may put the system at risk.

Table 2.1: Performance of simulations for the analog and digital controllers with fixed and variable time steps

Method	Nb. Newton iterations	Runtime (s)
Analog cont. - fixed time steps	31580	16.94
Analog cont. - variable time steps	436	0.34
Digital cont. - fixed time steps	31717	13.77
Digital cont. - variable time steps	3422	1.69

2.4 Summary and discussion

In this section, the most important observations are summarized, and associated verdicts are discussed.

2.4.1 Accuracy

Based on the results, it can be observed that in most cases the accuracy between simulating an analog or a digital controller (having the digital controller as the benchmark) is impacted severely by the sampling time. When the sampling time is smaller than the time step ($T < h$) the trajectories are identical between the two, while with a sampling time larger than the time step ($T > h$) there is a notable discrepancy. This is due to the fact that the controller modeled as continuous with a DAE is discretized by the integration method and updated according to the time step, while the digital controller has a pre-defined sampling time and is interfaced to the rest of the system with a DAC.

In addition, the number of quantization bits can heavily impact the controller; consequently, the response of the system under control. Therefore, this calls for extra caution while estimating digital controllers with continuous counterparts.

Furthermore, although using a variable step size method helps with the performance of systems controlled by continuous controllers, it results in slightly lower accuracy (due to the allowance of higher time steps). However, it doesn't affect the digital controllers' accuracy significantly, since the step size is limited to the biggest difference between the sampling times of the digital controllers.

2.4.2 Performance

In the case of using a variable step size solver, the continuous models outperform the digital ones since no step size reduction is imposed. When a digital model of the controller is used, the time step is always limited to the biggest difference between the digital controller sampling times (time events).

On the other hand, when the continuous controller model is used, there is a need for detecting and locating the state events introduced due to the controller limits. When the digital controller model is used, this is not necessary due to the delays introduced by the ZOH. Thus, even if higher time-steps can be implemented with the continuous models, a zero-crossing detection algorithm needs to be implemented.

2.4.3 Modeling

To model the controllers as continuous devices requires their equations to be accessible and differentiable. This means that heuristic (and other non-equation-based) controllers can not be included and numerically solved. However, in the case of digital controllers, due to the time-delays introduced by the ZOH and since their equations don't appear in the solver, they can be treated as a black box with an input and output that is called before each time step.

Finally, it is shown that although chattering problems still exist for the digital controller models, deadlocks cannot happen. Therefore, in the case of deadlock problems, the performance of the digital controller is much better in the deadlock zone as the solver doesn't stagnate.

3 Review of Discrete Events Handling Methods and Challenges

To perform dynamic simulations, the equations of the continuous components of the power system are typically described using a large set of non-linear, hybrid DAEs, and the discrete-in-nature digital components must be modeled with difference equations. To solve such a set of equations, integration methods come in handy. However, they become slow if there are numerous discontinuities over the simulation time horizon since the solver is forced to constantly reduce the time steps to land on discontinuity times. A classic example of such a case is the simulation of systems with digital controllers, as was discussed in the previous section. The process required for handling discontinuities during the simulation and the associated challenges arising are discussed in the following.

3.1 Discrete events handling

Integration of the continuous DAEs ignoring a discontinuity causes the loss of accuracy and a huge value of the local error estimate. The reason is that the approximation of the solution computed in the presence of discontinuity is computed using an expired set of equations or solution flow, leading to a large value of the error estimate. Consequently, the step size will be rejected and reduced drastically [23], and that time interval integration will be repeated using the new smaller step size. This process may repeat several times. Thus, there is a need for an appropriate routine to detect and locate the discontinuity effectively and adjust the discretization step size to reach the discontinuity location without passing it too far and prevent the algorithm from hunting around the discontinuity. Finally, the discontinuity that is detected and located should be treated effectively. The basic steps of an algorithm for handling such discontinuities are [23]:

1. **Detection**: The possible discontinuity should be detected effectively. The procedure used for the detection should be computationally inexpensive since it is going to be continuously activated throughout the simulation.
2. **Location**: After the discontinuity is detected, its location should be computed accurately with an algorithm tailored to the numerical integration method that is being used.
3. **Treatment**: After the location of the discontinuity is identified, a methodology to bridge the solution flow of the equations sets of the system before and after the discontinuity is required. The selected treatment must ensure the accuracy of the solution over the discontinuity.

Improper handling of discontinuities can lead to successive step failures and loss of accuracy [23]. To avoid this problem, based on the assumption that the state variables and their derivatives are continuous, a solution called Discontinuity Locking [24] is widely used in different numerical simulations [1, 4, 25]. Basically, this method locks the equations of the system during a time interval of integration and prevents them from updating in case of an event, i.e. the detection stage happens only at mesh points of the integration process and after the discontinuity location is identified using a proper algorithm, the equation set of the system will be updated. Finally, the treatment happens to bridge the solution of two equation

sets of the system.

Overall, discrete events in DAEs can be categorized as state events and time events [19]. The events happening at specific points in time, that are known in advance, are called time events while the events triggered by rules applied to the state variable values of the system are called state events. In other words, while the state events happen when a guard function is satisfied, for instance, a state variable reaches a specific value, time events happen when an independent variable reaches a certain amount [19]. Generally, a time event occurs at a specified value of time (independent variable) and state events happen by reaching a certain state. The operation of digital controllers with specific sampling and action times is a classical example of time events. On the other hand, the operation of transformer tap changers due to a change in the system's voltage is considered a state event. Time events are involved only in the third step of the discontinuity handling process called the treatment stage since their occurrence can be predicted in advance.

Let's consider a hybrid power system modeled with DAEs in an explicit form as follows [26][1]:

$$\boldsymbol{\Gamma}(\boldsymbol{z})\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}, t) \tag{3.1}$$

$$\boldsymbol{g}(\boldsymbol{y}, \dot{\boldsymbol{y}}, \boldsymbol{z}(t_*^-), t) = \boldsymbol{z}(t_*^+) \tag{3.2}$$

where $\boldsymbol{\Gamma}$ is a diagonal matrix in which its elements determine if an equation is a differential either an algebraic, i.e. $\boldsymbol{\Gamma}_{ll} = 1$ if equation (1) is a differential equation and $\boldsymbol{\Gamma}_{ll} = 0$ if it is algebraic one. Furthermore, $\boldsymbol{y}$ is the state variable vector (both differential and algebraic), $\boldsymbol{z}$ is the vector of discrete variables that model the events, $\boldsymbol{f}$ is the differential equations, and t is the independent variable of time. A set of arbitrary discontinuity functions or as it is called in some references guard functions denoted by $\boldsymbol{g}$, determine the discrete states of the equation system [1]. The guard functions control the solution flows, i.e. when a guard function is activated (changes it sign) at time t_* the equation set of the system should be updated and the trajectory of the solution jumps to a different flow. In other words, the DAE (4.5) is a finite collection of continuous DAEs, one for each discrete variable change, $\boldsymbol{z}(t_*^-)$ to $\boldsymbol{z}(t_*^+)$ [26]. Besides, this system is subjected to a set of initial values which also determines the flow which the solution starts on. Finally, let the step size of the integration process will be denoted by $h_n = t_n - t_{n-1}$. A hybrid system containing two sets of equations is illustrated in Figure 3.1.

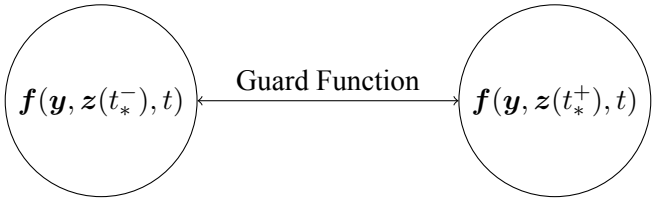

Figure 3.1: A generic hybrid system transition to a new state triggered by its guard function

Numerical integration methods for finding an approximation to the solution of a DAE are divided into two main categories of single-step (or Runge-Kutta methods) and multi-step methods [22]. The difference is that the multi-step methods use previous integration steps to approximate the solution. The

[1] In this report, matrices are denoted by bold capital letters and vectors are denoted by bold small letters.

most popular multi-step method is the Adams family methods while Backward Differentiation Formula (BDF) methods are suitable for stiff problems.

From another point of view, they can be divided into explicit and implicit methods. An implicit method has the unknown vector $\boldsymbol{y_n}$ (approximation of $\boldsymbol{y}$ at t_n) on both sides of the method's formulation. Thus, in the case of using an implicit method, a nonlinear system of algebraic equations must be solved for every time step [27]. It is noteworthy that implicit methods are more convenient for the integration of stiff problems such as power system DAEs.

3.2 Discontinuity detection

As discussed in the previous section, the presence of a discontinuity makes the integration algorithm reduce the step size due to the large value of the error estimate. This is the foundation of the discontinuity detection presented in [23]. Basically, it is suggested that if a step size is executed and a large value of error estimate is computed, consequently, the step size is rejected and the newly calculated step size is drastically smaller, a discontinuity is detected. Thus, the detection method is to check if the local error estimate is much greater than ϵ (which is determined by the user) or $h_{new} \ll h_{old}$.

Therefore, the time events could be located immediately, but locating the state events is cumbersome. For a simple example, a heating system in an office could be taken into consideration. This system is turned on manually every day at 8 a.m. (time events). After reaching an exact temperature, it is turned off automatically, and by decreasing the temperature to a certain point,t it will be turned on automatically (state events).

When a guard function changes its sign is referred to as *zero-crossing* [3]. In other words, according to the intermediate value theory, for a continuous guard function and a given interval $[t_a, t_b]$, at least one root must exist in this interval if the signs of $\boldsymbol{g}(t_a, \mathbf{y_a}, \mathbf{z_a})$ and $\boldsymbol{g}(t_b, \mathbf{y_b}, \mathbf{z_b})$ are different [5]. Thus, when a zero-crossing happens in a guard function or when its trajectory crosses the zero, a discontinuity has occurred, and specific measures, which will be discussed in the next section, need to be taken to locate it accurately. Zero-crossing is used widely in many algorithms of discontinuity handling for the detection stage. Since the sign check is not a heavy computational task (it only involves function evaluations), it usually takes place for every step. However, [28] suggests doing the sign check only when the step is rejected to reduce the computational cost.

3.3 Detection Complexity

As the system model becomes more complex and the number of discontinuities increases, some issues may arise in the detection phase, and a simple change of sign evaluation may not be sufficient [3, 29]. The simple zero-crossing detection algorithm has its drawbacks, and different research works have tried to address them. For instance, there might be more than one zero-crossing within the integration time interval studied, leading to the need to locate the earliest one for treatment. These multiple roots could be either an odd or even number. When there are multiple roots, the simple check of the signs is not adequate, and more elaborate methods must be used. The issues that may arise and their addressing solutions in recent works in the literature are summarized in the following.

3.3.1 Even roots

It can be seen that a simple zero-crossing evaluation using a sign change between two points of a time interval cannot detect an even number of roots since in this case there is no sign change. An example of a guard function crossing zero twice is illustrated in Figure 3.2.

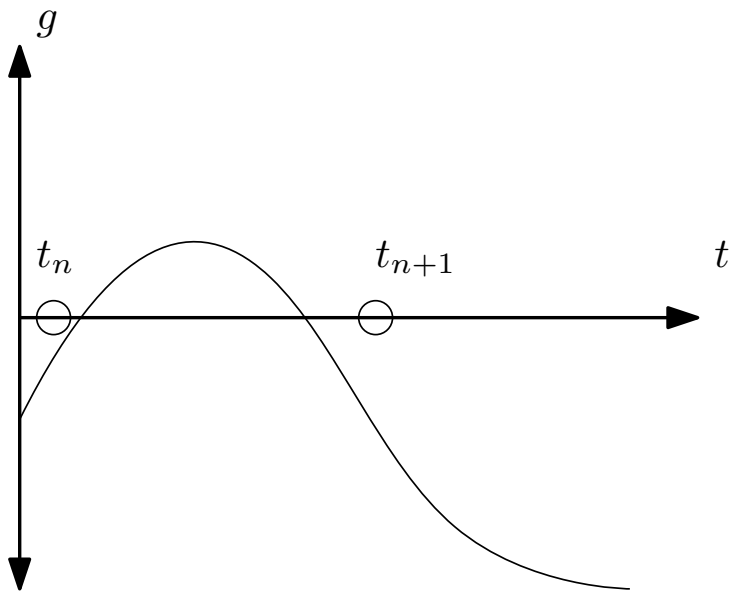

Figure 3.2: Missed even roots happening in between two time steps taken t_n and t_{n+1} [3]

Thus, zero-crossing needs to be completed or replaced by other methods that can detect the complexity of even roots. To achieve this, [1] proposes to detect the discontinuity using the sign change both in the guard function and its derivative. The sign change in the derivative of the guard function shows passing through zero in both directions in the time interval, thus an even number of roots. In other words, the situation of even roots can be distinguished using this technique. Two following auxiliary variables show the sign change in guard function and its derivative, respectively:

$$s_1 = g_t \cdot g_{t+h} \tag{3.3}$$

$$s_2 = \dot{g}_t \cdot \dot{g}_{t+h} \tag{3.4}$$

where g_t and $\dot{g}_t$ denote the evaluation of the guard function and its derivative at time t, respectively. The proposed algorithm is summarized in Table 3.1.

Table 3.1: States of the method used in [1] for detecting the discontinuity and actions taken

condition	s_1	s_2	number of events	Action
(a)	$+$	$+$	0	The integration process continues
(b)	$+$	$-$	2	The step size is reduced, and the integration is repeated for the new step size until it gets to a condition except (b)
(c)	$-$	$-$	1	The integration process proceeds to the *Location* stage

Another method proposed in literature capable of detecting even roots is based on the Sturm sequence algorithm [30] and is presented in [31]. The Sturm sequence is an algorithm that can be used to find

the number of real and complex roots of a function. The proposed method contains two modules for detecting and locating the discontinuities, respectively. The first module takes a time interval and the guard functions as the inputs and returns the number of roots in that specific time interval. Then, the second module computes the location of the first root. Using this method, it can be assured that all of the discontinuities are detected efficiently. This detection algorithm is also used in [6].

Another solution proposed in [4] would be including the dynamics of the guard functions in the system modeling. In this method, by including the guard function in the system of equations, the step size would be adjusted by the integration method automatically when too large errors stemming from even roots are produced [3, 5]. The authors of [5] propose a detection method based on including the dynamics of the guard functions and their derivatives in the detection phase as well. According to this method after a sign check, three different situations may occur:

1. All of the discontinuity functions change the sign. In this case, the earliest root is found using the Illinois algorithm [32, 33].
2. None of the discontinuity functions change sign. In this case, the signs of the derivatives of discontinuity functions are checked to see if extra evaluation of the guard function in the middle of the time step is needed.
3. Some of the discontinuity functions change the sign and some of them don't. In this case, the earliest root t^* is found using using Illinois algorithm [32, 33]. Then, a reduced time step up to the t^* is searched for any other possible roots for the discontinuity functions without sign change but have sign change in their derivatives.

Finally, Zero-crossing refinement is another method to deal with even roots, proposed in [3, 5]. In this method, the time interval is divided into several smaller ones and the sign check happens at the end of smaller time intervals. Although this method is computationally more efficient compared to other methods reviewed so far, it just reduces the chance of even roots happening and doesn't guarantee the detection of even roots.

3.3.2 Masked Even Roots

This situation happens in systems with more than one guard function and one of the guard functions prevents the others' discontinuities from being detected. Let's assume a system containing two guard functions that for a specific time step, one of them crosses the zero once and the the other one twice. In this case, the guard function with the even number of roots will be masked by the guard function with the odd number of roots and will not be considered at all. The possible solution would be again using zero-crossing refinement [3] to reduce the chance of it happening.

3.3.3 Double Detection

Double detection happens when the value of the guard function becomes exactly zero at the point of discontinuity [3]. In this case, two events might be detected, once crossing to zero and once crossing away zero, while only one zero-crossing happens. The solution proposed by [3] is to consider only one of them by distinguishing them using OR operators as is shown in the following equation for a case of

zero-crossing from negative values to positive.

$$\delta = \delta_{-to+} = \delta_{-to+}|\delta_{-to0}|\delta_{0to+} \tag{3.5}$$

where δ denotes an event and its subscript shows the crossing direction, and | is the OR operator. For instance, δ_{-to+} means a discontinuity detected when the guard function's value changes from a negative value to a positive value.

3.4 Discontinuity location

When the existence of a discontinuity is detected using one of the methods described above, the exact location of the discontinuity needs to be computed accurately. In this section, different methods used to locate discontinuities and tackle the complexities that may arise are discussed.

3.4.1 Interpolation-based Methods

A method for discontinuity location based on interpolation is proposed in [19, 34]. The authors suggest using y and $\dot{y}$ at the mesh points t_n and t_{n+1} to construct the Hermit interpolation polynomial approximating the solution at other points of the time interval. This inverse linear interpolation is then used to approximate the event time (where the guard functions intersect the solution). Direct interpolation has been used to approximate the solution at the time of event calculated in the previous stage. A cycle of these two stages, i.e. using inverse interpolation to approximate the event's location and using direct interpolation to approximate the solution at the event's location, can be repeated until the time of discontinuity is determined with a user-specified threshold. It should be noted that this location method is used in combination with a Runge-Kutta integration method which doesn't provide an interpolation intrinsically.

Another interpolation-based method for discontinuity location is presented in [35]. In case of an event, a reduced step size obtained from a linear interpolation calculated using values of guard function at the mesh points will be used for the integration. This procedure is repeated until the time of the event is determined with a prescribed accuracy. Furthermore, the authors of [25] suggest using a search method to locate the discontinuity. However, the search procedure requires the values of the unknowns to be able to evaluate the guard functions at points other than mesh points. Thus, using interpolation formulas to access information for all time interval points other than mesh points is proposed.

In the papers mentioned above, the interpolation formula is used to approximate the variables at points other than mesh points. However, the algorithm in [4] constructs interpolation polynomials for guard functions and uses them for both detection and location of the discontinuity. As discussed in the previous section, they benefit from a combination of the exclusion test and Newton/bisection method to detect the event. However, for the location stage, the system of equations below at the time of the discontinuity t_* must be solved.

$$\boldsymbol{\Gamma}\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}, t^*) \tag{3.6}$$

$$\boldsymbol{g}^*(\boldsymbol{y}, \dot{\boldsymbol{y}}, \boldsymbol{z}, t^*) = \pm\epsilon_g \tag{3.7}$$

where the parameter ϵ_g is a positive bounded value called discontinuity tolerance. This system of equa-

tions is solved using the Newton iteration method and since a good initial guess is already available for the unknowns from the detection stage, it usually needs only a few iterations to converge.

3.4.2 Search Methods

There are different search methods to find the roots of a function in a given time interval [32]. These methods have been widely discussed and used to locate the event in different research works. Some of the search methods are summarized in the following:

1. Newton-based method: Newton-Raphson method (also known as Newton's method) uses the tangent line of $\boldsymbol{g}$ at the selected point (usually a guess) to calculate the zero-crossing. The equation below shows the formula of this method, where x denotes a point on the time interval and its subscript n shows the point number.

$$x_{n+1} = x_n - \frac{\boldsymbol{g}(x_n)}{\dot{\boldsymbol{g}}(x_n)} \tag{3.8}$$

   This is one of the most used methods for root finding in discontinuous systems. It is used for locating the zero-crossing in [4, 6] in combination with the bisection method. Its application can be seen in other research works such as [36] as well.

2. Secant-based method: The secant method is similar to Newton's method. However, it uses the secant line instead of the tangent line to find the zero-crossing. The equation below is the formula for this method.

$$x_{i+1} = x_i - \frac{\boldsymbol{g}(x_i)}{\frac{\boldsymbol{g}(x_i) - \boldsymbol{g}(x_{i-1})}{x_i - x_{i-1}}} \tag{3.9}$$

   This method's application in zero-crossing location is shown in [36]. Also, a combination of the secant and bisection methods are applied in [31]. The package IDA of SUNDIALS which is one of the most-used packages for solving DAEs, also uses this method for locating the root [37, 38].

3. Bisection-based method: This method uses only the signs of the guard function values at the start and endpoint of the function. If their multiplication is negative it halves the time interval and checks the sign of the function value at the midpoint of the time interval. Then, it selects one of the two halves of the main interval to repeat the same procedure based on the dissimilarity of the signs of the new interval [32]. This process continues until the bracketed interval containing the root is smaller than a specified threshold by the user. This method is used for a zero-crossing location in many research works such as [4, 23, 25, 31].

4. Regula-Falsi: Regula-Falsi or false position is another method that can be used to find the root of a function in an interval [39]. It uses the equation below to calculate the new point.

$$x_{i+1} = \frac{(x_{i-1} \cdot \boldsymbol{g}(x_i)) - (x_i \cdot \boldsymbol{g}(x_{i-1}))}{\boldsymbol{g}(x_i) - \boldsymbol{g}(x_{i-1})} \tag{3.10}$$

5. Illinois method: This algorithm is similar to the bisection method and has been used in [5, 27] for zero-crossing location. However, instead of halving the time interval, it performs the bisection method on the function values and halves the interval between values of the function to find the

new point.

6. Inverse Quadratic Interpolation: Inverse Quadratic Interpolation (IQI) needs three points on g to calculate the new point (fourth point). This method fits a sideways parabola to the guard function in the selected interval where a root is detected and where the parabola crosses zero is the new point [32]. This process continues until the difference between the old point and the new point is less than a user-specified threshold.
7. Combination of methods: Some algorithms use a combination of the above methods. For instance, MATLAB's `fzero` function can be used to find the roots using a combination of the secant method, bisection method, and inverse quadratic interpolation [32]. This function enjoys the reliability of the bisection method and the fast speed of secant or IQI methods.

3.5 Location Complexity

Although the search methods detailed above are effective in most cases, there are special cases in which these methods may face a challenge to locate the event effectively. In the following, these special cases and solutions addressed in the literature are discussed.

3.5.1 Discontinuity Sticking

Generally, when a discontinuity is detected and located properly, the equation set of the system is updated and the integration process resumes. In this situation, the new initial values required for the integration process are calculated, and the process restarts. However, in some special cases, these new initial values used to restart the integration process may lead the guard function trajectory to start from a situation where the same discontinuity handled earlier is triggered again. This is depicted in Figure 3.3 and is referred to as "discontinuity sticking". It could cause simulation inefficiency and failure [4].

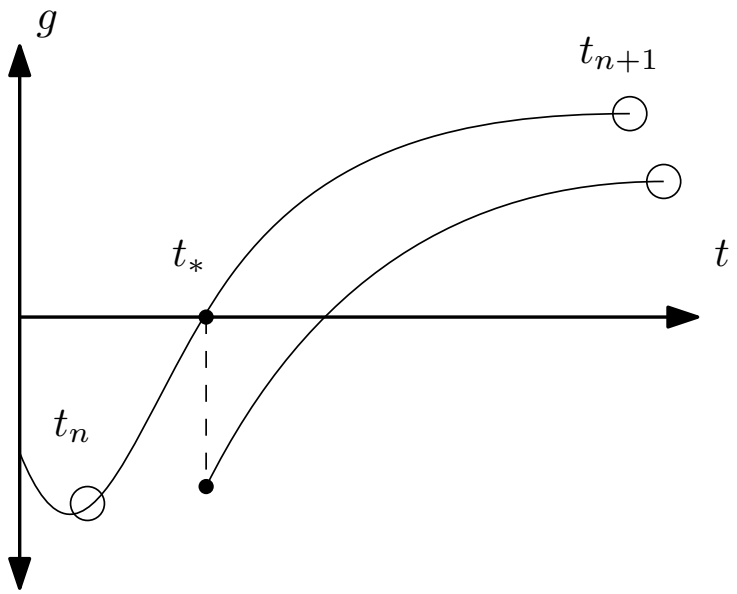

Figure 3.3: Discontinuity sticking problem [4], forcing the same discontinuity to be detected again

In the above figure, circles show the time steps, t_* is the event's time calculated by the zero-crossing

of the guard function.

One way to address this problem is presented in [4]. Instead of the guard function itself, a polynomial interpolation of the guard function is used for both detection and location of the discontinuity which is discussed before using equations (3.6) and (3.7). The parameter ϵ_g, which is called the discontinuity tolerance, can be calculated using the guard function's slope and the state event tolerance δ, which is bounded between ϵ_g^{min} and ϵ_g^{max}. The maximum value controls the accuracy of the calculations while the minimum value prevents from discontinuity sticking and ensures that after restarting the integration process, the same discontinuity won't happen again.

Another solution to the discontinuity sticking problem is presented in [5]. The authors propose that when a discontinuity is successfully handled, the updated equation set of the system is locked for the time interval $t_* + h_{min}$ and the event detection algorithm is deactivated up to the end of that time step. This solution is illustrated in Figure 3.4.

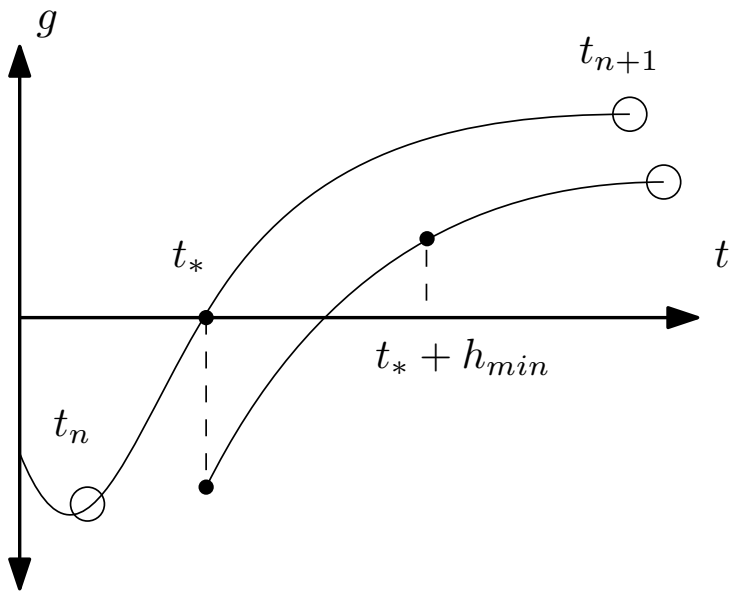

Figure 3.4: Solution to the Discontinuity sticking problem presented in [5], locking the discontinuity detection mechanism until $t_* + h_{min}$

3.5.2 Cycling

Sometimes, the simulation may get stuck jumping between two or more discontinuity states in an endless closed loop. In this case, a modeling modification, such as adding another state between the cycling states or modifying the guard functions, is proposed in [40]. Alternatively, another solution is ignoring the discontinuities in the current step after reaching a certain number of iterations or reducing step size (if possible) [40] to break the cycling.

3.5.3 Chattering

This complexity happens when both vector fields of the solutions on both sides of the discontinuity are toward the discontinuity. In this case, the next step causes a discontinuity detection and mode switch. Since, also in this side of the discontinuity surface, the solution flow is toward the discontinuity, the

mode will switch again and this fast cycling behavior referred to as chattering will continue infinitely. According to [41, 42], basically 3 situations may happen at the surface of the discontinuity:

1. The vector field of the solution is toward the discontinuity surface from one side of it and leaves it from the other side
2. The vector field of the solution is pointing away from the discontinuity surface
3. The vector field of the solution is toward the discontinuity surface on both sides of the discontinuity surface

In the third situation, the simulation starts switching modes while slowly crawling toward the discontinuity surface. The difference between chattering and cycling is that chattering happens between two flows repeatedly while cycling may happen with more than two flows. A proposed method to address this problem [42] called sliding mode simulation uses the bisection method as an appropriate method for finding the discontinuity surface. After a chattering situation is detected by switching between two modes, a binary search predetermines the number of steps required for the simulation to exit chattering. Then, the velocity of the sliding motion toward the discontinuity surface is calculated (based on the Filippov theory) and implemented on the solution trajectory. In other words, when a chattering situation happens, the fast switching will be removed while the slow crawling on the discontinuity surface is taken into account.

3.5.4 Zeno Behaviour

The discrete events happening while getting closer and closer in time after every time step, toward a limit point is called Zeno behavior [3]. In other words, the distance between events decreases until a limit point. The difference between this complexity mode with chattering is that the speed of fast switching between two modes increases over time in the case of the Zeno, however, the effect of both pathological behaviors is the same, halting the simulation [3]. A famous example of this situation is the bouncing ball problem [43–45], and different solutions are proposed to deal with this problem. As the ball hits the ground and loses some energy every time, the time distance between events decreases until a threshold beyond that is called the Zeno neighborhood.

Reference [45] considers systems with Zeno points incomplete and defines actions to be taken in order to make the system complete i.e. the system will be able to be integrated beyond the Zeno point. Also, regularizing the system with the Zeno point is proposed in [44]. Furthermore, a simple solution called ”dynamic zero crossing location” is discussed in [3], which defines criteria to detect the Zeno behavior to temporarily deactivate the zero-crossing location in order to let the simulation continue beyond the Zeno point.

3.5.4.1 Unilateral events

Let’s consider the situation that the right-hand side of the differential function is not defined beyond the guard function. In such a case, search methods are not useful anymore to locate the discontinuity. Such events, which are called unilateral events [46], need a different location method that doesn’t allow the numerical simulation to pass the discontinuity.

A solution to this kind of problem is addressed in [29] which resembles the simulated system to a control system. Based on this analogy, the integration step size is taken as input, the guard function as the output, and the equation of the numerical method as the system dynamics. The proposed algorithm selects the feedback law as a step size control to approach the discontinuity surface asymptotically without passing it. As the simulation continues the chance of the event being detected increases and since the simulation is not going to pass the event so there are no worries in that area. Furthermore, a similar solution is presented in [46]. In this method, an extrapolation polynomial is used for the detection and location of the roots. The proposed algorithm can be summarized as follows:

1. An extrapolation polynomial is constructed with the same order of accuracy as the integration method used. This polynomial extrapolates the guard function over $h_{n+1} \in [0, \infty]$.
2. if the polynomial has any roots detected the smallest root is determined and denoted by h^*, otherwise $h^* = \infty$.
3. The step size calculated by the traditional integration algorithm based on the error estimate denoted by h^{err} is calculated.
4. the integration continues for the step size equal to the minimum of the step sizes calculated in the previous stages $h_{n+1} = min(h^*, h^{err})$.

3.5.5 Other methods

In this section, other location methods that are not mentioned so far, are briefly introduced and discussed.

Reference [28] proposes to take the step-size (h_n) as a variable in the Runge-Kutta formulation appended by the guard function when an event is detected and calculate it in a way that $t_n + h_n$ exactly lands on the discontinuity. The authors propose simplified Newton iterations to solve this system effectively. A similar approach, called α-method, is elaborated in [47] for a fourth-order Runge-Kutta method. They define α as a variable which is the portion of the time interval where the discontinuity is located. When the discontinuity is detected (by a simple sign check) the parameter h is replaced by αh. Then, two methods are proposed to calculate α. The first method is to calculate α by recalculating the coefficients of the Runge-Kutta method while the second method suggests calculating alpha using a Newton iteration method. Finally, it is concluded that the second method is more accurate.

Sturm sequence is used for the detection in [6, 31] as it is mentioned before. Using this algorithm, the number of roots in the specified interval is specified. For the location part, [31] exploits a standard bisection or secant algorithm to find the first root of a list of roots detected by the Sturm sequence while this is done in [6] by reusing bisecting and Sturm sequence to find the first root's location.

A method that uses the order of the discontinuity to obtain a more efficient process of discontinuity location is presented in [23]. According to that research work, the discontinuity has the order of q if f has continuous partial derivatives through order $q-2$ and there is a finite number of discontinuities in at least one of the partial derivatives of f of order $q-1$. In other words, the order of the derivative of the state variable that has a discontinuity is taken into account in the discontinuity location process. In case the order of the discontinuity equals one (worst case scenario) the discontinuity location would be a simple bisection method [32]. However, for higher discontinuity orders (especially order 2 discontinuity) a faster location method is proposed. For discontinuities of order bigger than 2 however, the calculations of the

intersection especially for a parabola or higher-order curves would be quite cumbersome and therefore the proposed method is working efficiently up to second-order discontinuity.

3.6 Treatment

When a discontinuity is successfully detected and located, the time step will be reduced, and the equation set of the system will be updated (if needed) to resume the simulation from a new point. In this case, there is a need for new initial values for proceeding with the simulation, but the values available are related to the previous equation set of the system. Therefore, a special treatment is required to calculate the new initial values for the updated equation set and bridge the solution trajectory. However, complexities may be considered during this stage that are reviewed in the next section.

It should be noted that the important premise for the following discussion is that the time step is reduced after the detection and location stages to land on the event. In other words, the treatment process can be divided into two stages of time step adjustment and the treatment approach itself. The alternative methods to the time step reduction method will be discussed in the next chapter.

3.6.1 Treatment complexity

After reducing the time step and landing on the event, a normal treatment is to use an explicit method, such as forward Euler for the first step size (or at least for the predictor part of the method if a predictor-corrector method is being used), since this method doesn't require any value from previous steps. Then, for the rest of the step sizes (or the corrector), a Backward Euler or Trapezoidal method could be useful. From this point on, the order of the method can be gradually increased in the case of using a variable-order method. For instance, a treatment with a similar logic is incorporated in the IDA package, which is a DAE solver [37]. It has a specific function for calculating initial values, which uses Newton iteration combined with a line-search algorithm, and another function that uses the calculated initial values for restarting the simulation. For the first few steps, the algorithm tries to double the step sizes and increase the order until it reaches the maximum order or the local error test fails. Although this procedure is usually the default treatment used to restart the simulation in the presence of a discontinuity [23], often some improvements are suggested, which are discussed in the following.

Reference [23] states that for discontinuities of order $q \geq 3$ the simulation should be restarted with a method of order no more than $q - 1$ to keep the local error limited to the method's order. However, realizing the order of the discontinuity at least for $q \geq 3$ is quite cumbersome, i.e. it needs a complicated algorithm, many rejected time steps, and many extra calculations.

A different improvement is proposed in [6]. In that research work, a pair of fifth and fourth-order Runge-Kutta methods is used for integration and error control. Also, the fourth-order continuous extension is used for the interpolation polynomials. However, standard interpolation polynomials derived from numerical integration often have discontinuities at mesh points. In other words, Inaccuracy between the predicted value of a guard function using the integrator polynomials g_{n+1} and the exact value $g(t_{n+1})$ defined with e_g produces discontinuity. Therefore, a blending polynomial is proposed to be added to the interpolation polynomial in order to fill the gap between the two values of the guard function and make it continuous. This polynomial could also prevent failure in root detection, if there is an event in that gap,

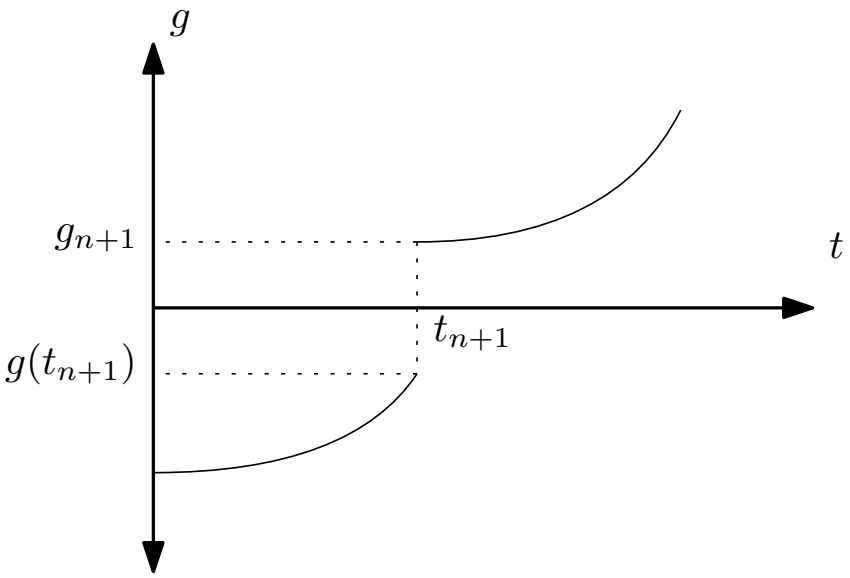

Figure 3.5: Hidden root located at the transition [6]

as it is illustrated in Figure 3.5. The suggested blending polynomial is as follows:

$$P_\sigma = (2\sigma^3 - 3\sigma^2)e_g \tag{3.11}$$

where σ is the interpolation polynomial parameter. This augmented polynomial has the following features:

$$P_\sigma(0) = 0, \quad \dot{P}_\sigma(0) = 0, \quad P_\sigma(1) = -e_g, \quad \dot{P}_\sigma(1) = 0 \tag{3.12}$$

A different method using Filippov theory [48] is discussed in [49]. A discontinuity divides the plane of the solution trajectories of an ordinary differential equation (ODE) into two regions:

$$R^- : \quad \dot{\boldsymbol{y}} = \boldsymbol{f}^-(\boldsymbol{y}) \quad \boldsymbol{g}(\boldsymbol{y}) < 0, \quad R^+ : \quad \dot{\boldsymbol{y}} = \boldsymbol{f}^+(\boldsymbol{y}) \quad \boldsymbol{g}(\boldsymbol{y}) < 0 \tag{3.13}$$

Based on the Filippov theory, the following equation can be used to categorize the situations that may happen when a solution trajectory intersects with the discontinuity surface.

$$w^\pm(\boldsymbol{y}) = \nabla \boldsymbol{g}(\boldsymbol{y}) \boldsymbol{f}^\pm(\boldsymbol{y}) \tag{3.14}$$

According to the above equation the point y on the discontinuity surface is a crossing point or sliding point if $w^-(\boldsymbol{y}) \cdot w^+(\boldsymbol{y}) > 0$ or $w^-(\boldsymbol{y}) \cdot w^+(\boldsymbol{y}) < 0$, respectively. A crossing intersection point means the solution trajectory crosses the discontinuity surface, while a sliding point makes the trajectory slide along the discontinuity surface. In case of reaching a sliding point, the vector field defines:

$$\dot{\boldsymbol{y}} = (1 - \alpha)\boldsymbol{f}^-(\boldsymbol{y}) + \alpha \boldsymbol{f}^+(\boldsymbol{y}) \tag{3.15}$$

where:

$$\alpha = \frac{w^-(\boldsymbol{y})}{w^-(\boldsymbol{y}) - w^+(\boldsymbol{y})} \tag{3.16}$$

Furthermore, the Filippov theory specifies that the solution crosses the discontinuity with $\boldsymbol{f}^-$ or $\boldsymbol{f}^+$ when $\alpha = 0$ or $\alpha = 1$, respectively. Although this theory is defined for and ODE [48], the authors extend its applicability to index-1 DAEs by rewriting a DAE as a constrained ODE using three different methods as follows:

1. **Direct Substitution Approach**: This method makes an ODE out of the DAE by substituting the algebraic constraint directly to the differential equation itself.
2. **Singular Perturbation Approach**: This method replaces the algebraic constraint $\boldsymbol{z} = \boldsymbol{k}(\boldsymbol{y})$ with $\epsilon\dot{\boldsymbol{z}} = \boldsymbol{k}(\boldsymbol{y}) - \boldsymbol{z}$ which results the same solution when $\epsilon \to 0$.
3. **Weak Formulation Approach**: This method differentiates the algebraic constraint reaching a differential equation of the algebraic constraint.

An interpolation-based solution to avoid the time step reduction is discussed in [50, 51] where the feedbacks of the systems are interpolated at the switching points in the middle of the time step to determine their on or off state. The suggested approach is useful for dealing with bi-value power electronic switching devices in EMT simulations. Nevertheless, it cannot handle the events of the digital controller as it requires calculating a value for the controller outputs and not just setting or resetting a binary variable.

3.7 Summary

Handling discontinuities during power system dynamic simulations is challenging and computationally expensive. They must be detected in each time step, located accurately, and treated to bridge the solution flow before and after their occurrence.

In this chapter, a variety of different detection, location, and treatment approaches discussed in the literature were reviewed. Furthermore, complex situations arising during the discontinuity handling in each stage and possible solutions suggested in the literature were discussed. In addition, basic concepts of the topics were explored and explained.

Nonetheless, digital controllers' time events are exempt from detection and location stages since their time is known in advance. However, the fast and accurate treatment of thousands of time events arising from their operation has remained an open challenge.

4 Digital Controller Treatment

According to the discussion in the previous sections, the operation of digital controllers introduces many discontinuities to the simulation. Although they are time events and there is no need for the detection and location stages, the constant need for time step reductions to treat them may lead to painfully slow simulations. Therefore, there is a need for a method that is able to simulate power systems with digital controllers quickly and accurately. Current methods are either accurate but computationally heavy, such as the step reduction method (SRM), or light-performing but with the cost of accuracy loss, such as the simplified simulation method (SSM) where the discontinuity is shifted to the time step instant, and analog treatment method (ATM) which removes the discrete nature of the digital controller [7].

In this section, the classical methods of simulating digital controllers are categorized and reviewed first. Then, we propose a novel method for handling time events in a system containing many digital controllers that fills the gap. The method is based on an interpolation treatment of the discrete events, within each time step, without stopping the simulation or reducing the time step. Thus, variable time-step methods are not hindered, and the computational burden is significantly reduced. Thus, the proposed method concerns the first stage of treatment, while the second stage of bridging the solution before and after the discontinuity depends on the choice of integration approach.

4.1 Classical methods

The number of modern digital components in power systems has grown significantly over recent decades. Among these, controllers have evolved rapidly, either through the adoption of advanced digital technologies or by replacing legacy analog systems with digital counterparts. The discrete operation of digital controllers introduces discontinuities into the DAEs that describe power system dynamics. These discontinuities are typically classified as either state or time events [19].

State events are discontinuities that occur due to conditions of the system at unknown times, for example, when a controller hits a limit and certain equations must be altered. In contrast, time events occur at predetermined instances during the simulation, such as when a digital controller samples and updates its output. Simulating systems with state events requires detecting and accurately locating the discontinuity, whereas time events do not, as their occurrence times are known in advance. In this section, methods able to treat the discontinuities arising from the operation of digital controllers are reviewed, and their performance and accuracy are compared to the proposed interpolation-based method (IBM) at the end.

To review the methods, let's consider a simple continuous system under control by a digital controller as schemed in Fig. 4.1.

The continuous system representing a power system can typically be modeled using a DAE set subject to initial values known as the initial value problem (IVP):

$$\begin{aligned} \mathbf{0} &= \boldsymbol{F}(\dot{\boldsymbol{y}}(t), \boldsymbol{y}(t), e(t)) \\ \boldsymbol{y}(0) &= \boldsymbol{y}_0, \;\; e(0) = e_0 \end{aligned} \tag{4.1}$$

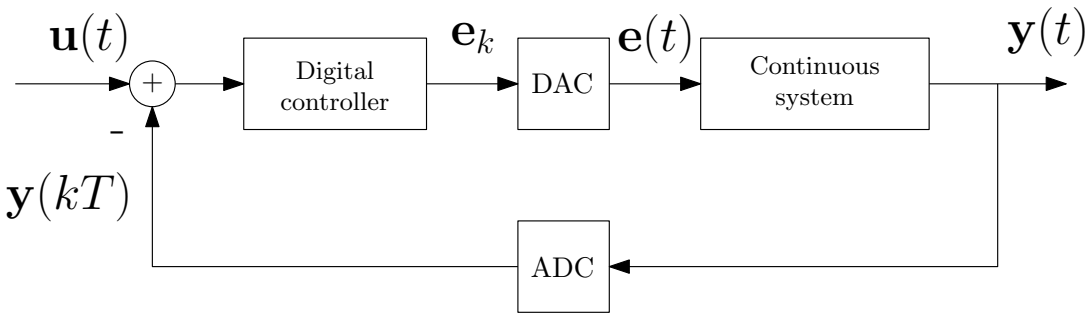

Figure 4.1: A continuous system under control by a digital controller scheme

$\boldsymbol{y}(t)$ denotes the vector of differential-algebraic state variables of the system, $e(t)$ is the continuous output of the controller fed to the system, and $\boldsymbol{y}_0$ and e_0 are the initial values of the state variables and controller output, respectively.

The digital controller modeled with difference equations is described as follows (see 2.1.3):

$$\boldsymbol{e}_k = \boldsymbol{\zeta}(\boldsymbol{e}_{k-1}, \boldsymbol{y}(kT)) \tag{4.2}$$

where $\boldsymbol{\zeta}$ is the controller function relating the controller output $\boldsymbol{e}_k$ to its previous one $\boldsymbol{e}_{k-1}$ and quantized state variables under monitor by the controller $\boldsymbol{y}(kT)$ at the k-th sampling time with period T.

To obtain this IVP problem, (4.1) must be discretized using a numerical integration method, and the resulted algebraic equations will be solved in a time instants t_n over time steps $h_n = t_n - t_{n-1}$, using a Newton method [52]. Therefore, the solution of the state events of the continuous system $\boldsymbol{y}(t_n) = \boldsymbol{y}_n$ will be obtained by solving the residual function iteratively:

$$\boldsymbol{g}(\boldsymbol{y}(t_n), \boldsymbol{e}(t_n)) = 0 \tag{4.3}$$

In the following sections, the methods capable of solving the IVP problem defined in equations (4.1) and (4.2) are discussed.

4.1.1 Analog treatment method (ATM)

To avoid slow simulations for systems with multiple digital controllers, an analog approximation of (4.2) can be used to represent the analog equivalent of the digital controller using a DAE (see 2.1.2):

$$\mathbf{0} = \boldsymbol{\zeta}'(e(t), \boldsymbol{y}(t)) \tag{4.4}$$

where $\boldsymbol{\zeta}'$ represents the continuous approximation of $\boldsymbol{\zeta}$

Then, the controller DAE can be pooled with the continuous system DAE (4.1) and solved all together.

The analog treatment method schemed in Fig. 4.2 completely ignores the discrete nature of the digital controller. Therefore, there is no need for any special treatment like step reductions since no discontinuity is imposed on the simulation. This method is widely used, especially for large-scale dynamic simulation of power systems where multiple digital controllers may need to be simulated, since it makes the sim-

ulation fast and smooth, removing the need for handling difference equations and the events arising. However, due to ignoring the sampling rate of the digital controllers, inaccuracies may appear. Most importantly, ATM cannot simulate non-equation-based digital controllers since they cannot be approximated with DAEs. This limits the application of ATM only to classic equation-based controllers.

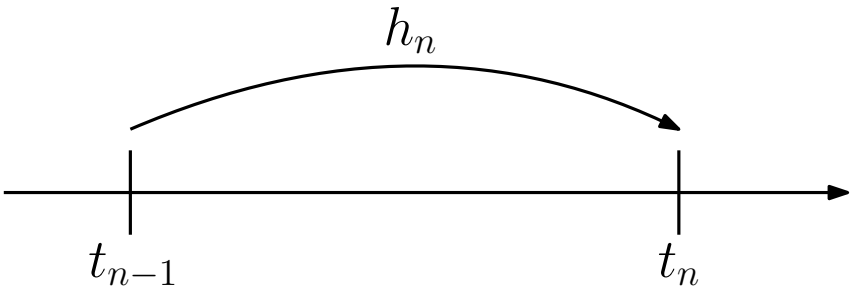

Figure 4.2: Schematic of time integration using analog treatment method with no explicit event handling

4.1.2 Step reduction method (SRM)

The SRM handles each time event of the digital controller sampling by reducing the time step h_n to align precisely with the discontinuity [19] as shown in Fig. 4.3. It then computes the controller output and proceeds to integrate the system equations using that as a constant input to the system DAE. In this approach, equation (4.2) is solved before equation (4.1), with $\boldsymbol{e}(t)$ is assumed to be constant over the entire integration step due to ZOH.

SRM is the most accurate method (closest to the real digital controller behaviour) since it considers all the time events individually and reduces time steps to treat each one of them. However, the time steps become significantly limited (depending on the sampling rate of digital controllers), leading to a computationally heavy simulation (see Fig. 2.3).

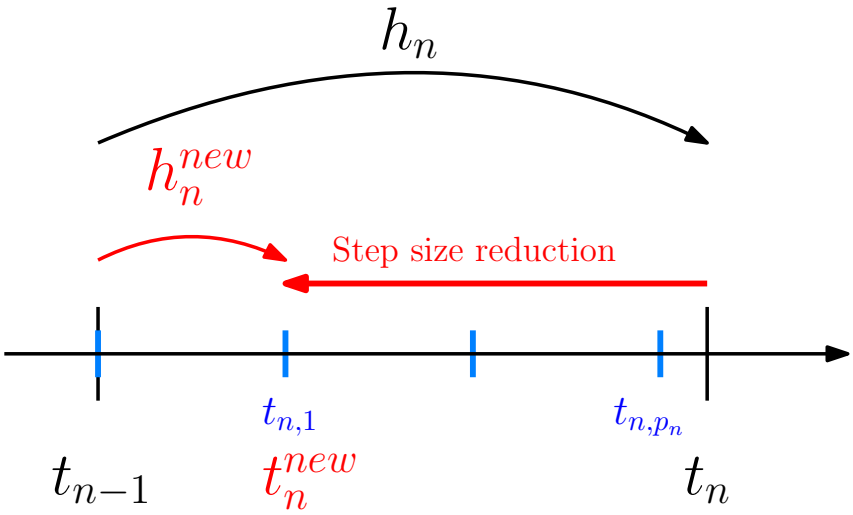

Figure 4.3: Schematic of time integration using step reduction method. The black vertical lines denote the simulation time steps, while the light-blue vertical lines denote the digital controller sampling period.

Upon finding events in the time step h_n, it denies the original time step by taking a new time step with the reduced size equal to $h_n^{new} = t_{n,1} - t_{n-1}$ to land on the first event found in the time step, as

shown in Fig. 4.3. $t_{n,1}$ denotes the first event found in the time step t_n, and p_n is the index of the last event in the same time step.

4.1.3 Simplified simulation method (SSM)

Contrary to SRM, which shifts back the time step to land on the event, the simplified simulation method proposed in [40] shifts the event, forcing it to happen when the time step is taken, as is illustrated in Fig. 4.4. This approach solves the system equations (4.1) first to acquire $\boldsymbol{y}_n$ and then (3.3) to compute e_k, before recomputing the time step.

This method, developed for state events, is not suitable when there are numerous interconnected time events of digital controllers in one time step, since it cannot shift all of them to the end of the time step. The reason is that each controller output is a function of its previous output and the feedback that comes from the system at each sampling time. Therefore, depending on the number of events arising from each digital controller operation in one time step, different levels of inaccuracy may occur.

Nevertheless, the computational performance of SSM is better than SRM since it doesn't reduce the time step and conducts the simulation with relatively large time steps. It also has better performance compared to ATM due to solving a smaller DAE. The reason is that the ATM has the largest DAE set among all the methods due to the controller equations added to it.

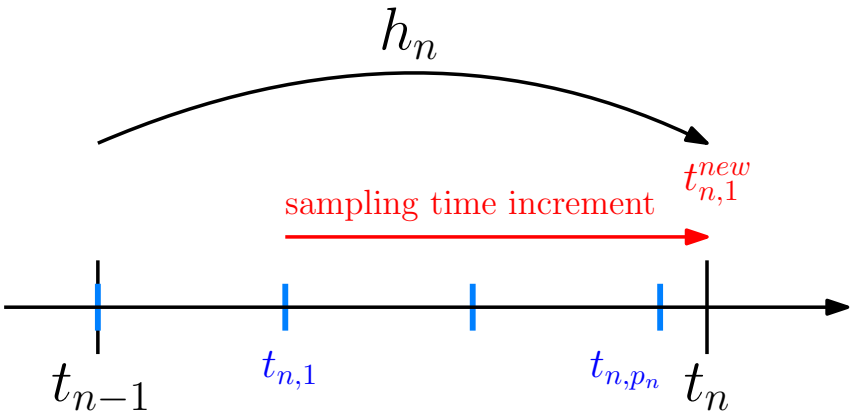

Figure 4.4: Schematic of time integration using the simplified simulation method, with event times shifted to math the solver time steps

4.2 Interpolation-based method

This section presents a novel approach for handling multiple time events in systems with multiple smart digital controllers. The method employs an interpolative treatment of time events within each time step, avoiding the need to halt the simulation or reduce the time step. As a result, variable time-step methods remain fully effective, and computational overhead is significantly reduced. The proposed approach supports phasor-domain dynamic simulations of both equation-based and black-box digital controllers. This allows accurate and efficient modeling of digital sampling actions without simplification, relaxation, or substitution with analog equivalents.

The proposed method leverages interpolation to estimate the system's state variables, which serve as

inputs to the digital controllers. A similar strategy is employed in modern EMT simulators to determine the switching state of power electronic devices [50, 51]. In our approach, digital controller outputs are computed and refined during each Newton iteration of the nonlinear solver by incorporating them into the system's state variable vector. Consequently, these outputs influence both the solver's convergence criteria and the time-step selection through the error estimation scheme.

The interpolation-based method (IBM) is designed for use with variable time-step integration, where step size is governed by error estimation, similar to systems controlled by analog controllers. In other words, the time steps are not constrained or reduced by the digital controllers' time events. This enables the simulation of systems with non-equation-based smart digital controllers while fully retaining the benefits of a variable time-step scheme.

4.2.1 Digital controller interface

To elaborate on IBM, let's again consider the continuous system under control by a digital controller as was shown in Fig. 4.1. For simplicity, let's assume the continuous system is defined in the form of an IVP of ordinary differential equations (ODEs):

$$\begin{aligned} \dot{\boldsymbol{y}}(t) &= \boldsymbol{f}(\boldsymbol{y}(t), \boldsymbol{e}(t)) \\ \boldsymbol{y}(0) &= \boldsymbol{y_0},\ \boldsymbol{e}(0) = \boldsymbol{e}_0 \end{aligned} \tag{4.5}$$

where the digital controller is modeled with difference equations (4.2).

To interface the digital controller with the continuous system, the discrete controller output signal $\boldsymbol{e}_k$ must be converted to a piecewise analog signal denoted by $\boldsymbol{e}(t)$ using a DAC block placed after the controller. Due to the ZOH approach, this block keeps the signal constant between the samples, formulated as:

$$\boldsymbol{e}(t) = \boldsymbol{e}_k, \quad t \in [kT, (k+1)T[\tag{4.6}$$

Similarly, the system output $\boldsymbol{y}(t)$ is converted to a discrete signal using an ADC block to feed to the controller input $\boldsymbol{y}(kT)$.

Typically, the discrete controller input and output signals $\boldsymbol{y}(kT)$ and $\boldsymbol{e}_k$ are quantized. A uniform quantization method with q bits can be used to quantize the digital signals. Defining the measures of the intervals in which the output and input signals $\boldsymbol{e}$ and $\boldsymbol{y}$ take values by $U_{\boldsymbol{e}}$ and $U_{\boldsymbol{y}}$, respectively, a q-bit uniform quantization approach is formulated as[53]:

$$\begin{aligned} \boldsymbol{e}_{q,k} &= \text{round}\left(\frac{\boldsymbol{e}_k 2^q}{U_{\boldsymbol{e}}}\right) \cdot \frac{U_{\boldsymbol{e}}}{2^q} \\ \boldsymbol{y}_q(kT) &= \text{round}\left(\frac{\boldsymbol{y}(kT) 2^q}{U_{\boldsymbol{y}}}\right) \cdot \frac{U_{\boldsymbol{y}}}{2^q} \end{aligned} \tag{4.7}$$

4.2.2 Integration scheme

Now that the digital controller is interfaced with the continuous system, a proper integration method can be used to solve the index-1 DAE set of the combined system. In this work, a variable time step predictor-corrector scheme is designed [22]. So, $\boldsymbol{y}_n$ which is the solution of state variables $\boldsymbol{y}(t_n)$ at time t_n ($n \in \mathbb{N}$)

will be obtained by taking variable in size time steps $h_n = t_n - t_{n-1}$ over the simulation time horizon.

The predictor is used for achieving an initial estimate of the solution, denoted by $\boldsymbol{y}_n^{(0)}$, using an explicit integration method. Then, the corrector calculates the solution $\boldsymbol{y}_n$ using an implicit integration formula combined with Newton iterations. In other words, the solution is gradually refined by solving an implicit set of discretized equations of the system's DAE using the Newton method 2.3.

For this work, we have chosen a pair of second-order Adams-Bashforth and Adams-Moulton methods [22] for the explicit predictor and implicit corrector, respectively. It should be noted that for the sake of comparability, this combination of integration methods is used for all the simulations using SRM, ATM, SSM, and IBM.

4.2.3 Convergence test

During the corrector phase, a convergence test is applied to the normalized state variables vector after each Newton iteration, based on a user-defined threshold. Iterations continue until this convergence criterion is met, with a maximum number of iterations typically imposed to prevent non-converging scenarios. If it fails to converge, the step may be repeated using a smaller time step size, given that the minimum size of the time step is not reached.

4.2.4 Time-step selection

For each time step, an error test is conducted using Milne's estimate [22] to adjust the size of the subsequent time step [22]. According to this approach, in predictor-corrector methods, the local error can be estimated by multiplying the difference between the predictor and corrector by a coefficient determined by the specific method pair. Then its norm is compared to a user-specified tolerance to decide whether to accept or repeat the time step. For the second-order Adams methods [22] used in this work, the estimate is given by:

$$\boldsymbol{d}_n = \left\| \frac{\boldsymbol{y}_n - \boldsymbol{y}_n^0}{3(\frac{h_{n-1}}{h_n})} \right\|_\infty \leq \text{Tolerance} \tag{4.8}$$

Based on the results of (4.8) three situations may arise:

1. If the estimated error is less than the Tolerance set by the user, the next time step sizes will be increased until they reach the Tolerance.
2. If the estimated error is greater than the Tolerance, the current time step will be rejected and the integration will restart for this time step with a smaller size.
3. If the estimated error is greater than the Tolerance and the time step is already at the minimum value, the current time step is accepted, but a warning will be issued to the user to adjust the settings if necessary.

4.2.5 Incorporating digital controllers

When Smart Digital Controllers are incorporated into the dynamic simulation, several discrete events are introduced into the process described above. Figure 4.5 illustrates the 'normal' time steps (h_n) and their corresponding solutions ($\boldsymbol{y}_{n-1}$ and $\boldsymbol{y}_n$), based on the variable time-step method. It also shows

the sampling times of the smart digital controllers, along with their intermediate solutions and outputs ($\boldsymbol{y}_{n,i}$ and $\boldsymbol{e}_{n,i}$). The black horizontal piecewise lines represent the ZOH controller signal $\boldsymbol{e}(t)$, while the approximate solutions are marked with blue crosses for the main time steps and red crosses for the intermediate ones.

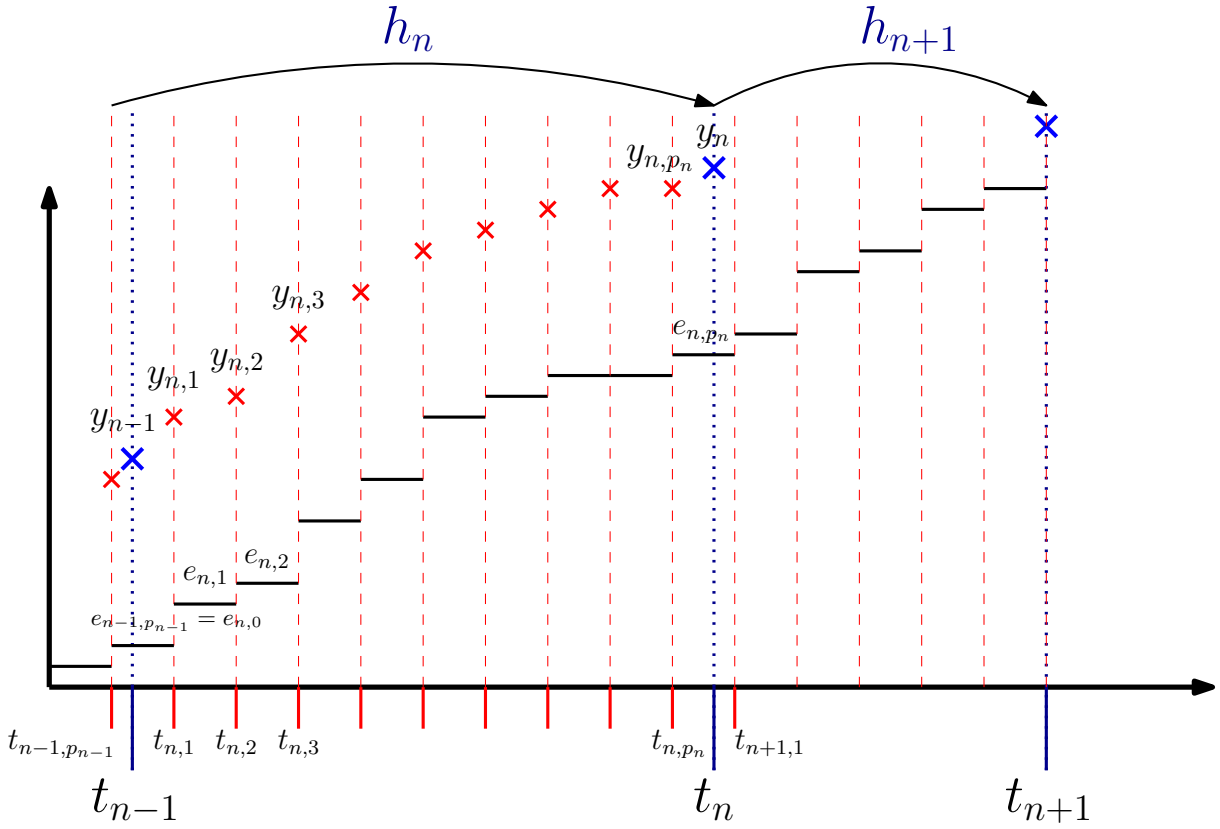

Figure 4.5: Example of the integration time steps and the intermediate control actions in a variable time-step simulation with smart digital controllers.

Let p_n denote the number of digital controller actions occurring within the time step $h_n = t_n - t_{n-1}$. We represent the value of $\boldsymbol{e}$ at time $t_{n,g}$ as $\boldsymbol{e}_{n,g}$, where $g \in [1, p_n]$ and $t_{n-1} < t_{n,g} < t_n$. It is important to note that the sampling times kT do not necessarily align with the integration time steps t_n, as illustrated in Fig. 4.5.

As stated before, accurately and efficiently solving this combined system presents significant challenges. A traditional approach involves reducing the time step h_n to align with the controller actions, which greatly restricts the use of variable time-step methods. To overcome the resulting computational and accuracy issues, a new method is proposed in the following section.

4.2.6 State events

As previously mentioned, a state event occurs due to internal system conditions, for example, a transformer tap change triggered by a voltage violation or a variable reaching its limit. In systems controlled by continuous analog controllers, such events must be detected during integration and precisely located within a time step. To ensure accuracy, the time step must be adjusted to land exactly on the event. This approach allows the continuous controller to respond without delay, maintaining high fidelity in the

simulation.

However, in systems with digital controllers, the discrete nature of control means that a state event occurring within the system will not be addressed until the controller's next sampling instant. In other words, the state event's impact is shifted to the next sampling action of the controller response, or it is effectively delayed until the next scheduled control action.

An illustration of this phenomenon is shown in Fig. 4.6. The vertical lines represent the digital controller's sampling actions within the time step h_n, while the red cross marks the occurrence of a state event between two sampling instants. That is the instant when the event triggering conditions are satisfied. However, due to the discrete nature of digital control, the controller cannot respond until the following sampling action. As a result, the state event is handled with a delay. This implies that, in simulations of systems controlled solely by digital controllers, state events effectively transform to time events, eliminating the need for event detection or time-step refinement to precisely capture their occurrence.

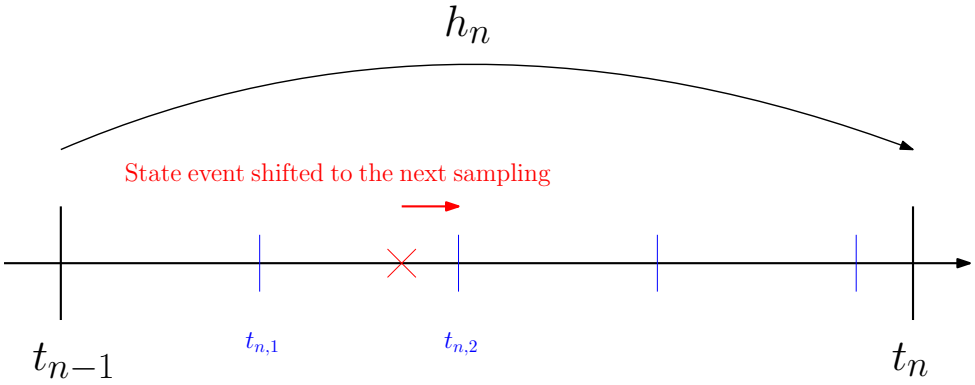

Figure 4.6: A scheme showing the state event shifted to the next time event in a system controller by digital controllers.

Finally, it is important to note that if a state event is not intended to be handled by a digital controller, the typical approach is to reduce the time step to accurately capture the event. However, this does not impose a significant computational burden on the solver, as state events occur relatively infrequently compared to the more frequent time events associated with digital controllers' operation.

4.2.7 Formulation

This section provides the formulation of IBM that allows the discrete events of digital controller actions to be handled. IBM takes large time steps over multiple events, including their impact on the solution by interpolating the state variables of the system at sampling points of the digital controller ($t_{n,g}$), and calculating the controller outputs as new state variables in the same Newton solution.

Let's consider that the objective is to find the solution $\boldsymbol{y}_n$ over the time step t_n with the size $h_n = t_n - t_{n-1}$. We start with the explicit stage of IBM obtaining a preliminary solution $\boldsymbol{y}_n^0$ and controller outputs $\boldsymbol{e}_{n,g}^0$. The results of this stage are then used in the implicit corrector stage as initial values.

Moving to the corrector stage, the residual function for the implicit integration method to be solved for t_n denoted by $\boldsymbol{g}$ can be defined as follows:

$$\boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{e}_{n,p_n}) = 0 \tag{4.9}$$

It should be noted that only the last controller output $\boldsymbol{e}_{n,p_n}$ is required to obtain the solution $\boldsymbol{y}_n$. However, it depends on the previous controller outputs, and they depend on the system's state variables at sampling points. Therefore, the controller outputs can be formulated:

$$\begin{aligned}
&\boldsymbol{e}_{n,1} = \zeta(\boldsymbol{e}_{n,0}, \boldsymbol{y}_{n,1}, t_{n,1}) \\
&... \\
&\boldsymbol{e}_{n,g} = \zeta(\boldsymbol{e}_{n,g-1}, \boldsymbol{y}_{n,g}, t_{n,g}) \\
&... \\
&\boldsymbol{e}_{n,p_n} = \zeta(\boldsymbol{e}_{n,p_{n-1}}, \boldsymbol{y}_{n,p_n}, t_{n,p_n})
\end{aligned} \tag{4.10}$$

where $e_{n,0}$ is equal to the last controller output in the previous time step t_{n-1} due to ZOH, and $\boldsymbol{y}_{n,g}$, $g \in [1, p_n]$ is the system state variables fed back to the controller as inputs at sampling points of the controller $t_{n,g}$. The time step for each sampling time of the digital controller is equal to $h_{n,g} = t_{n,g} - t_{n-1}$.

Now the variables are defined, the intermediate state variables $\boldsymbol{y}_{n,g}$ will be estimated using an interpolation function $\boldsymbol{w}_n$. Using the interpolated values, the residuals for the controller outputs can be formed to be solved alongside the state variables of the system. We have used a second-order Taylor expansion formula for the interpolator as follows:

$$\begin{aligned}
\boldsymbol{y}_{n,g} =& \boldsymbol{w}_n(t_{n,g}) = \boldsymbol{y}_{n-1} + h_{n,g}\dot{\boldsymbol{y}}_{n-1} \\
&+ \frac{h_{n,g}^2}{h_n^2}(\boldsymbol{y}_n - \boldsymbol{y}_{n-1} - h_n\dot{\boldsymbol{y}}_{n-1})
\end{aligned} \tag{4.11}$$

It should be noted that to bound and control the integration method's error, it should have a higher order of local error than the interpolation polynomial [54]. Therefore, for most second-order integration methods, the Taylor interpolant polynomial with an order equal to the integration method (2nd order) will be adequate.

The equations (4.10) are now included in the equations set of the system to be solved for time step t_n. Thus, the state variables of the system are extended by the controller outputs for the time step, adding p_n new variables associated with $\boldsymbol{e}_{n,g}$ ($g \in [1, p_n]$).

To form the formulation of the new extended problem, we define a new state variable vector $\boldsymbol{z}_n$, pooling together the state variables of the system $\boldsymbol{y}_n$ and the controller signals $\boldsymbol{e}_{n,g}$ ($g \in [1, p_n]$):

$$\boldsymbol{z}_n = \begin{bmatrix} \boldsymbol{z}_{n,1} \\ \boldsymbol{z}_{n,2} \end{bmatrix} \tag{4.12}$$

with:

$$\boldsymbol{z}_{n,1} = \boldsymbol{y}_n \tag{4.13}$$

$$\boldsymbol{z}_{n,2} = \begin{bmatrix} \boldsymbol{e}_{n,1} & \boldsymbol{e}_{n,2} & \dots & \boldsymbol{e}_{n,g} & \dots & \boldsymbol{e}_{n,p_n} \end{bmatrix}^T \tag{4.14}$$

We also consider a new residual vector for the new state variable vector $\boldsymbol{z}_n$:

$$\tilde{\boldsymbol{g}} = \begin{bmatrix} \tilde{\boldsymbol{g}}_1 \\ \tilde{\boldsymbol{g}}_2 \end{bmatrix} \tag{4.15}$$

with:

$$\tilde{\boldsymbol{g}}_1(\boldsymbol{z}_n) = \boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{e}_{n,p_n}) \tag{4.16}$$

$$\tilde{\boldsymbol{g}}_2(\boldsymbol{z}_n) = \begin{bmatrix} \boldsymbol{e}_{n,1} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,0}, \boldsymbol{y}_{n,1}) \\ \dots \\ \boldsymbol{e}_{n,g} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,g-1}, \boldsymbol{y}_{n,g}) \\ \dots \\ \boldsymbol{e}_{n,p_n} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,p_{n-1}}, \boldsymbol{y}_{n,p_n}) \end{bmatrix} \tag{4.17}$$

Consequently, and writing it in terms of interpolating polynomials it will be written:

$$\tilde{\boldsymbol{g}}(\boldsymbol{z}_n) = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{e}_{n,p_n}) \\ \boldsymbol{e}_{n,1} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,0}, \boldsymbol{w}_n(t_{n,1})) \\ \dots \\ \boldsymbol{e}_{n,g} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,g-1}, \boldsymbol{w}_n(t_{n,g})) \\ \dots \\ \boldsymbol{e}_{n,p_n} - \boldsymbol{\zeta}(\boldsymbol{e}_{n,p_{n-1}}, \boldsymbol{w}_n(t_{n,p_n})) \end{bmatrix} \tag{4.18}$$

Now, the new residual vector $\tilde{\boldsymbol{g}}$ can be solved for $\boldsymbol{z}_n$ to find $\boldsymbol{y}_n$.

4.2.8 Newton iterations

The extended problem, combining the state variables of the system and the controller outputs, can be solved iteratively using a Newton scheme. At the m-th Newton iteration, the solution vector and interpolator function are defined as $\boldsymbol{y}_n^{(m)}$ and $\boldsymbol{w}_n^{(m)}$, respectively. Also, $\boldsymbol{y}_{n,g}^{(m)}$ and $\boldsymbol{e}_{n,g}^{(m)}$ are the value as at the interpolated steps. The IBM internal equations are thus expressed as:

$$\begin{aligned} \boldsymbol{y}_{n,g}^{(m)} &= \boldsymbol{w}_n^{(m)}(t_{n,g}), \quad \forall g \in [1, p_n] \\ \boldsymbol{e}_{n,g}^{(m)} &= \boldsymbol{\zeta}(\boldsymbol{e}_{n,g-1}^{(m)}, \boldsymbol{y}_{n,g}^{(m)}), \quad \forall g \in [1, p_n] \end{aligned} \tag{4.19}$$

Now, the Newton iteration to be solved is formulated:

$$\boldsymbol{J}_n^{(m)}(\boldsymbol{z}_n^{(m+1)} - \boldsymbol{z}_n^{(m)}) = -\tilde{\boldsymbol{g}}(\boldsymbol{z}_n^{(m)}) \tag{4.20}$$

where $\boldsymbol{J}_n^{(m)}$ is the Jacobian matrix of the extended system (4.18) at the m-th Newton iteration. Since the state variables extend by the number of controller outputs in each time step p_n, and due to using a variable time step method, the size of the Jacobian will change over different time steps with different

sizes. Therefore, it can be formulated as:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \end{bmatrix} \tag{4.21}$$

Computing $\boldsymbol{J}_n^{(m)}$ can be highly challenging, particularly when the smart digital controller is proprietary or operates as a black box. Although automatic differentiation tools exist for numerically computing the Jacobian [55], it is important to note that the equations and states in (4.17) correspond to different time instances, and interpolation-based approximations may vary significantly between iterations.

4.2.9 Jacobian

Sub-matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ correspond, respectively, to the controlled system Jacobian, the impact of the controllers on the controlled system, the impact of the system on the controllers, and the controller system Jacobian. To mitigate the Jacobian computation challenges and improve computational efficiency, an approximate Jacobian matrix can be utilized. In this context, the following approximations can be adopted.

4.2.9.1 IBM-A

The most simplified Jacobian can be defined as the following:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \approx 0 \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} \approx \boldsymbol{0} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{I}_{p_n} \end{bmatrix} \tag{4.22}$$

where the sub-matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ are replaced by zeros. In other words, the impact of controllers on the system and the impact of the system on the controller are neglected within a Newton iteration. Also, the sub-matrix $\boldsymbol{D}$ is replaced with an identity matrix $\boldsymbol{I}_{p_n}$, indicating that the relationship between the controller outputs and their previous ones is neglected. Its size is determined by the number of controller sampling actions in the time step p_n.

Since this is a dishonest Jacobian, it is expected that a larger number of Newton iterations will be required to converge, compared to the exact Jacobian with all the sub-matrices accurately included.

4.2.9.2 IBM-AB

This approximation maintains sub-matrix $\boldsymbol{C}$ to zero but considers the sub-matrix $\boldsymbol{B}$, which includes the impact of controller outputs on the system's equations:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \approx \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial e_{n,p_n}} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} \approx \boldsymbol{0} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{I}_{p_n} \end{bmatrix} \tag{4.23}$$

It is noticeable that in each time step, only the last output of the controller feeds back to the system. Therefore, $\boldsymbol{B}$ is a sparse matrix that has only one non-zero element per controller input. This approximation leads to an upper block triangular Jacobian matrix.

4.2.9.3 IBM-AC

This approximation maintains sub-matrix B to zero but considers the sub-matrix C, which has the impact of the system state variables on the controller equations:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \approx 0 \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} \approx \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{y}_n} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{I}_{p_n} \end{bmatrix} \tag{4.24}$$

This sub-matrix is also a sparse matrix with non-zero elements for each sampling of the controller per system variable that is monitored. This approximation leads to a lower block triangular Jacobian matrix.

Computing this sub-matrix needs four function evaluations per sampling action per Newton iteration per system's equation (assuming a second-order interpolation polynomial), which, depending on the system's size and number of samplings, may lead to a major performance drop.

4.2.9.4 IBM-ABC

This variation has both sub-matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ included simultaneously. Although it is expected to have fewer Newton iterations using this Jacobian variation, the overall computation may increase due to calculating both sub-matrices $\boldsymbol{B}$ and $\boldsymbol{C}$.

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \approx \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial e_{n,p_n}} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} \approx \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{y}_n} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{I}_{p_n} \end{bmatrix} \tag{4.25}$$

4.2.9.5 Decoupled IBM

The core idea is to decouple the system and controller equations into two separate Newton loops, where the output of each loop is used to update the inputs of the other in subsequent iterations. This approach effectively acts as an iterative decoupled IBM (DIBM) embedded within the overall Newton iteration process.

Initially, the system equations are solved, and the resulting solution is used to compute updated controller outputs. These updated outputs then serve as inputs for the next iteration of the system solution. This iterative process continues until both the system and controller solutions converge. A simplified schematic of the DIBM approach is shown in Fig. 4.7.

To better describe DIBM, let's again consider the state variables and controller outputs formulated as (4.13) and (4.14), respectively. In addition, the residuals are formulated as (4.16) and (4.17) for the system and controller equations, respectively. In the first stage, (4.13) and (4.16) result in the state variables and their associated mismatches used to solve the systems equations in the m-th Newton iteration as formulated in the following:

$$\boldsymbol{A}_n^{(m)}(\boldsymbol{z}_{n,1}^{(m+1)} - \boldsymbol{z}_{n,1}^{(m)}) = -\tilde{\boldsymbol{g}}_1(\boldsymbol{z}_{n,1}^{(m)}) \tag{4.26}$$

where $\boldsymbol{A}_n^{(m)}$ is the sub-matrix of the Jacobian regarding the system's equations (see 4.21).

The state variables computed in the first stage trigger the second stage, where they are used for the interpolation function in (4.19) to obtain the interpolated state variables for each sampling time. Then, (4.14) and (4.17) result in a new set of controller outputs and associated mismatches, respectively. Finally,

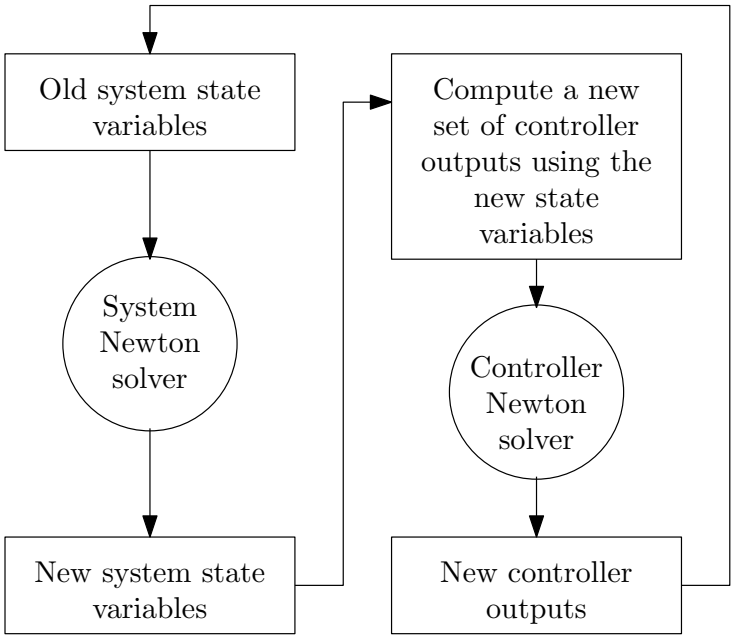

Figure 4.7: A simple scheme showing Newton iterations in DIBM.

the following Newton formula results in the corrected controller outputs for the next iteration:

$$\boldsymbol{D}_n^{(m)}(\boldsymbol{z}_{n,2}^{(m+1)} - \boldsymbol{z}_{n,2}^{(m)}) = -\tilde{\boldsymbol{g}}_2(\boldsymbol{z}_{n,2}^{(m)}) \tag{4.27}$$

where $\boldsymbol{D}_n^{(m)}$ is the identity matrix in 4.21.

This process repeats until the combined state variables vector z_n converges.

4.2.10 Jacobian comparison

Let's assume that the number of continuous system equations is a, and for the digital controller equations is b.

In the case of the ATM, the Jacobian matrix used to solve the problem has a fixed size of $(a + b) \times (a + b)$, as all equations are solved simultaneously at each time step. This matrix size remains constant throughout the simulation.

For both SRM and SSM, the Jacobian matrix size depends solely on the number of continuous system equations and is equal to $a \times a$, as the controller outputs are computed separately (either before or after solving the DAEs at each time step), and this size remains constant throughout the simulation. Therefore, ATM has a larger system to solve at each time step.

In contrast to the previous methods, which maintain a constant Jacobian matrix size throughout the simulation, the Jacobian matrix in IBM varies at each time step. Its size depends on the simulation time step h_n and the number of digital controller samples p_n within that interval. For instance, with a single digital controller containing one equation ($b = 1$), the Jacobian matrix takes the form $(a+p_n) \times (a+p_n)$.

The increased size of the Jacobian matrix (due to multiple controller actions within a single time step) can result in a matrix significantly larger than the system itself. For example, consider a small system with ten DAEs ($a = 10$) and four controllers ($N_C = 4$), each performing five control actions within a

time step. This yields $4 \times 5 = 20$ interpolation points, leading to a Jacobian matrix of size 30×30, which is nine times larger than the original system. Furthermore, since the number of control actions p_n can vary at each time step, the Jacobian must be frequently recomputed. To balance performance and accuracy, a so-called "dishonest" Jacobian approach [56] may be employed, where certain sub-matrices ($\boldsymbol{B}$, $\boldsymbol{C}$, or $\boldsymbol{D}$ in (4.21)) are simplified. The dishonest Newton method preserves accurate residual computations while using an approximate Jacobian, reducing overall computation time at the cost of a slower convergence rate [57].

It is important to note that, in all variations, the sub-matrix $\boldsymbol{D}$ is approximated by an identity matrix due to the opaque nature of the controllers. Specifically, as indicated by (4.17), $\boldsymbol{D}$ is inherently a bidiagonal band matrix with non-zero entries along the main diagonal and the sub-diagonal. However, when controllers are treated as input-output black-box models, computing the sub-diagonal elements automatically becomes highly complex or even infeasible. As a result, the sub-diagonal is set to zero, effectively approximating $\boldsymbol{D}$ as an identity matrix.

4.2.11 Flowchart

The flowchart of the proposed approach for a single time step is shown in Fig. 4.8. It should be noted that the extended state variable vector $\boldsymbol{z}_n$ is employed for both the convergence check and error estimation, as illustrated in the flowchart. Consequently, the outputs of the controllers influence the time step adjustment process.

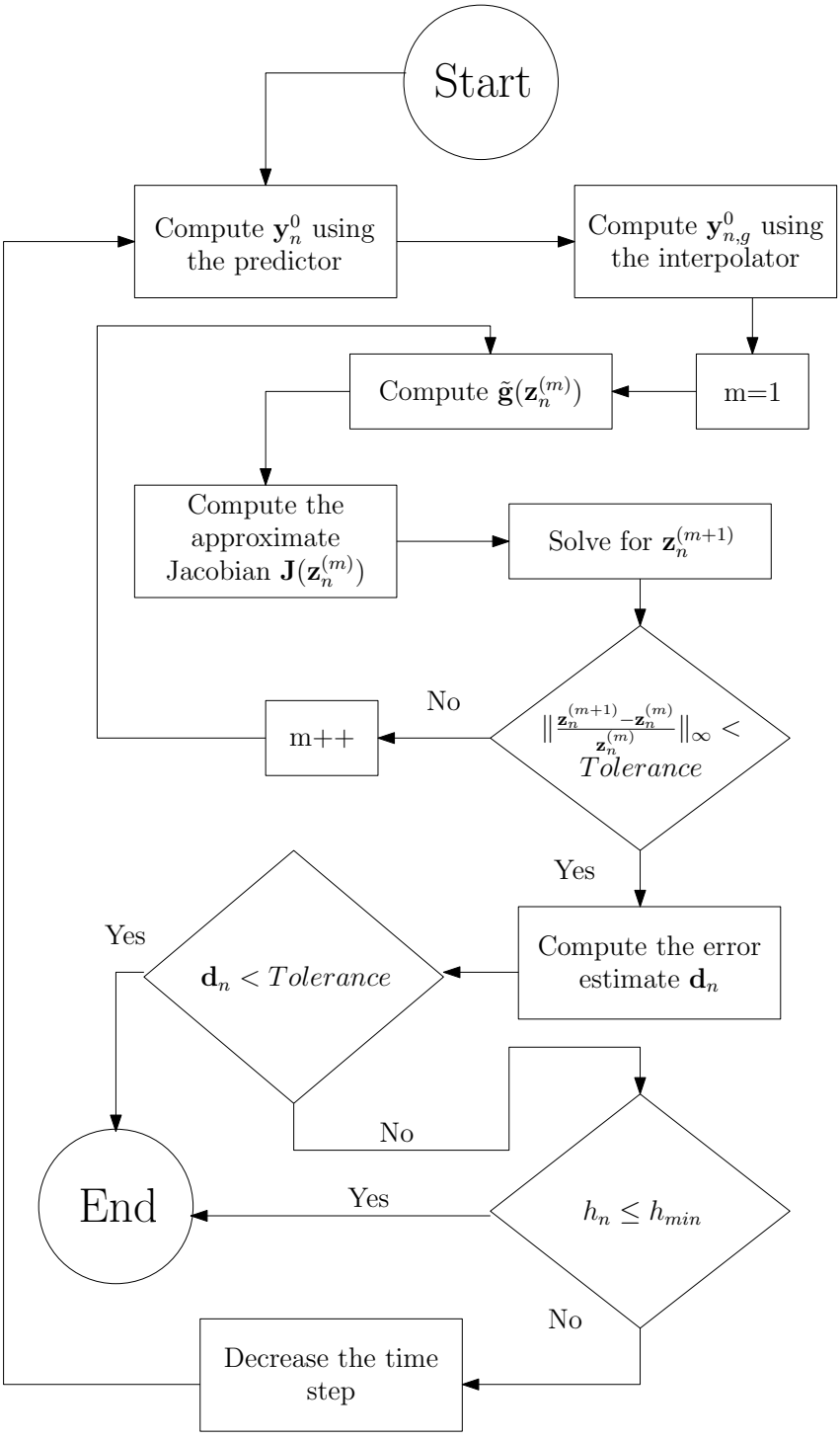

Figure 4.8: Flowchart of the interpolation-based method for one time step

4.3 Simulation results

In this section, three sets of simulations are conducted to showcase the performance and accuracy of IBM as follows:

- The first set of simulations showcases the performance and accuracy of IBM compared to SRM.
- The second set explores the accuracy and performance of DIBM compared to IBM.

- Finally, a comparison of IBM with SRM, ATM, and SSM is presented in the third set of simulations. It also discusses the Jacobian variations of IBM and their impact on performance.

A variable-step predictor-corrector integration method, based on a second-order Adams–Bashforth and Adams–Moulton pair, is employed to solve all case studies [22]. For the IBM approach, a second-order Taylor expansion polynomial is used for interpolation. The same set of system and controller DAEs is solved across all methods, except for ATM, which utilizes analog equivalents. Additionally, uniform quantization with 16-bit resolution is assumed for both the ADC and DAC components. The time-step adjustment factors are set to 1.25 for step size increase and 0.5 for decrease. The minimum and maximum allowable time steps are defined as 1 ms and 1 s, respectively.

Furthermore, Euclidean distance is used for comparing the accuracy of different trajectory results of different methods, as formulated [58]:

$$d(\alpha, \beta) = \sqrt{\sum_{n=1}^{N} (\alpha_i - \beta_i)^2} \tag{4.28}$$

where d denotes the distance between time series of trajectories α and β, and N is the total number of time steps. It is noteworthy that due to the variable time-step approach, the time-series trajectories might not align at the same points in time. Thus, a first-order linear interpolation is first used to estimate the solution at points between them. All solvers for all methods are implemented in MATLAB 2021 [59].

4.3.1 IBM

In this section, three test cases are considered for evaluation of the performance and accuracy of IBM compared to the classical SRM approach:

1. A general system consisting of two ODEs controlled by an equation-based integral digital controller.
2. A single-machine infinite-bus (SMIB) system having a synchronous generator controlled by an equation-based governor digital controller and a black-box fuzzy AVR digital controller connected to an infinite bus.
3. A modified Kundur test system consisting of 4 synchronous generators and an HVDC link between two areas. The first two generators have equation-based AVRs, while the rest have black-box fuzzy AVRs. The same equation-based governor digital controller is used for all of them. Furthermore, an equation-based power oscillations damping (POD) controller is considered for the HVDC link.

4.3.1.1 A single integral controller

In this section, a test system controlled by a single integral controller is considered. Referring back to Fig. 4.1, for the continuous system a set of two differential equations formulated as (4.29) is considered:

$$\underbrace{\begin{bmatrix} \dot{y_1} \\ \dot{y_2} \end{bmatrix}}_{\dot{\boldsymbol{y}}} = \underbrace{\begin{bmatrix} a & b \\ -b & 0 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{\boldsymbol{y}} + \underbrace{\begin{bmatrix} -b \\ 0 \end{bmatrix}}_{\boldsymbol{B}} e(t) \tag{4.29}$$

An integral controller with a gain $K_I = 0.07$ and sampling time $T = 0.1$ s is considered. The digital integral controller is defined as:

$$e_k = e_{k-1} + K_I T(u(kT) - y_2(kT)) \tag{4.30}$$

where $u(kT)$ is the set point (considered constant).

The simulated output for a step set point change is shown in Figs. 4.9 and 4.10 for one stable and one unstable system, respectively. As can be seen, no visible difference can be detected between the output trajectories of SRM and IBM.

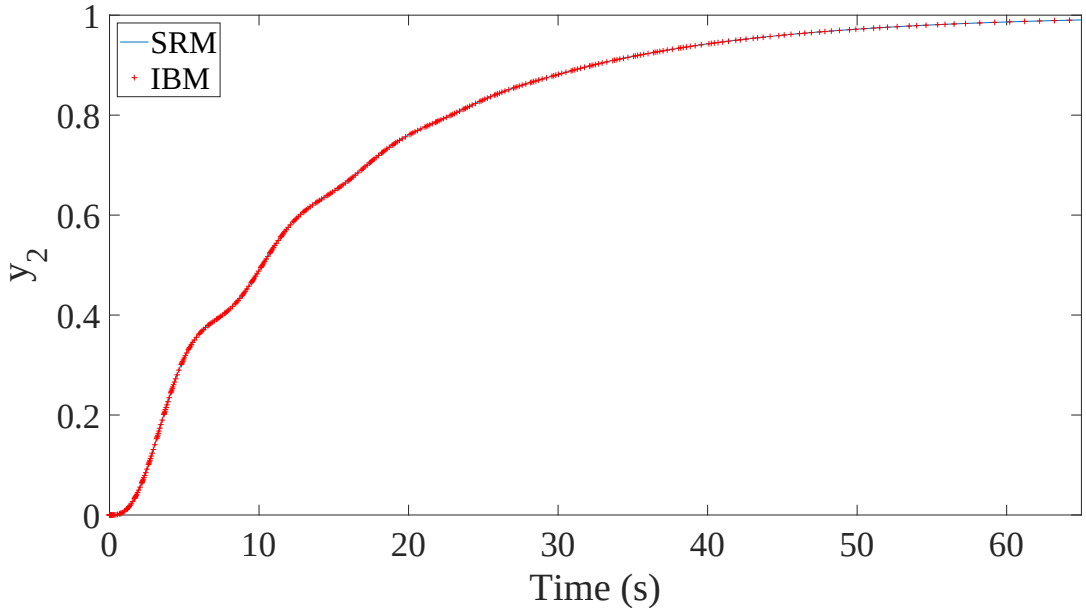

Figure 4.9: Output results for the stable system with the Integral controller

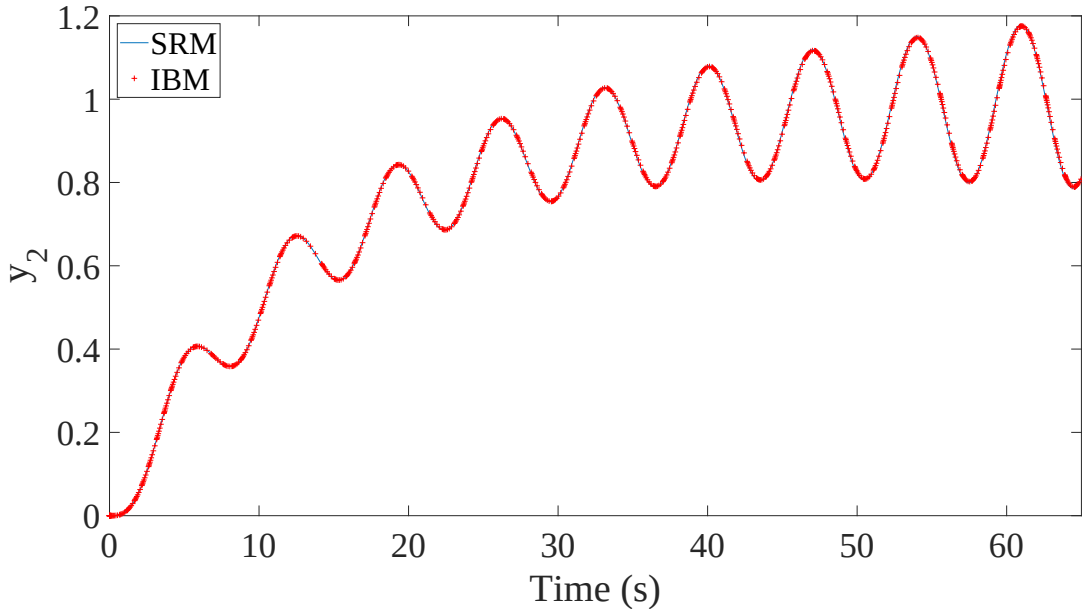

Figure 4.10: Output results for the unstable system with the Integral controller

The total number of Newton iterations required to solve the system is summarized in Table 4.1 as a benchmark to assess the performance of IBM compared to SRM. It can be seen that, in both the stable and unstable scenarios, IBM is faster than SRM. This can be explained from Fig. 4.11 and Fig. 4.12, where the time-step size is plotted for both methods. While SRM is blocked by the digital controller sampling time, IBM manages to increase the time-step size without sacrificing accuracy.

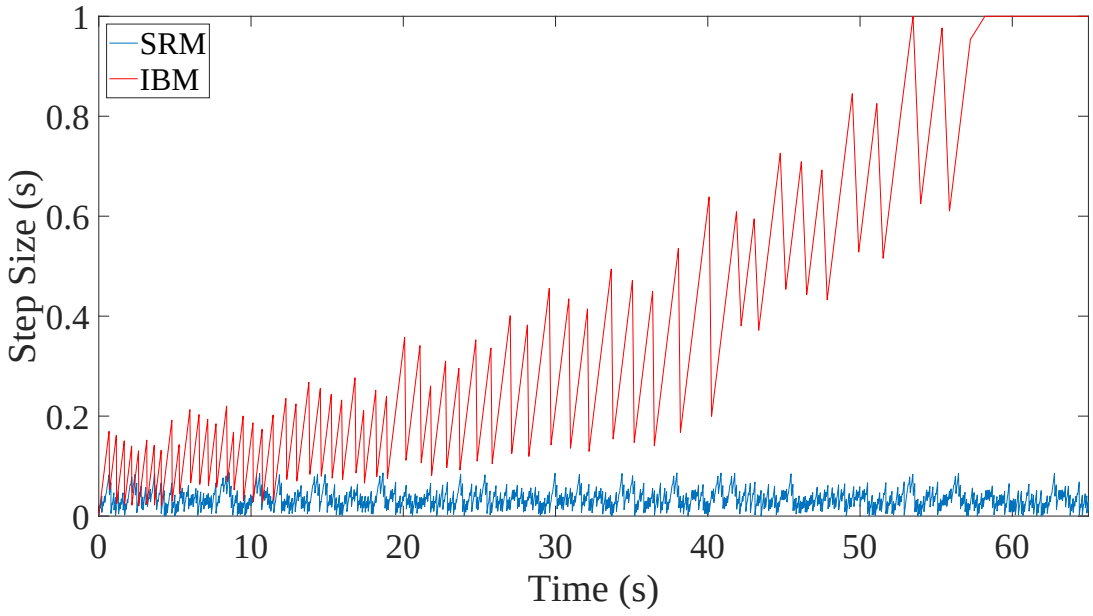

Figure 4.11: Step size results for the stable system with the Integral controller

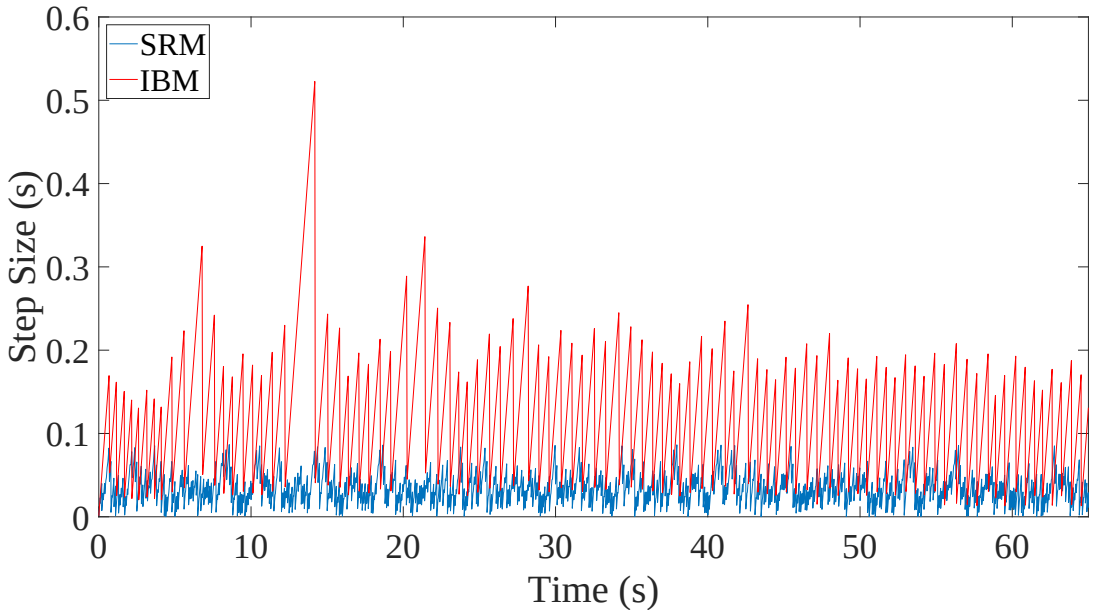

Figure 4.12: Step size results for the unstable system with the Integral controller

Table 4.1: Performance comparison between IBM and SRM for a system containing one integral controller in terms of the number of Newton iterations

System parameters	SRM	IBM
Stable case	5473	1133
Unstable case	5333	2403

4.3.1.2 Single machine model with excitation system and governor

In this section, a synchronous machine connected to an infinite bus (SMIB) by a tie of transmission lines is considered as shown in Fig. 4.13. The synchronous machine is controlled by a fuzzy-based excitation controller and a governor, as illustrated in Fig. 4.14. The DAE set describing the generator and the schematic of the exciter and governor systems are presented in Appendix I. It is noticeable that while the governor is an equation-based controller, the fuzzy AVR is non-equation-based and modeled as a black-box with an input and output [60]. The sampling times of the Governor and AVR are considered equal to 20 and 6 ms, respectively.

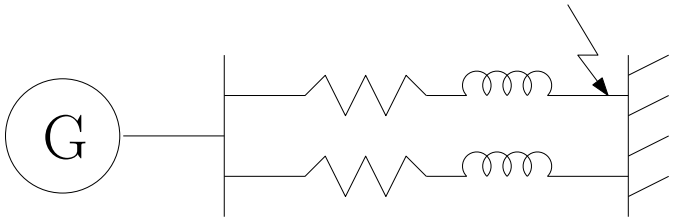

Figure 4.13: Schematic of the single machine infinite bus system

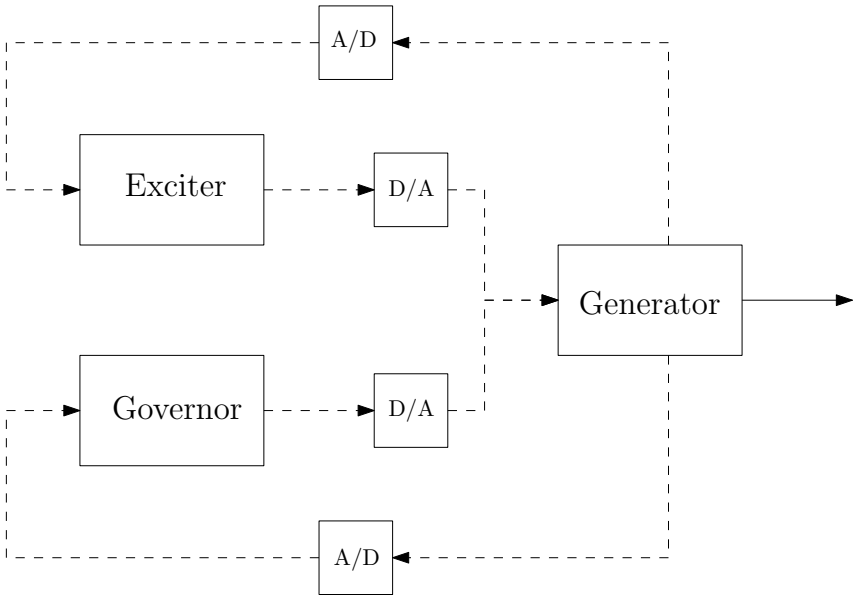

Figure 4.14: Overview of a synchronous machine with digital Governor and Exciter digital controllers

A short circuit with low impedance is applied at the end of the line connected to the infinite bus at time $t = 1$ s and cleared after 0.2 s by disconnecting the line. Figure 4.15 shows the output voltages of the generator. Furthermore, the output active and reactive power, and frequency of the generator are

illustrated in Figs. 4.16, 4.17, and 4.18. In addition, the output signal for the excitation system and the governor is illustrated in Figs. 4.19 and 4.20. It can be seen that the simulation outputs are identical in both methods for all the simulated signals.

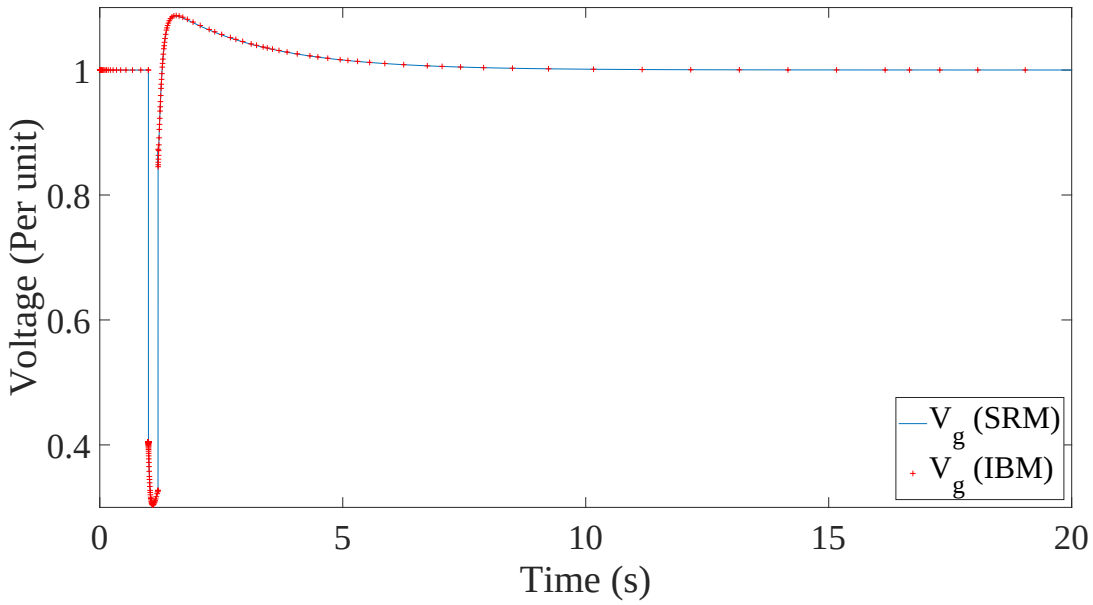

Figure 4.15: The generator's voltage output of SMIB system

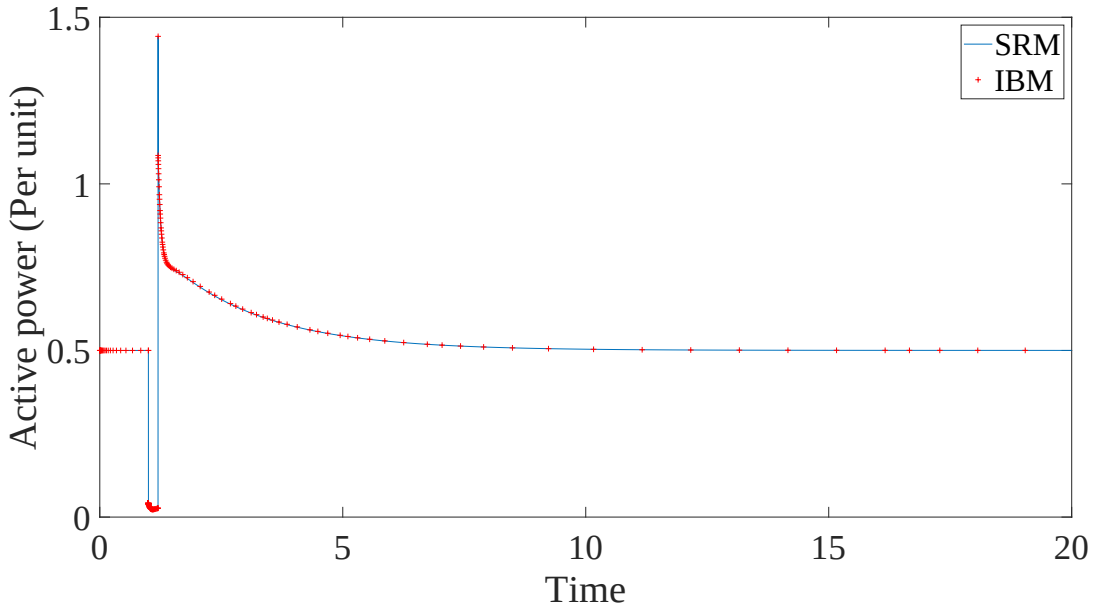

Figure 4.16: Active power of the generator of SMIB system

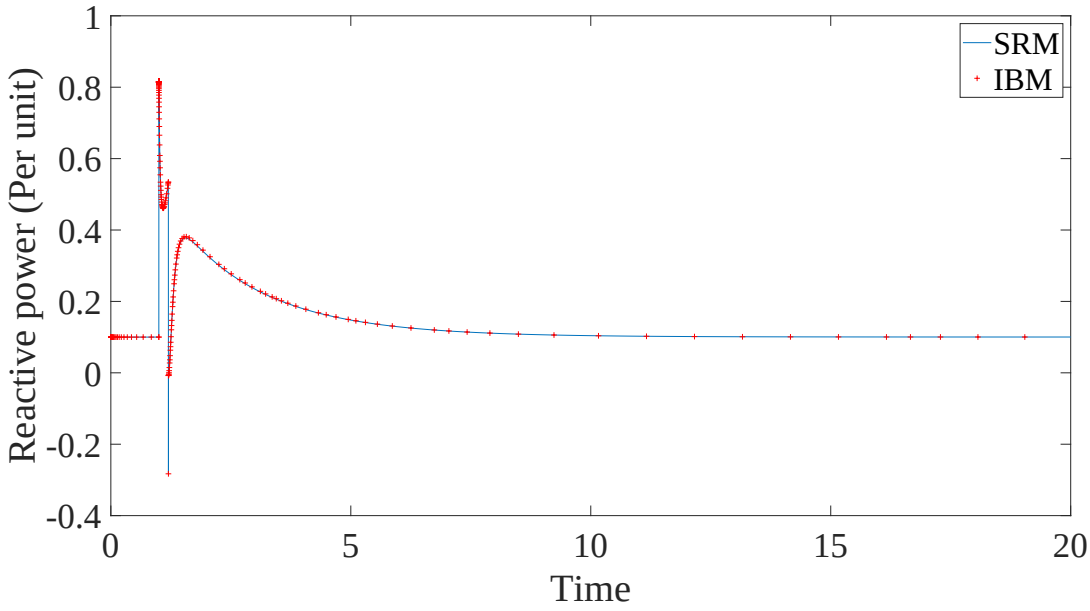

Figure 4.17: Reactive power of the generator of SMIB system

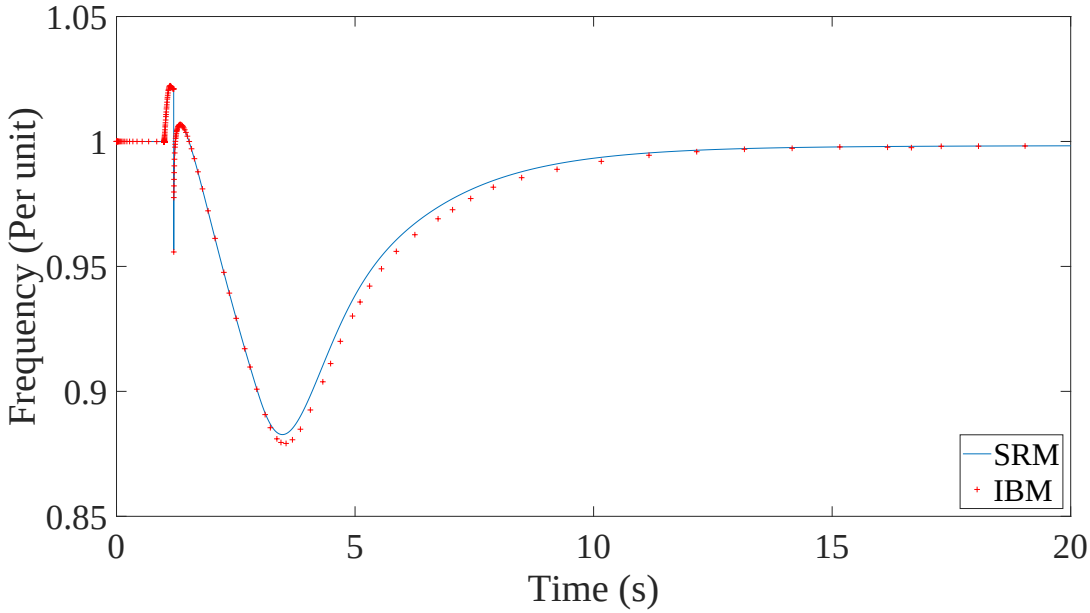

Figure 4.18: Frequency of the generator of SMIB system

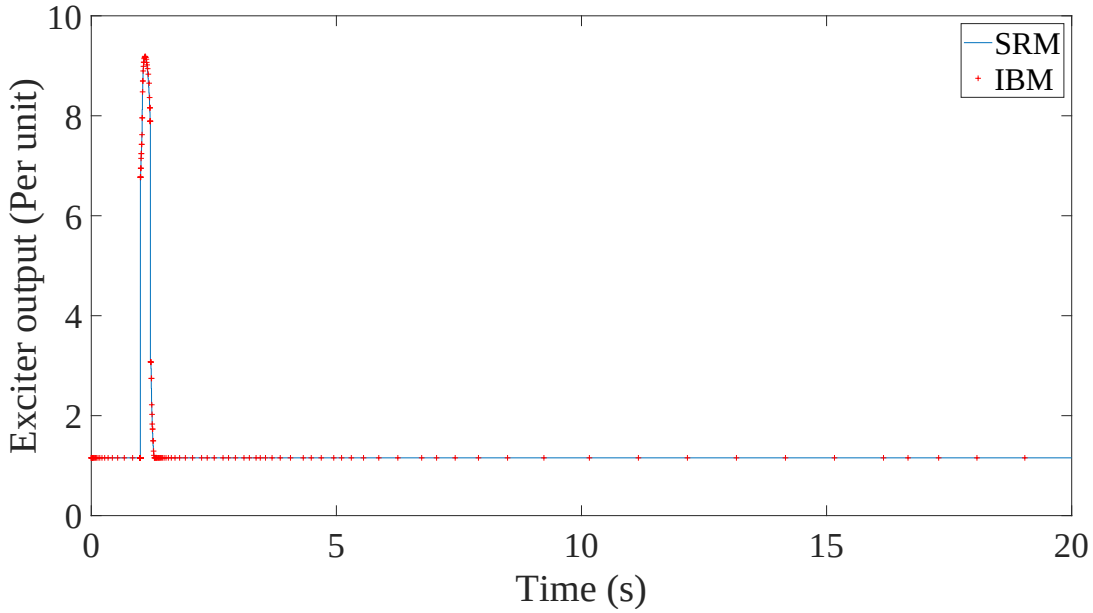

Figure 4.19: Output of the digital excitation system of the generator of SMIB system

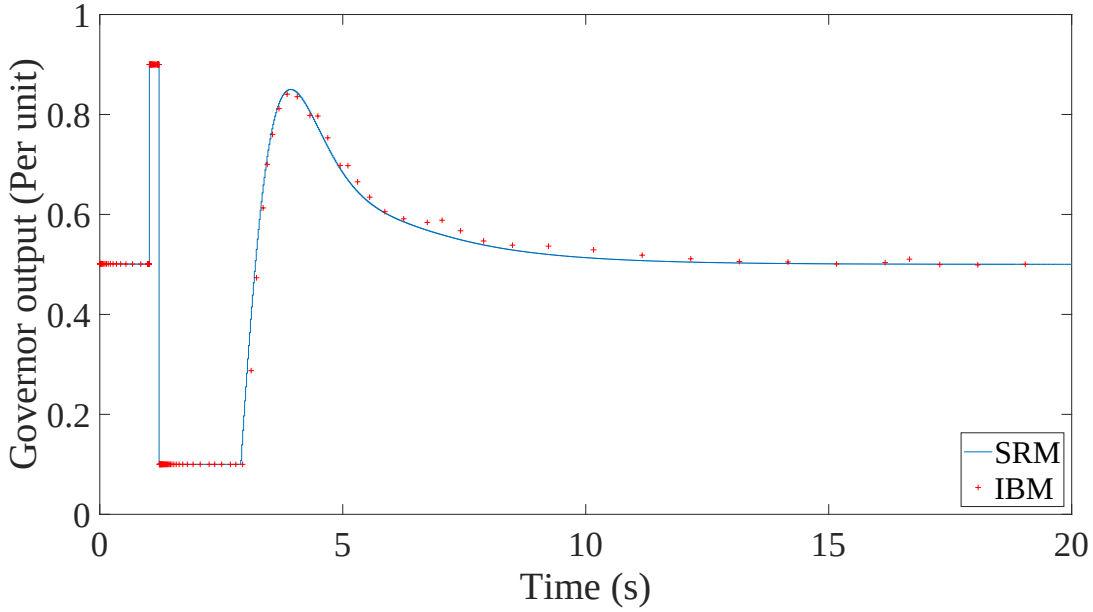

Figure 4.20: Output of the digital governor of the generator of SMIB system

In addition, as can be seen in Fig. 4.20, the governor reaches its upper and lower limits equal to 0.9 and 0.1, showing the non-linear nature of the simulated model. This also shows the ability of IBM to catch the state events caused by the controller reaching its limits without the need to reduce the step size. As discussed in section 4.2.6, it is achieved by the fact that the state events translate to time events since they are not seen by the digital controller before its next sampling. The same phenomenon happens for SRM as well, since all the controllers are digital. However, SRM handles the state event by a forced reduced time step.

The number of Newton iterations and the execution time required to solve the described system are

Table 4.2: Performance comparison between IBM, and SRM for the SMIB system

Method	Number of Newton iterations	Run time (s)
SRM	25491	9.35
IBM	483	1.90

summarized in Table 4.2. It can be seen that IBM is about 52 and 5 times faster than traditional SRM in terms of Newton iterations and execution time, respectively. The step sizes taken to simulate the SMIB system are shown in Fig. 4.21, justifying the accelerated execution time.

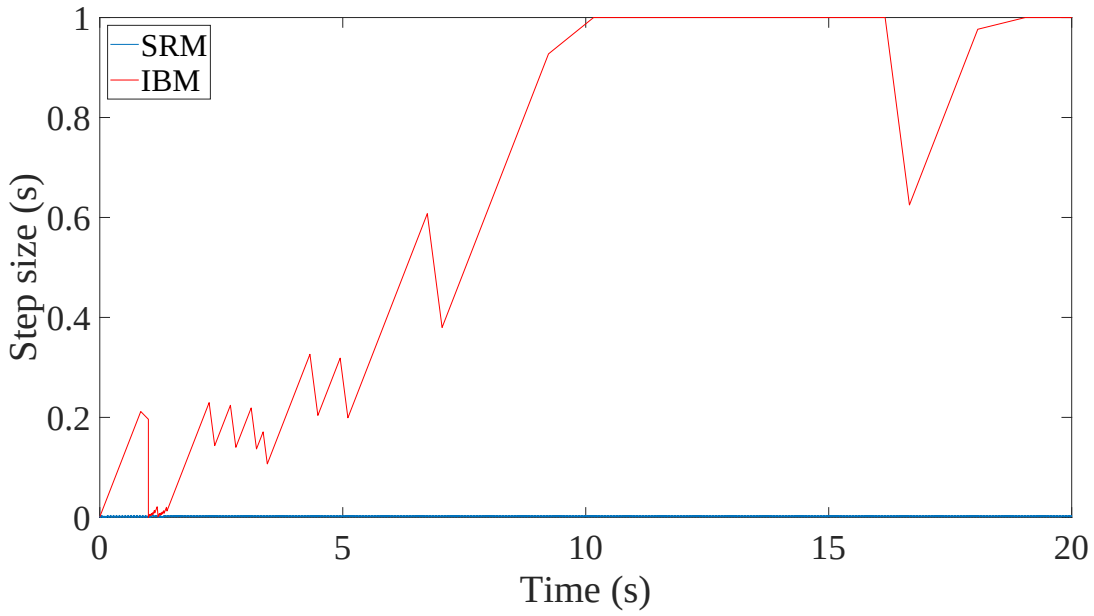

Figure 4.21: step size results for SMIB system

4.3.1.3 Modified Kundur system

In this section, a modified version of the Kundur test network [61], illustrated in Fig. 4.22 alongside its initial power flows, consisting of four synchronous generators and two loads, is considered as the test case. In addition, an HVDC link parallel with the tie between the two areas of the system is considered. Similar to the previous case study, each generator is controlled by a digital AVR [62] and a Governor [63]. While the AVR model used for generators 1 and 2 is considered equation-based, the non-equation-based fuzzy AVR is utilized for generators 3 and 4. So, the system has 4 digital governors, 4 digital exciters for the generators, and a power oscillations damping (POD) controller for the HVDC link [64], therefore, 9 controllers in total. The sampling time of digital governors and digital exciters for generators 1 to 4 is considered 20 ms, 30 ms, 40 ms, 50 ms, and 5 ms, 6 ms, 7 ms, 8 ms, respectively, and 90 ms for the POD controller. It should be noted that the sampling times were selected slightly differently and asynchronously to showcase the proposed method in a more realistic setting. The DAEs describing the generator and the schematics of all the controllers are presented in Appendix I.

A short circuit with a low impedance on bus 3 for 0.2 s is simulated as the disturbance. The voltage, speed deviation, AVR, governor, and POD output of the first generator are illustrated in Figs. 4.23, 4.24,

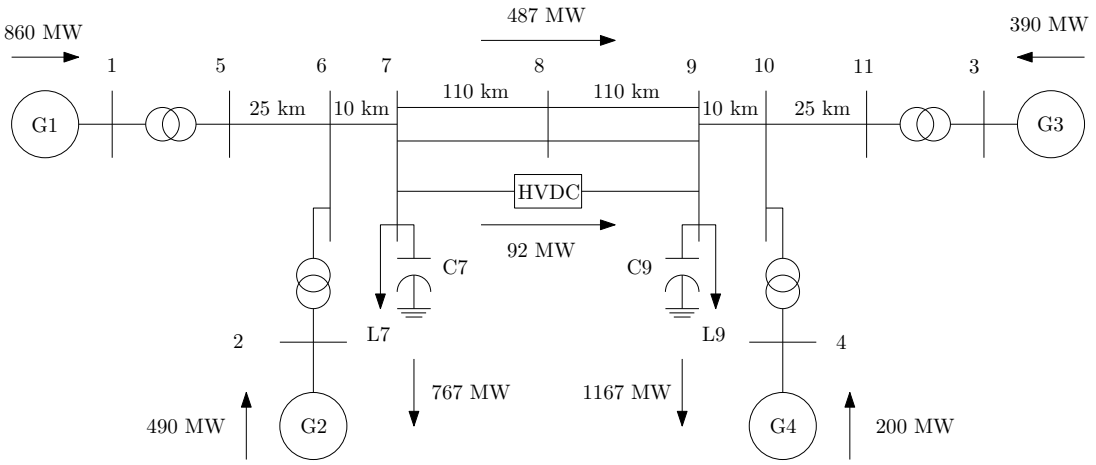

Figure 4.22: Schematic of modified Kundur network

4.25, 4.26, and 4.27 respectively. It is shown that the simulated results are almost identical between SRM and IBM.

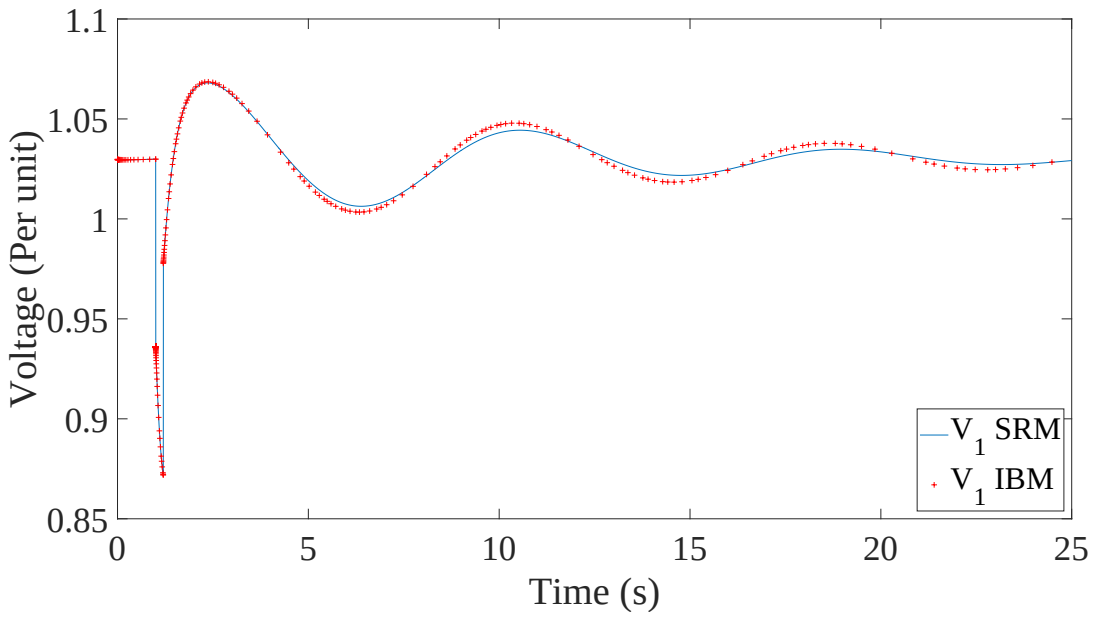

Figure 4.23: Bus 1 voltage magnitude of the Kundur system

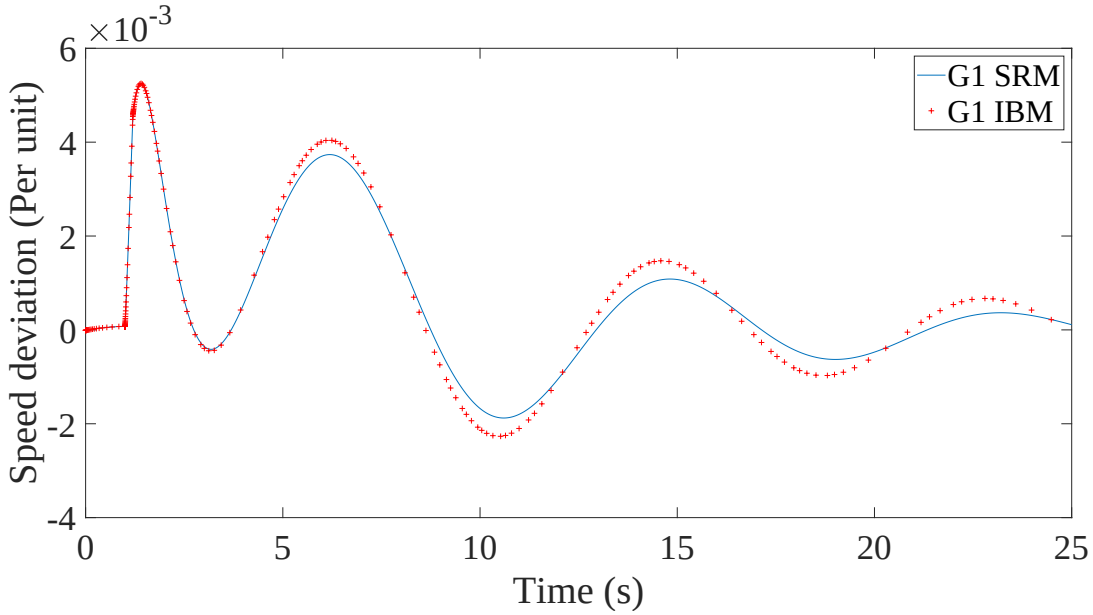

Figure 4.24: Speed deviation of generator 1 of the Kundur system

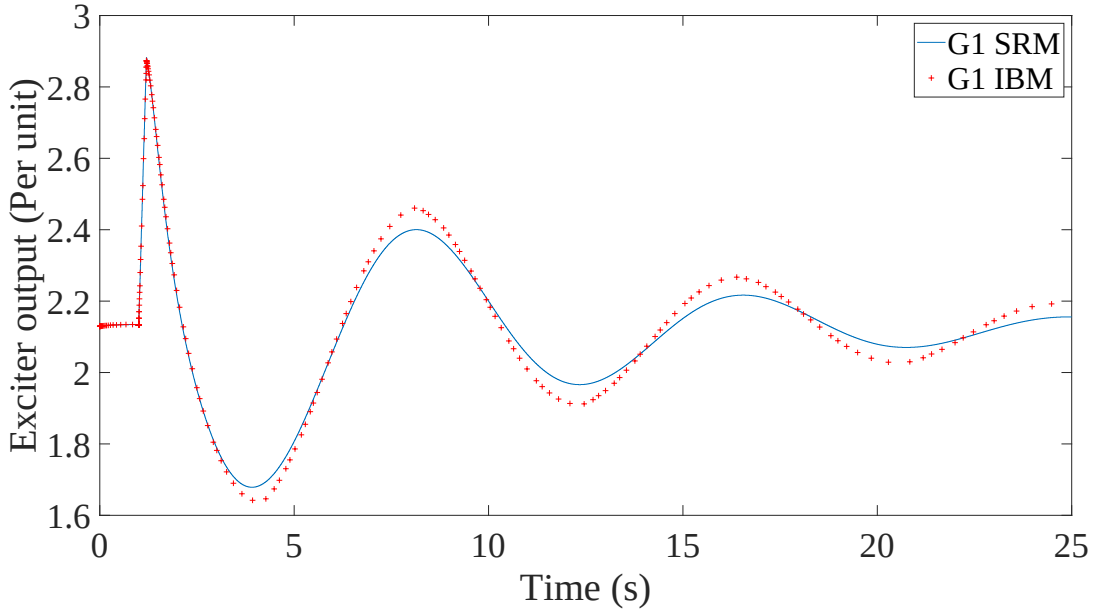

Figure 4.25: Output of the digital excitation system of generator 1 of the Kundur system

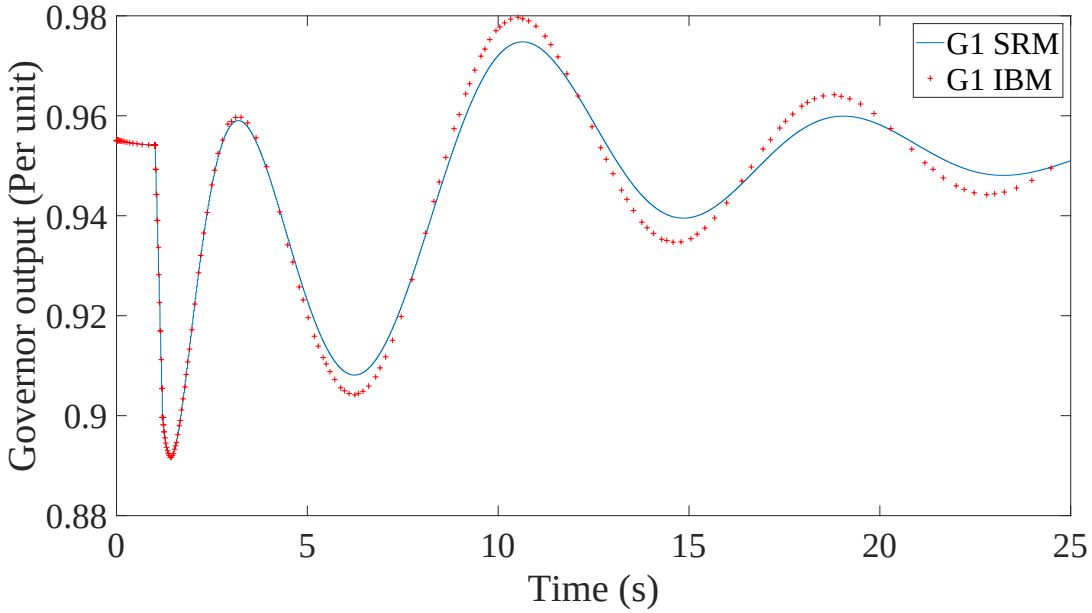

Figure 4.26: Output of the digital governor of generator 1 of the Kundur system

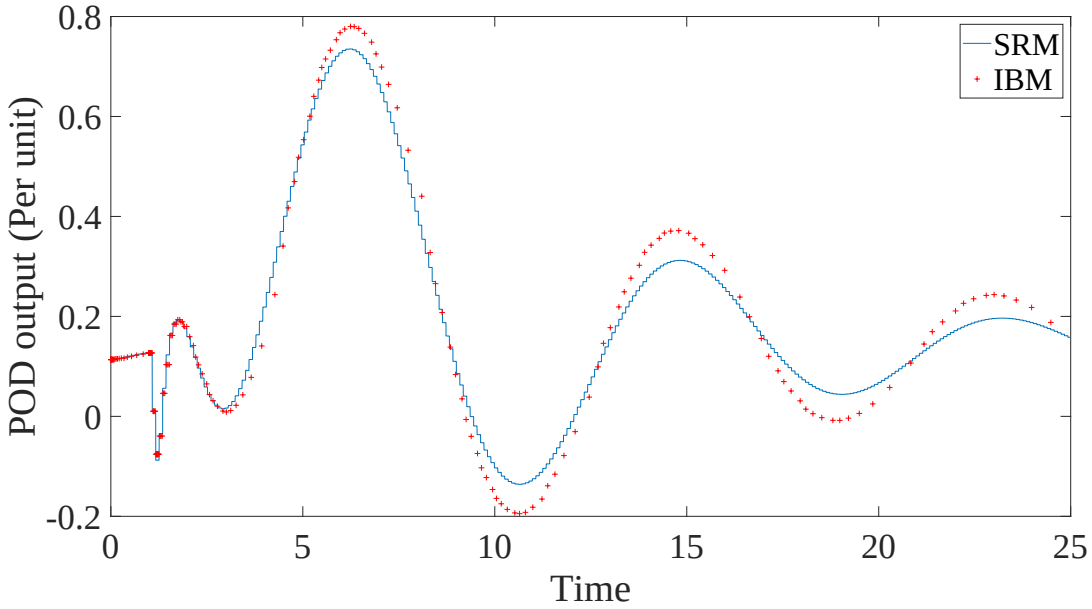

Figure 4.27: Output of the digital POD of the Kundur system

The performance of the methods is compared using the number of Newton iterations and the execution time required to solve the Kundur test network, and the results are summarized in Table 4.3. Again, it can be seen that IBM is much faster than SRM (about 16 times), as justified by the time-step size shown in Fig. 4.28. Also, by comparing all three case studies, it can be noted that, as the number of digital controllers of the system under simulation increases, SRM becomes slower at a much faster rate than IBM. The reason is that as the number of controllers grows, the difference between two adjacent sampling times becomes smaller and limits the step size further, while this is not the case in IBM.

Table 4.3: Performance comparison between IBM, and SRM for Kundur test system

Method	Number of Newton iterations	Run time (s)
SRM	64158	327.26
IBM	959	19.54

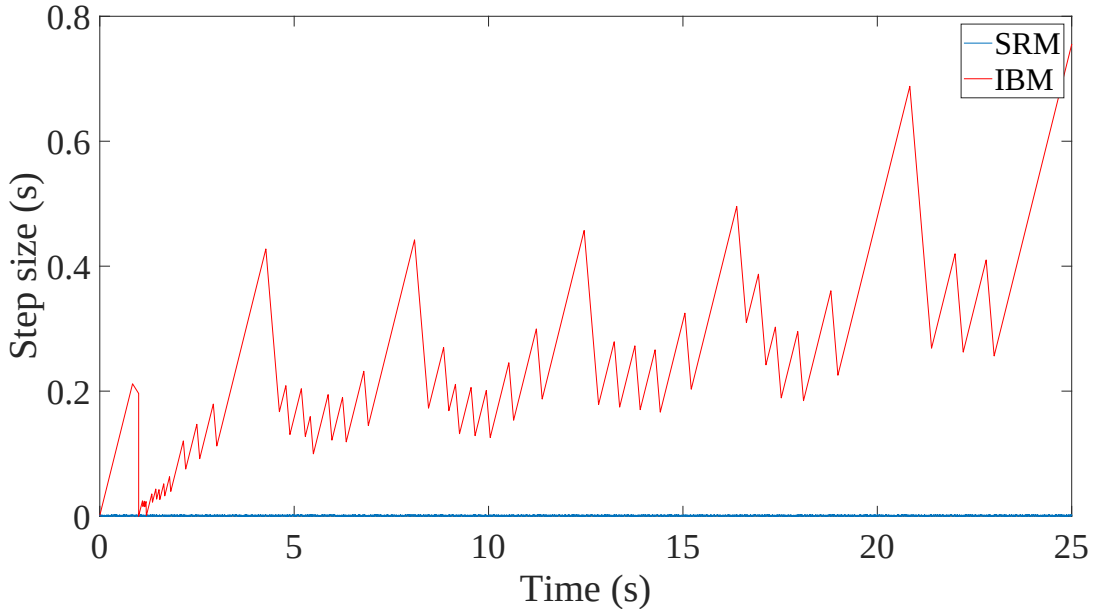

Figure 4.28: Step size results for the Kundur system

4.3.2 DIBM

Similar to the previous section, the same three case studies are used to showcase the accuracy and performance of DIBM compared to the original IBM and SRM. The only difference is that the third case study, the Kundur test system, lacks the added HVDC link and the associated POD controller, hence, it has 8 digital controllers. Also, all the controllers used are equation-based, which are presented in Appendix I. Therefore, the case studies in this section can be listed as follows:

- A general system consisting of a two-state continuous ODE system controlled by a digital integral controller.
- A single-machine infinite-bus (SMIB) system containing a synchronous generator connected to an infinite bus under control by two digital controllers: a governor and an exciter.
- The well-known Kundur test system, which has 4 synchronous generators and eight digital controllers in total (two for each generator.

4.3.2.1 A single integral controller

The trajectories of the system's second output, simulated over 50s, are presented in Fig. 4.29 and Fig. 4.30 for both stable and unstable cases (corresponding to different values of a and b), using all three methods: SRM, IBM, and DIBM. As illustrated, all three methods exhibit comparable accuracy in this case study, as was expected.

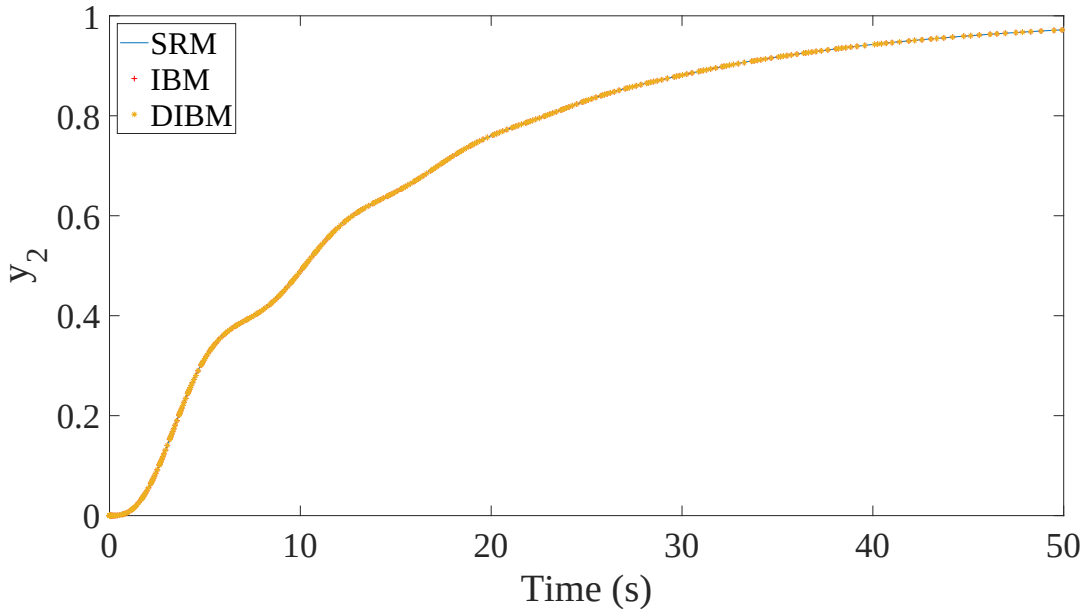

Figure 4.29: Output results for the stable system controlled by an integral controller

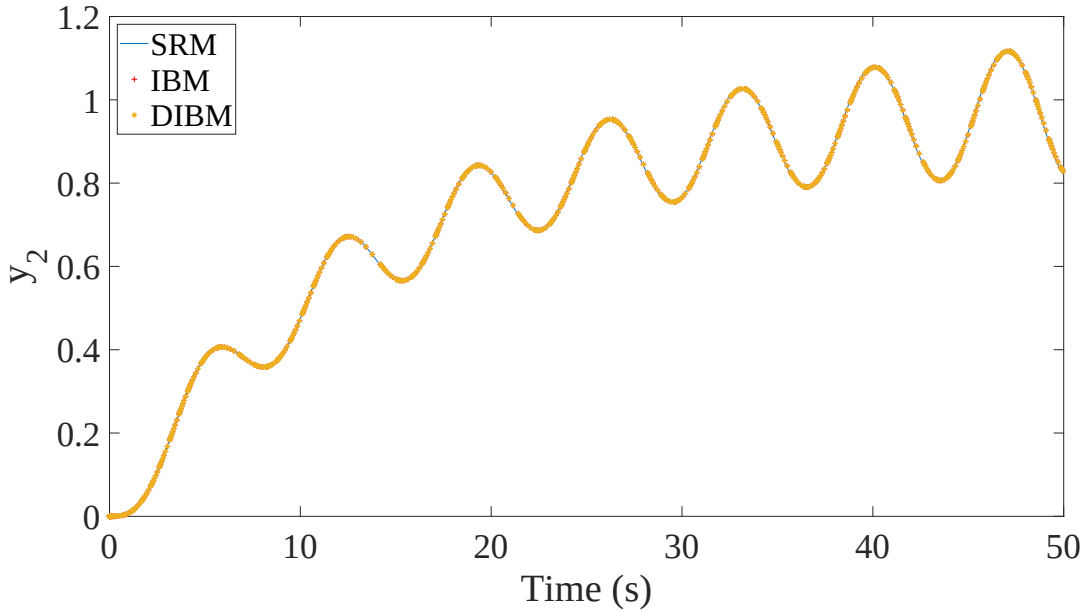

Figure 4.30: Output results for the unstable system controlled by an integral controller

Furthermore, the performance of all three methods is showcased in Table 4.4 in terms of the total number of Newton iterations for all time steps. It can be seen that the IBM-based methods are computationally faster than the SRM due to the time-step reduction. Comparing IBM with DIBM, the unstable case simulation shows no difference, while the simulation for the stable case is slightly faster. This is also justified by the time steps taken that are depicted in Fig. 4.31 and Fig. 4.32 for the stable and unstable scenarios, showing similar performance for both IBM and DIBM.

Table 4.4: Performance result of the first case study simulations in terms of number of Newton iterations using SRM, IBM, and DIBM

System state	SRM	IBM	DIBM
Stable	4106	1071	1061
Unstable	3999	1758	1758

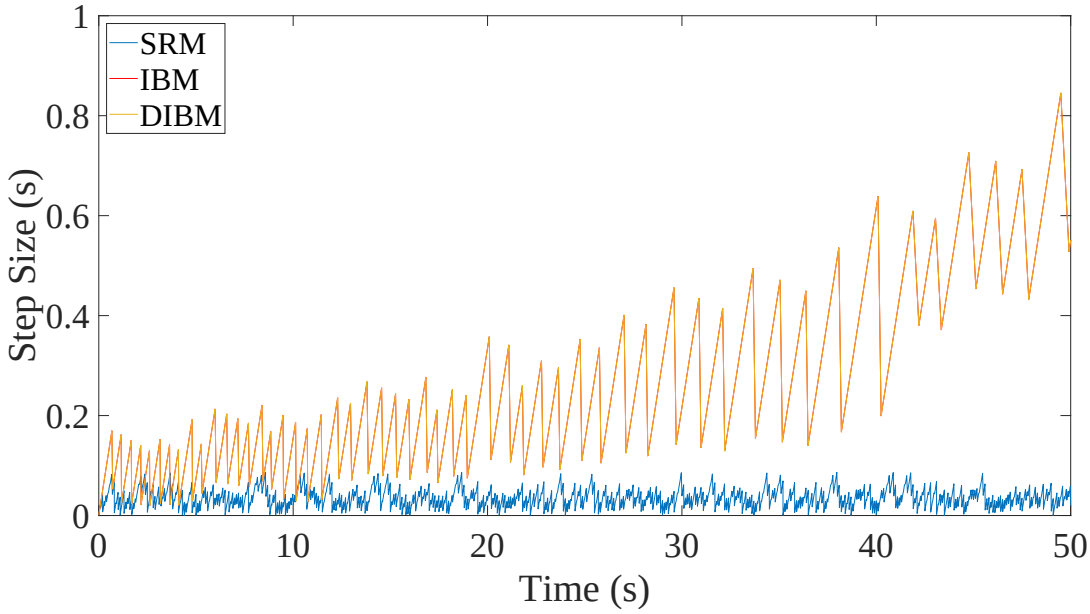

Figure 4.31: Step size results for the stable system with the Integral controller

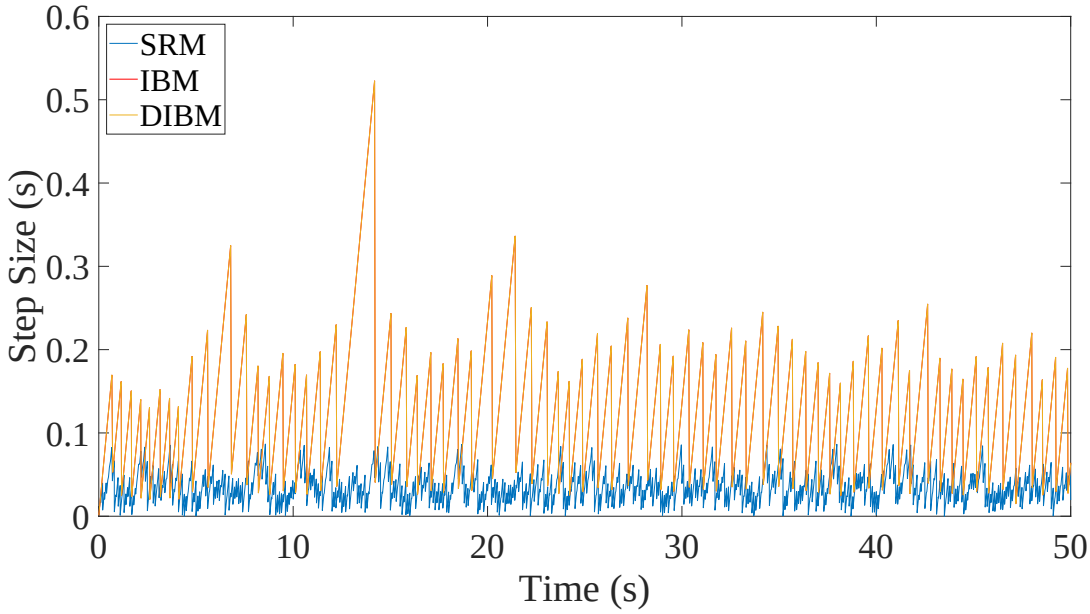

Figure 4.32: Step size results for the unstable system with the Integral controller

4.3.2.2 Single machine model with excitation system and governor

This test system consists of a synchronous generator connected to an infinite bus using a transmission line, as shown in Fig. 4.13. The generator is controlled by a digital governor with a sampling period $T_G = 0.2$ s and a digital exciter with a sampling period $T_G = 0.4$ s. The block diagrams are shown in I.

A short circuit at $t = 1$ s is applied to the infinite bus for 200 ms. The generator voltage, active power, reactive power, the governor output, and the exciter output are shown in Figs. 4.33, 4.34, 4.35, 4.36, and 4.37. Again, the same level of accuracy can be observed for all three methods.

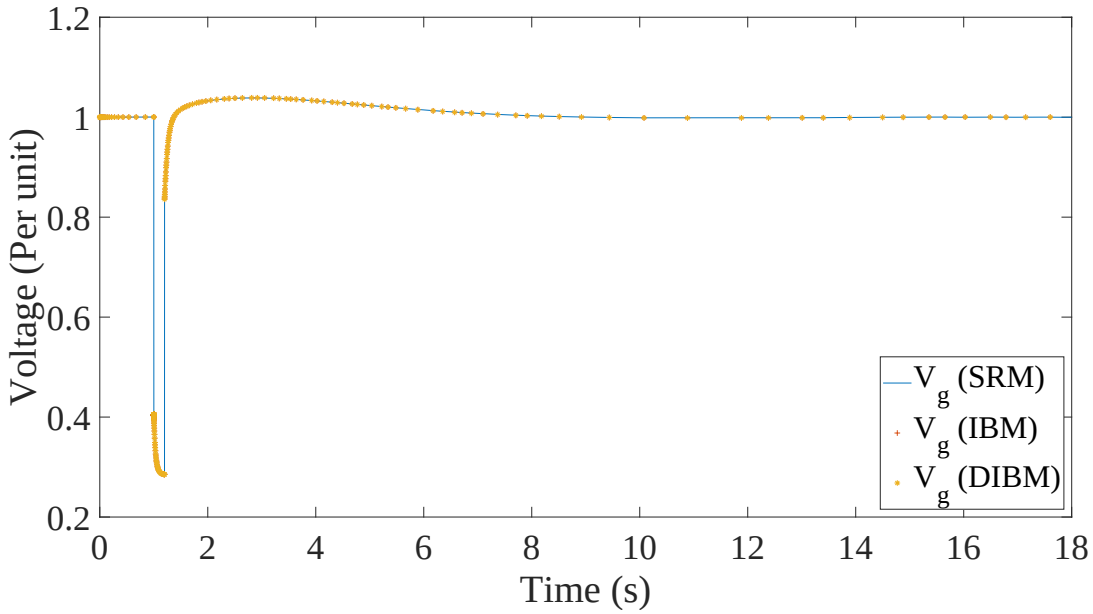

Figure 4.33: Generator voltage output of SMIB system

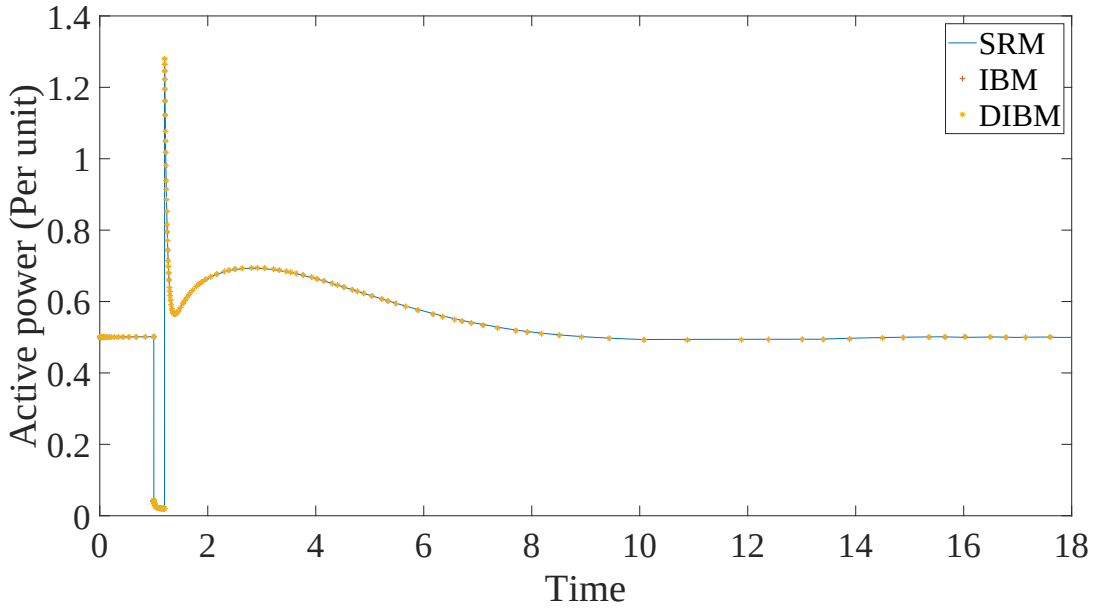

Figure 4.34: Active power output of SMIB system

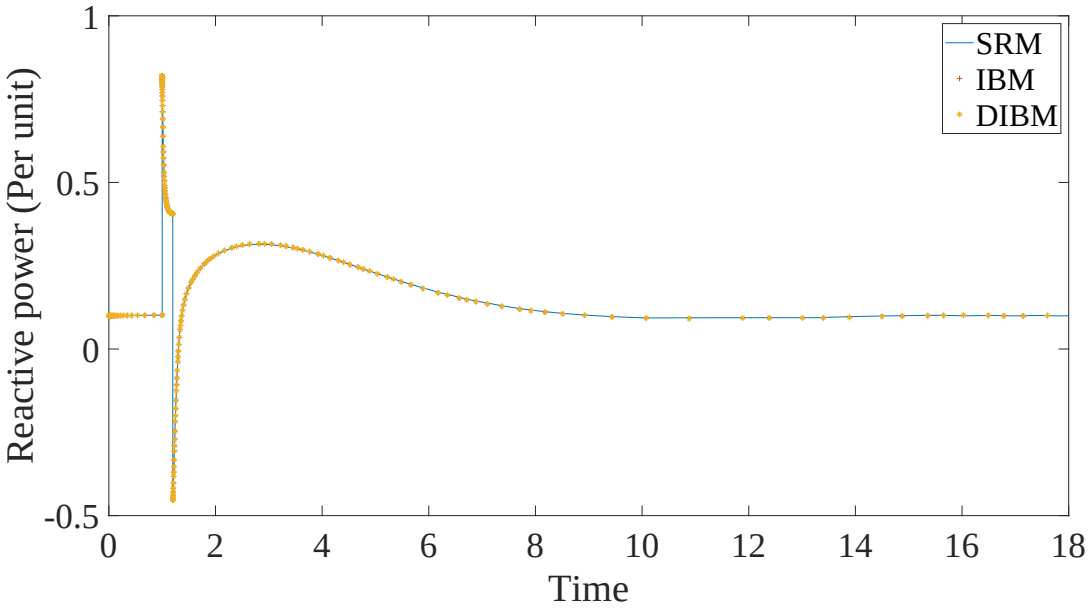

Figure 4.35: Reactive power output of SMIB system

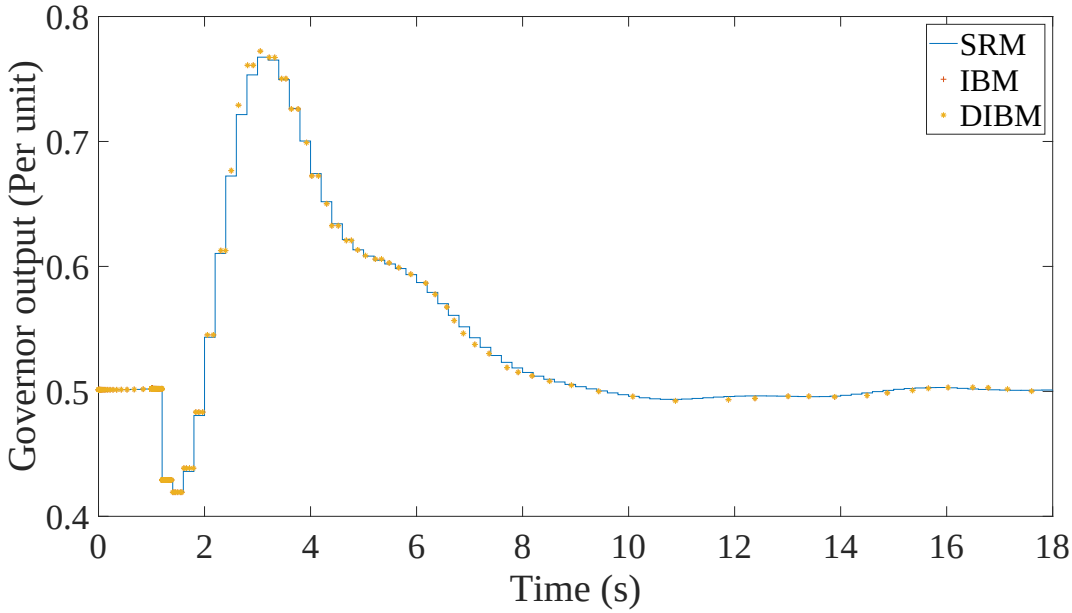

Figure 4.36: Governor output of SMIB system

Table 4.5: Performance result of the SMIB system simulations in terms of number of Newton iterations and run time using SRM, IBM, and DIBM

Method	Nb. Newton iterations	Average of 5 executions (s)
SRM	4801	4.40
IBM	643	0.95
DIBM	624	0.88

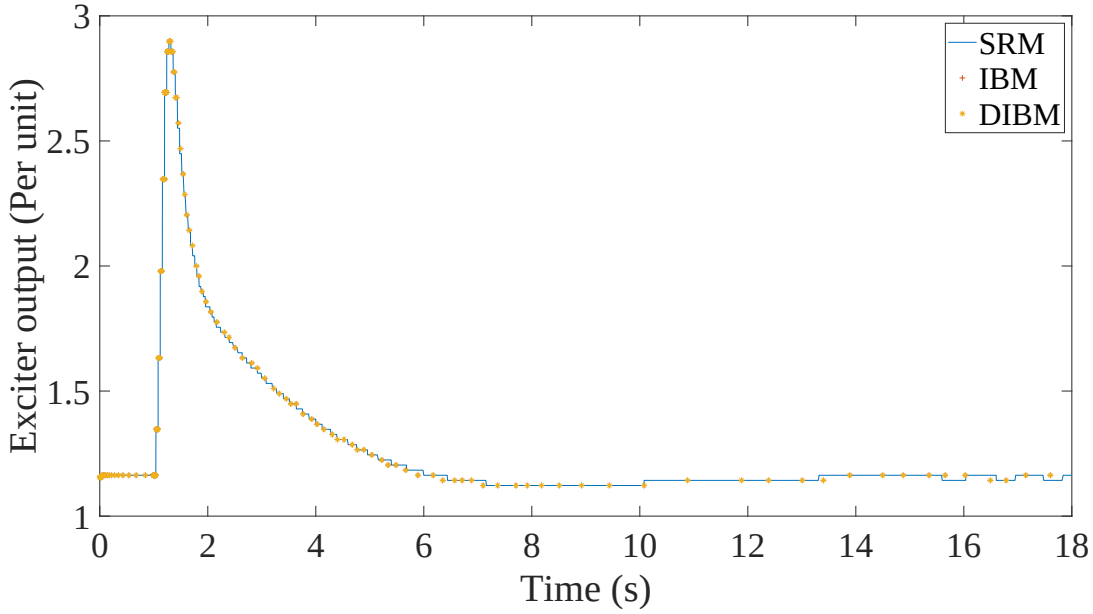

Figure 4.37: Exciter output of SMIB system

The performance comparison is shown in Table 4.5, listing the total number of Newton iterations and the execution time. The reported time is the average of 5 runs of the simulation to make sure of the validity of the numbers. Again, SRM shows clearly the least performance, while DIBM is slightly faster than IBM. The reason is that the controller outputs are calculated in each iteration with the updated state variables, which leads to faster convergence for some time steps. This can also be justified using the time steps taken, depicted in Fig. 4.38, showing similar time step sizes for IBM and DIBM.

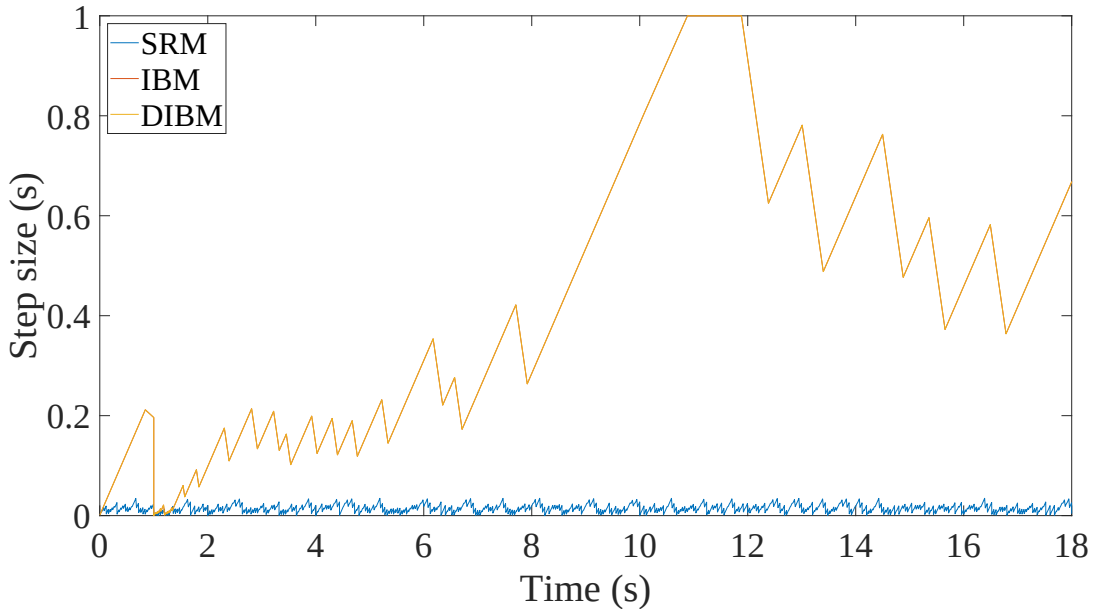

Figure 4.38: Step size results for SMIB system

4.3.2.3 Kundur system

For the last case study, the well-known Kundur system is considered. The sampling periods are equal to 210, 220, 230, and 240 ms for the governors, and 41, 42, 43, and 44 ms for the exciters, respectively.

A load reduction of 250 MW in L7 is simulated at $t = 1$ s and then restored at $t = 12$ s. The simulation results for the voltage of buses 1, 4, and 9, the governor outputs, and the exciter outputs are illustrated in Figs. 4.39, 4.40, and 4.41.

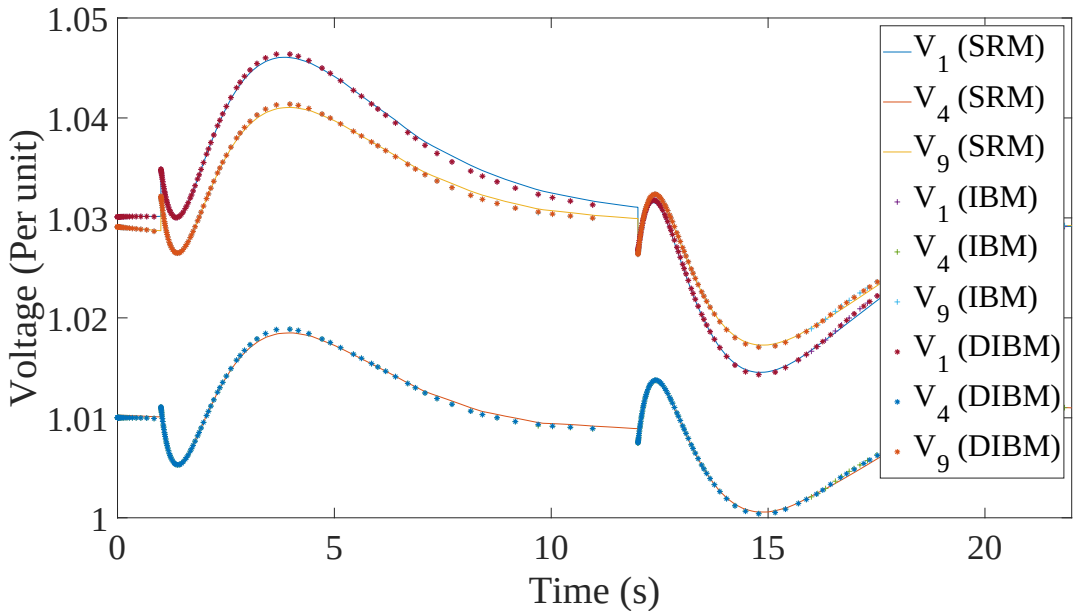

Figure 4.39: Voltage of bus 1, 4, and 9 of Kundur system

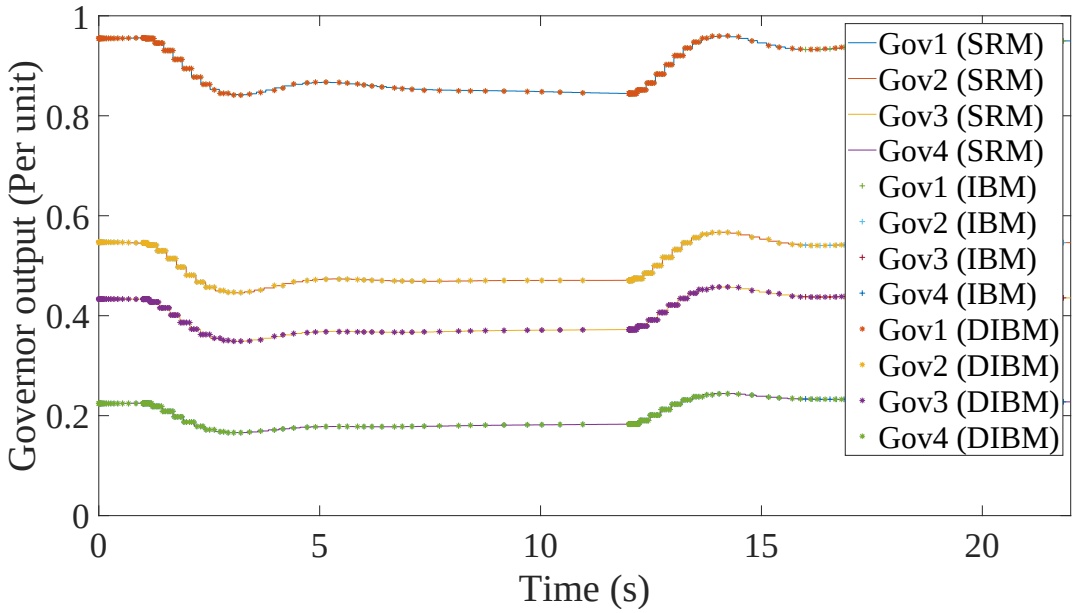

Figure 4.40: Governor outputs of Kundur system

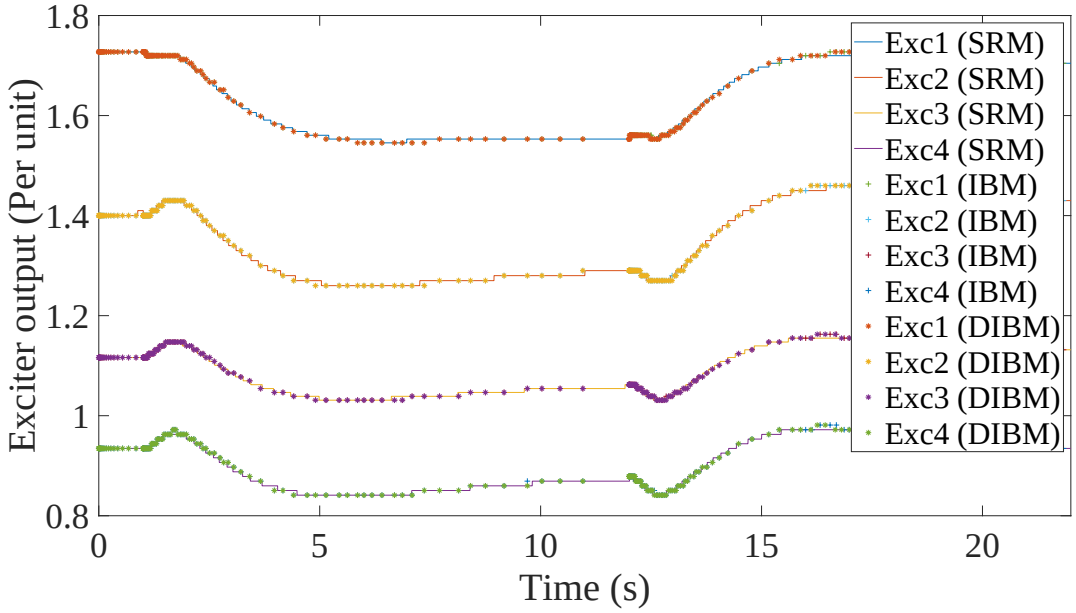

Figure 4.41: Exciter outputs of Kundur system

Once more, the trajectories show the same accuracy between all three methods. However, the performance results listed in Table 4.6 demonstrate that the number of iterations for DIBM is slightly less than IBM, consequently, the execution time for DIBM is smaller. Finally, the time steps taken by all three methods are shown in Fig. 4.42, supporting the performance results.

Table 4.6: Performance result of the third case study simulations in terms of number of Newton iterations and run time using SRM, IBM, and DIBM

Method	Nb. Newton iterations	Average of 5 executions (s)
SRM	11927	42.48
IBM	604	2.54
DIBM	598	2.38

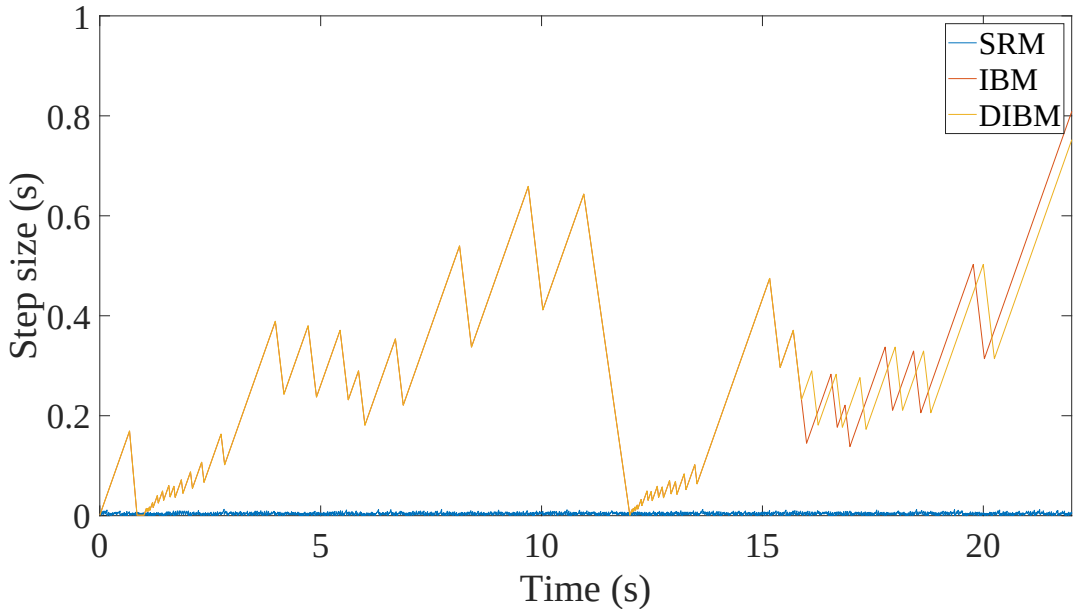

Figure 4.42: Step size results for Kundur system

4.3.3 Methods comparison

In this section, the proposed IBM is compared to SRM, ATM, and SSM regarding accuracy and performance.

Two case studies are used for this part of the simulation studies:

- The two-state system under control by the integral controller.
- The kundur system.

All the parameters and details of the case studies are discussed in the previous section, so they are neglected here.

4.3.3.1 A single integral controller

The output of the system y_2 is illustrated for stable and unstable scenarios (different b values) in Figs. 4.43 and 4.44, respectively. It should be pointed out that only IBM-AB is used for this case study.

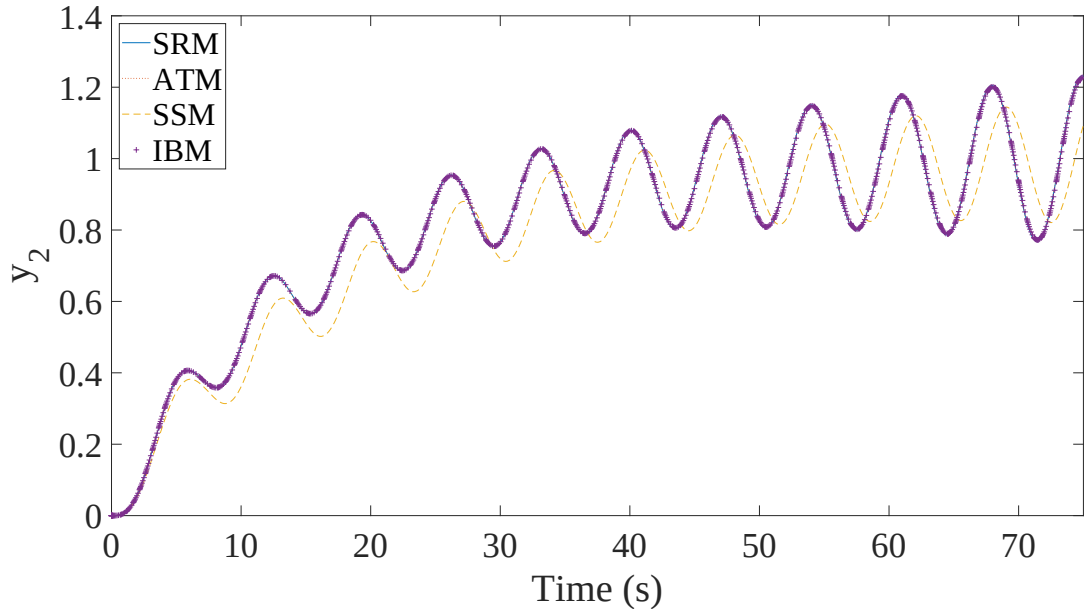

Figure 4.43: Simulation results for the methods SRM, ATM, SSM, and IBM for the stable system controlled by the integral controller

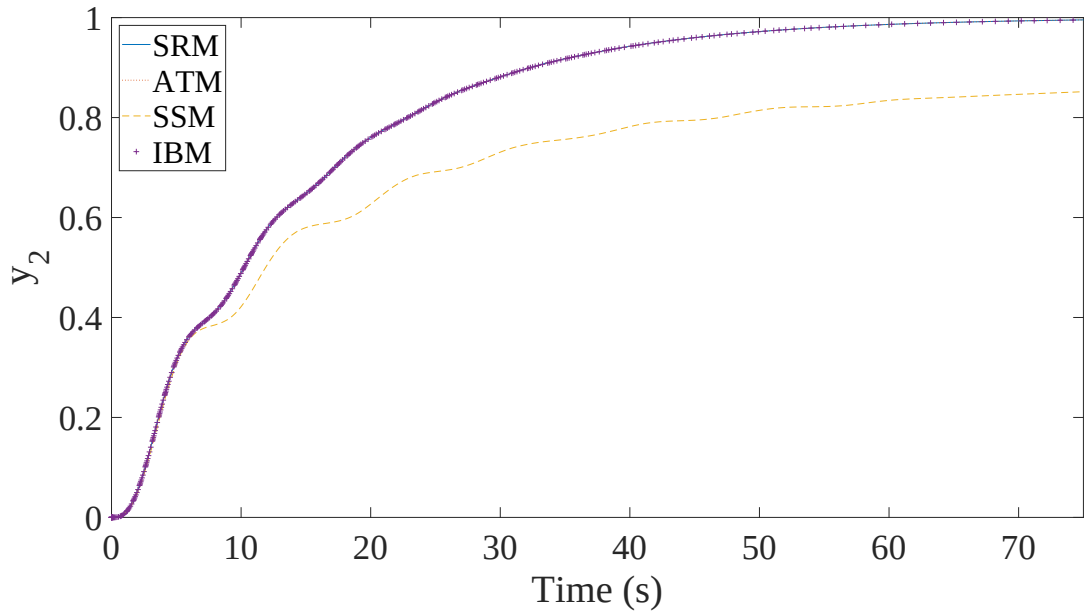

Figure 4.44: Simulation results for the methods SRM, ATM, SSM, and IBM for the unstable system controlled by the integral controller

As can be seen, the response of SRM, ATM, and IBM is almost identical, with SSM being the most inaccurate. The reason is that SSM cannot handle multiple time events in one time step. In other words, it can only shift one time event to the end of the time step and discards the rest since each has its unique feedback that SSM has no access to.

Figures 4.45 and 4.46 show the time steps taken for the stable and unstable systems, respectively. It can be seen that SRM limits the time steps to the difference between controller sampling times, while other methods are able to increase the time step size with no limitation. In addition, it is noticeable that

ATM has the fastest time step increase. Furthermore, the number of Newton iterations for solving the stable and unstable systems and for all the methods is listed in Table 4.7

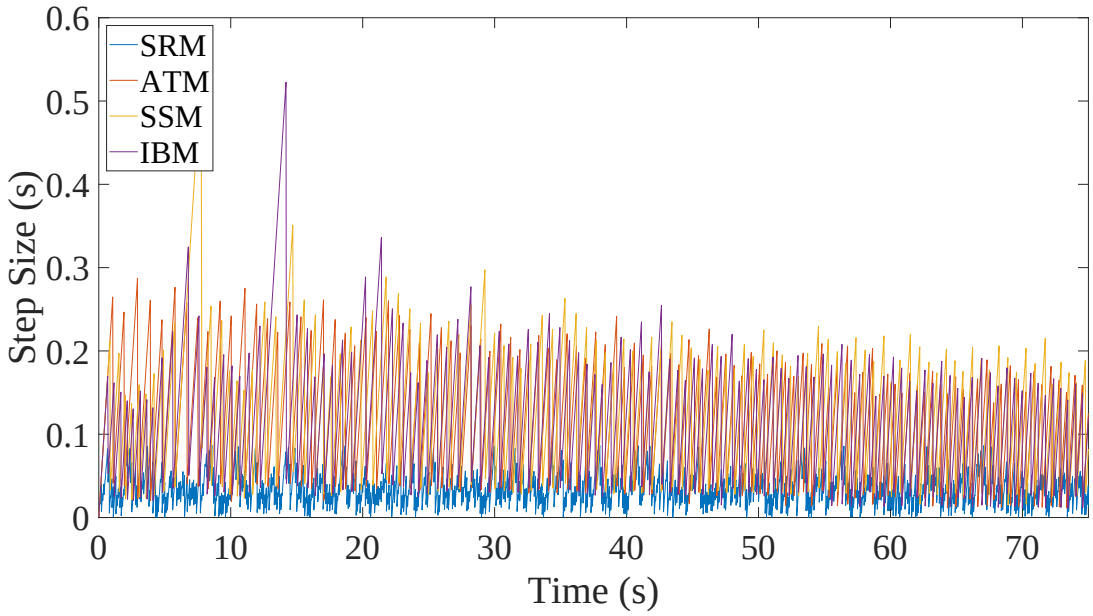

Figure 4.45: Step size results for the methods SRM, ATM, SSM, and IBM for the stable system with the Integral controller

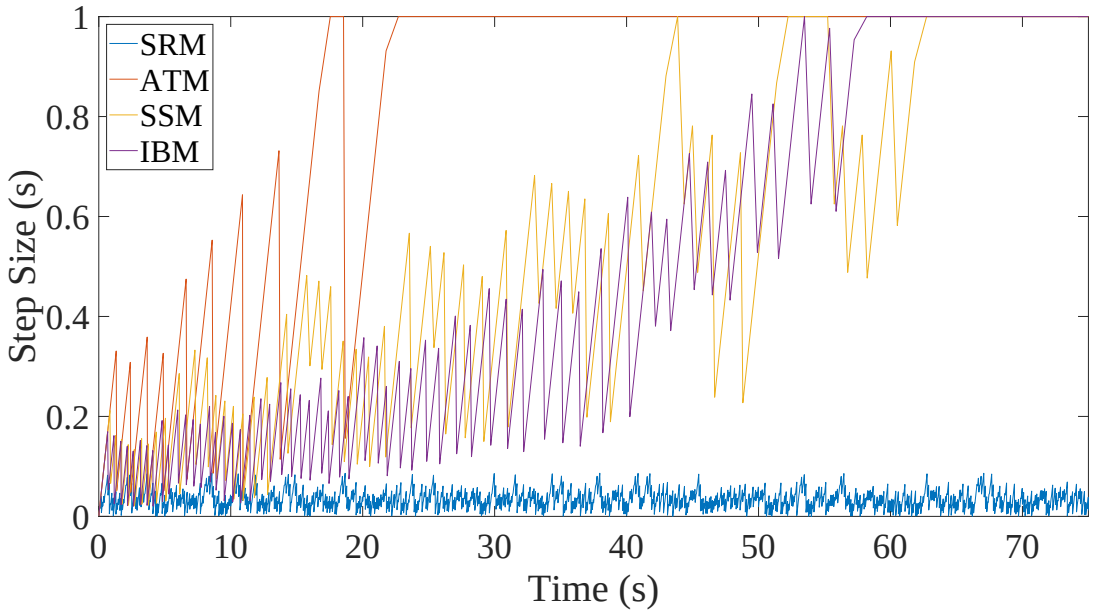

Figure 4.46: Step size results for the methods SRM, ATM, SSM, and IBM for the unstable system with the Integral controller

Table 4.7: Performance comparison between SRM, ATM, SSM, and IBM-AB for a system containing one Integral controller in terms of the number of Newton iterations

Method	SRM	ATM	SSM	IBM-**AB**
Stable case	6338	336	830	1180
Unstable case	6151	2497	2561	2895

4.3.3.2 Kundur system

As mentioned before, the system has 8 digital controllers in total. The sampling time T of the digital controllers is set to 210 ms, 220 ms, 230 ms, and 240 ms for the governors, and 41 ms, 42 ms, 43 ms, and 44 ms for the exciters.

A short circuit on bus 3 for 200 ms is simulated using all four methods. The simulation outputs are shown in Figs. 4.47, 4.48, 4.49, and 4.50, for the voltage of bus 1, the speed deviation of the third generator, the governor output of the third generator, and the exciter output of the third generator, respectively. An upper limit equal to 2 per unit is considered for the third exciter to assess the accuracy of the methods while facing non-linearity in the controllers, and the results are illustrated in Fig. 4.50.

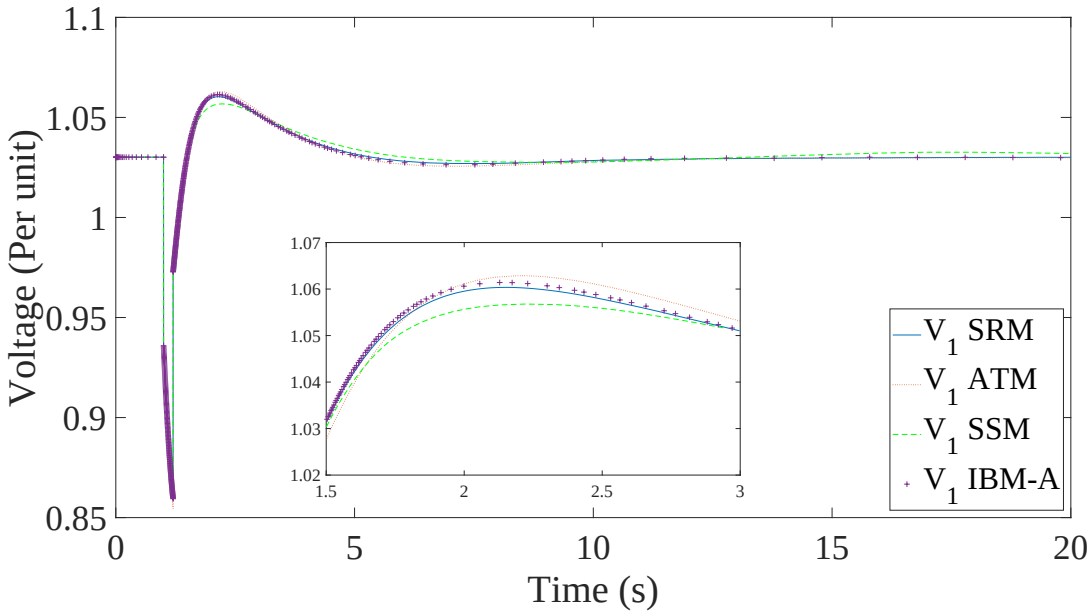

Figure 4.47: Voltage of bus 1 of the Kundur test system

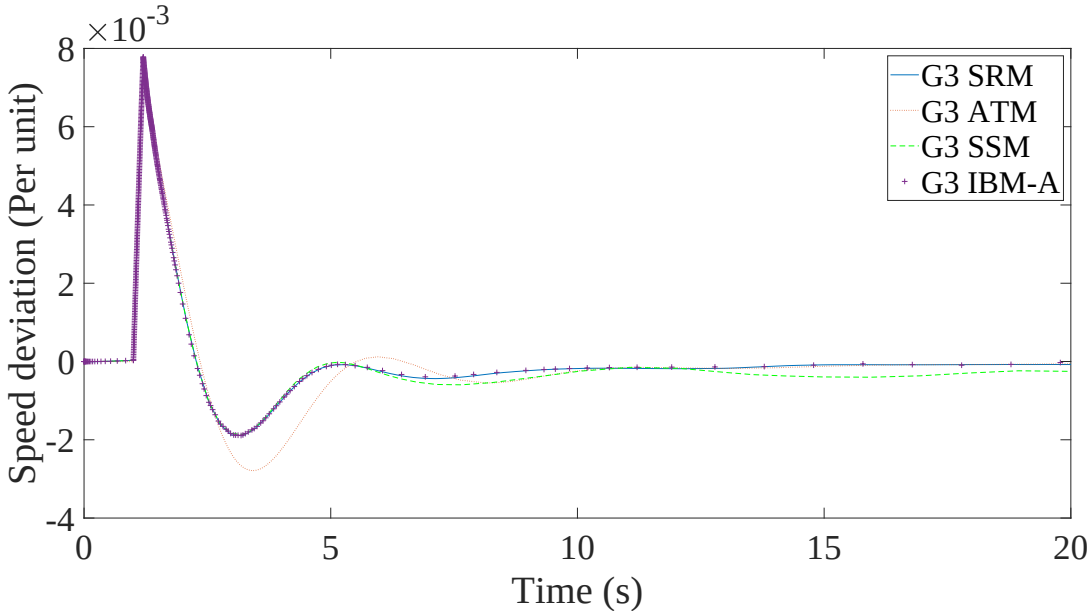

Figure 4.48: Speed deviation of the generator 3 of the Kundur test system

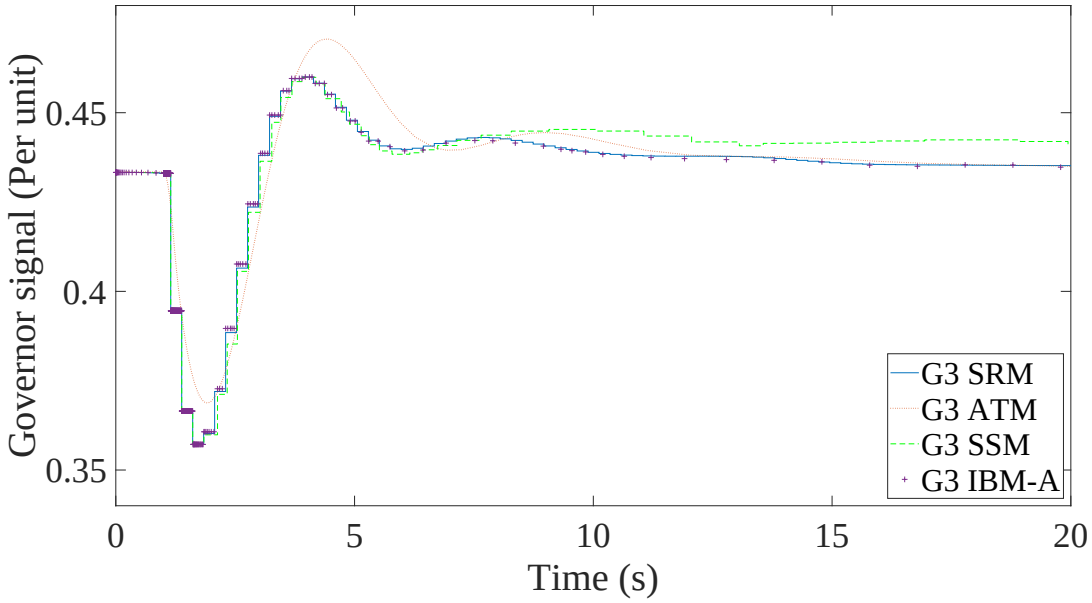

Figure 4.49: Governor output of the generator 3 of the Kundur test system

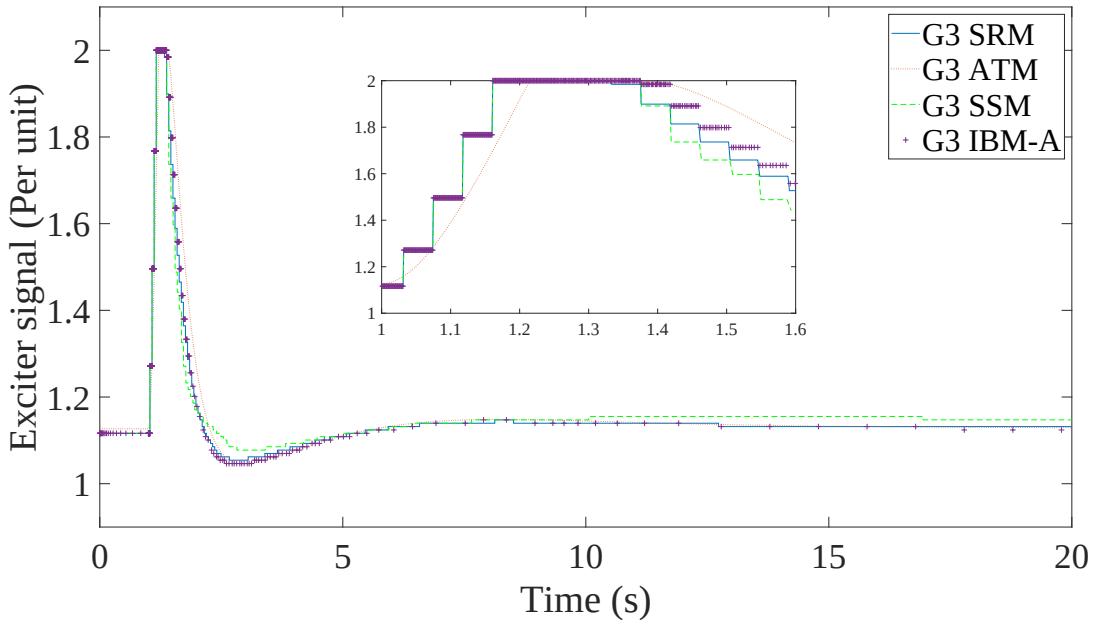

Figure 4.50: Exciter output of the generator 3 of the Kundur test system

As can be seen, ATM has the worst accuracy since it ignores the digital nature of the controller. This is also reflected in Table 4.8, which lists the Euclidean distance of the bus 1 voltage and speed deviation of the third generator with respect to SRM as the reference trajectory. It is also not the fastest method anymore, which is reflected in Fig. 4.51 that shows the time steps taken for all four methods. Furthermore, it can also be seen in Table 4.9, which summarizes the performance in terms of the number of Newton iterations, the number of function evaluations, and the average execution time of five repetitive simulations. The reason that ATM is not the fastest method anymore is that the size of the system to be solved is increased by the number of controllers' equations, while SSM solves a smaller system for each time step.

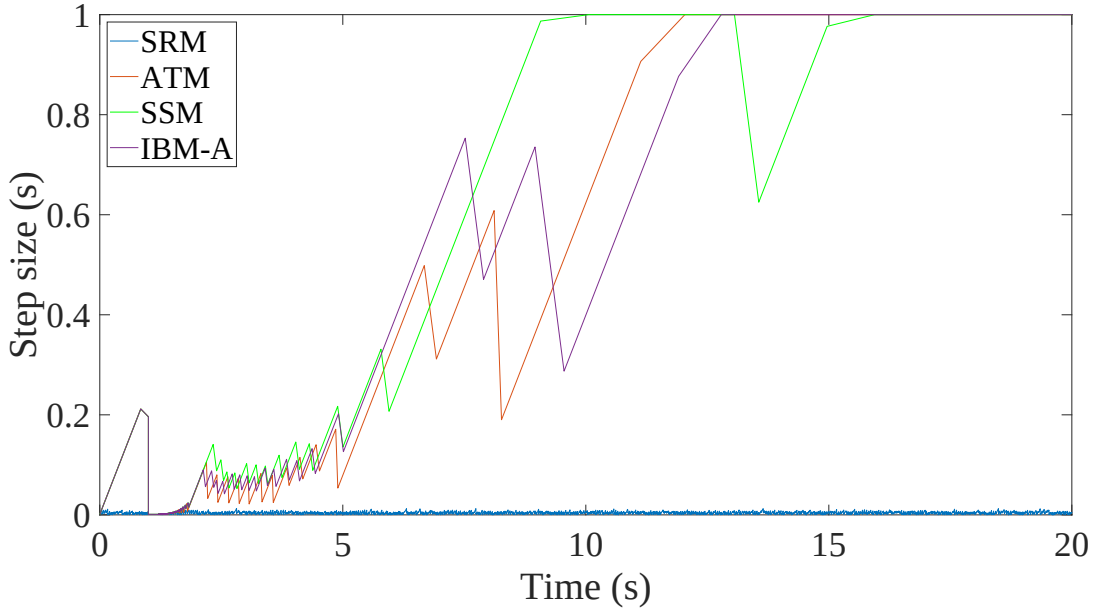

Figure 4.51: Step size results for the methods SRM, ATM, SSM, and IBM for the Kundur test system

Table 4.8: Accuracy comparison between ATM, SSM, and IBM with respect to SRM for Kundur system using Euclidean distance

Method	ATM	SSM	IBM-**A**
$\epsilon(V_1)$	0.1359	0.1160	0.1138
ϵ(Speed deviation 3)	0.007697	0.0009768	0.0004236

Table 4.9: Performance comparison between SRM, ATM, SSM, and IBM-A for Kundur system in terms of the number of Newton iterations, function evaluations, and runtime

Method	SRM	ATM	SSM	IBM-**A**
N. Newton iterations	9633	1255	1160	1241
N. function evaluations	1782105	292415	214600	229585
Average runtime (s)	33.01	6.21	3.88	4.45

4.3.4 Jacobian comparison

The impact of different Jacobians of IBM on performance is discussed in this section. The same simulation using the Kundur system is repeated for the IBM Jacobian variation methods. The accuracy and performance results are listed in Table 4.10 and Table 4.11, respectively.

Table 4.10: Accuracy comparison between IBM-A, IBM-AB, IBM-AC, and IBM-ABC with respect to SRM for Kundur system using Euclidean distance

Method	IBM-A	IBM-AB	IBM-AC	IBM-ABC
$\epsilon(V_1)$	0.113806	0.113755	0.113805	0.113754

Table 4.11: Performance comparison between IBM variations for Kundur system in terms of the number of Newton iterations, function evaluations, and runtime

Method	IBM-A	IBM-AB	IBM-AC	IBM-ABC
N. Newton iterations	1241	1209	1233	1206
N. function evaluations	229585	238121	393337	392652
Average runtime (s)	4.45	5.41	27.47	24.89

As the sub-matrixes are added, the number of Newton iterations decreases due to the more accurate Jacobian. However, the run time increases since they require many more function evaluations to be calculated. Comparing the runtime of IBM-AC and IBM-ABC with the IBM variations without the C sub-matrix, it can be noticed that the sub-matrix C calculation leads to a significant increase in function evaluations and consequently a performance drop. Among all the variations investigated, IBM-AB seems a reliable option both in terms of accuracy and performance.

4.4 Summary

In this section, 4 treatment methods capable of handling the time events of digital controllers were discussed. Each method has its weaknesses and strengths, either having good accuracy at the cost of performance loss or vice versa. Therefore, the interpolation-based method was proposed to fill the gap and have a fast yet accurate simulation of systems with digital controllers.

It was shown that for systems with many digital controllers, the analog treatment method, which is widely used today, is not always the fastest method (see Table 4.9), and it lacks accuracy (see Table 4.8). Also, it was shown that IBM has the highest accuracy relative to our reference method, SRM, while it has a performance similar to SSM, which is the fastest method. For instance, IBM is more than 16 times faster than SRM (see Table 4.3). Furthermore, several IBM variations based on simplified Jacobian matrix formations were introduced, and their accuracy and performance were compared, showing that perfecting the Jacobian may lead to a significant performance drop. However, it is shown that adding the sub-matrix B that takes into consideration the impact of the controller's output on the system's equations can be the best choice since it reduces the required number of Newton iterations by 2 percent while adding less than 1 second to the execution time (see Table 4.11).

5 Modifications and Extensions

In the previous section, the interpolation-based method was proposed for quickly and accurately simulating power systems with digital controllers. In this section, IBM is extended or modified to different versions, each suitable for specific challenges that may arise during the simulation of digital controllers in power system dynamics.

5.1 Simplified IBM

The SSM was presented and discussed in detail in Chapter 4. It is a fast method for obtaining quick system's dynamics. However, it suffered lack of accuracy, especially for systems with multiple digital controllers with relatively fast sampling rates. The reason is that SSM is developed for handling state events [65], and they rarely happen more than once in a time step. Therefore, while facing digital controllers, SSM shifts only one event to the end of the time step and discards the rest of them.

5.1.1 Simplified simulation challenge

To prevent issues that may occur while using SSM, some precautions are suggested[40]:

- It is essential to ensure that the flow is formulated in a way that is differentiable with respect to the state variables. If this condition is not satisfied, the dishonest Newton scheme may rely on an outdated Jacobian, potentially leading to divergence of the Newton iterations. In cases where the flow is non-differentiable, it must be reformulated as a combination of differentiable flows, accompanied by appropriate conditions that govern transitions between them.
- Since relatively large step sizes are used, the intermediate state variables produced during the simulation may deviate significantly from the true solution. Therefore, both the jump conditions and the flows must be formulated to robustly accommodate such deviations.
- SSM may result in cycling between flows. This issue can be addressed by appropriately modifying the state transition graph.
- Last but not least, a step reduction can be performed if none of the above suggestions are helpful to remove the simulation issue.

To better discuss the challenges of SSM for simulation of digital controllers, let's again consider a continuous system described by DAEs:

$$\begin{aligned} \dot{\boldsymbol{y}}(t) &= \boldsymbol{f}(\boldsymbol{y}(t), \boldsymbol{e}(t)) \\ \boldsymbol{y}(0) &= \boldsymbol{y_0},\ \boldsymbol{e}(0) = \boldsymbol{e}_0 \end{aligned} \tag{5.1}$$

which is under control by a digital controller described using difference equations:

$$\boldsymbol{e}_k = \zeta(\boldsymbol{e}_{k-1}, \boldsymbol{y}(kT)) \tag{5.2}$$

The reader may refer to Section 4.1 for details on variables.

Based on (5.2), there are mainly two reasons why SSM is not able to find the correct steady-state or follow fluctuations accurately. Fig. 5.1 shows a time step taken that has multiple time events from the operation of two digital controllers, indicated with different ink colors. First, SSM cannot consider multiple interconnected time events in one time step arising from the operation of a digital controller, since each controller output depends on the previous one. Furthermore, each controller output needs a unique feedback from the system in a specific sampling time, which is not available to SSM.

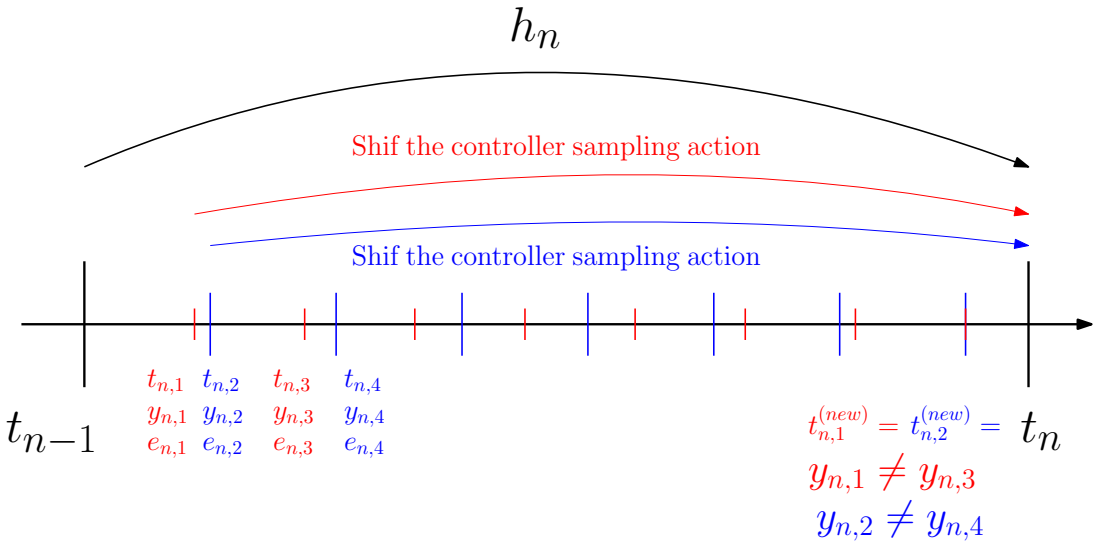

Figure 5.1: Time stepping using SSM in a simulation with two digital controllers with different sampling rates indicated by red and blue ink.

Therefore, there is a need for a simplified simulation method capable of simulating systems with digital controllers.

To address the issues mentioned above, the simplified IBM (SIBM) is developed with the aim of improving accuracy while maintaining the high performance of SSM. Therefore, the following modifications are introduced to SSM:

1. Considering (5.2), it can be seen that each controller output depends on the previous one. This means that the function ζ, which calculates the controller output, can be called recursively as many times as the controller samples in the time step h_n, instead of being called only once.
2. Instead of using the outdated feedback $y(kT)$ from the previous time step, an interpolation function can be used to estimate the feedback of the system at each sampling time, and then the interpolated feedback will be used in (5.2).

Fig. 5.2 shows the scheme SIBM for the simulation of a system with a digital controller over one time step. For each controller sampling action $t_{n,g}$, an interpolation formula $w_n(t_{n,g})$ is used to estimate the system's feedback at that sampling (blue dashed arrows):

$$\boldsymbol{x}_{n,g} = \boldsymbol{w}_n(t_{n,g}), \quad \forall g \in [1, p_n] \tag{5.3}$$

where $\boldsymbol{x}_{n,g}$ is a subset of system variables $\boldsymbol{y}_{n,g}$ under monitor by the controller, g is denoting the sampling

index, and p_n is the last sampling. Now the controller output can be calculated as (red solid arrows):

$$e_{n,g} = \zeta(e_{n,g-1}, \boldsymbol{x}_{n,g}), \quad \forall g \in [1, p_n] \tag{5.4}$$

Repeating this process for all the sampling in the time step recursively (dashed green arrow) results in the last controller output e_{n,p_n} which goes to the system as input to calculate $\boldsymbol{y}_{n+1}$.

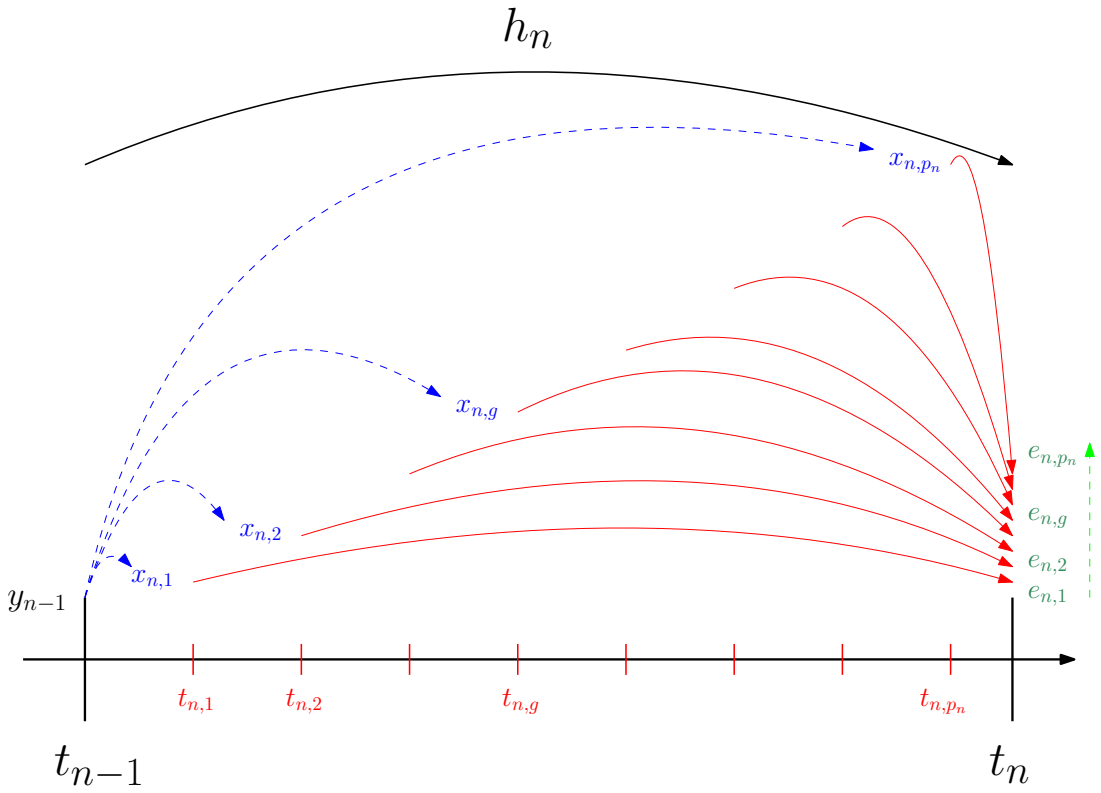

Figure 5.2: Time stepping using SIBM in a simulation with one digital controller. The blue ink indicates the interpolated variables. The red arrows indicate the sifting of samples. The green arrow shows the sequence of controller outputs being calculated.

5.1.2 Simulation results

In this section, a case study is considered to first showcase the challenge of SSM and its accuracy loss, then the accuracy and performance of SIBM, performing the same simulation, are illustrated.

Once again, the Kundur system is considered with 8 digital controllers, one governor, and an AVR for each synchronous generator.

5.1.2.1 Slow controllers

In the first case, we use controllers with slower sampling rates. Therefore, there will be less number of time events in each time step, and less number of them will be neglected by SSM. For this scenario, the sampling rate T of the digital controllers is set to 210 ms, 220 ms, 230 ms, and 240 ms for the governors, and 41 ms, 42 ms, 43 ms, and 44 ms for the exciters.

A short circuit on bus 3 for 200 ms is simulated using all four methods of SRM, SSM, IBM, and

SIBM. The simulation outputs are shown in Figs. 5.3, 5.4, 5.5, and 5.6, for the voltage of bus 1, the speed deviation of the third generator, the governor output of the third generator, and the exciter output of the third generator, respectively. It can be seen that while SSM has some minor accuracy loss, SIBM is perfectly following the correct trajectory, similar to IBM.

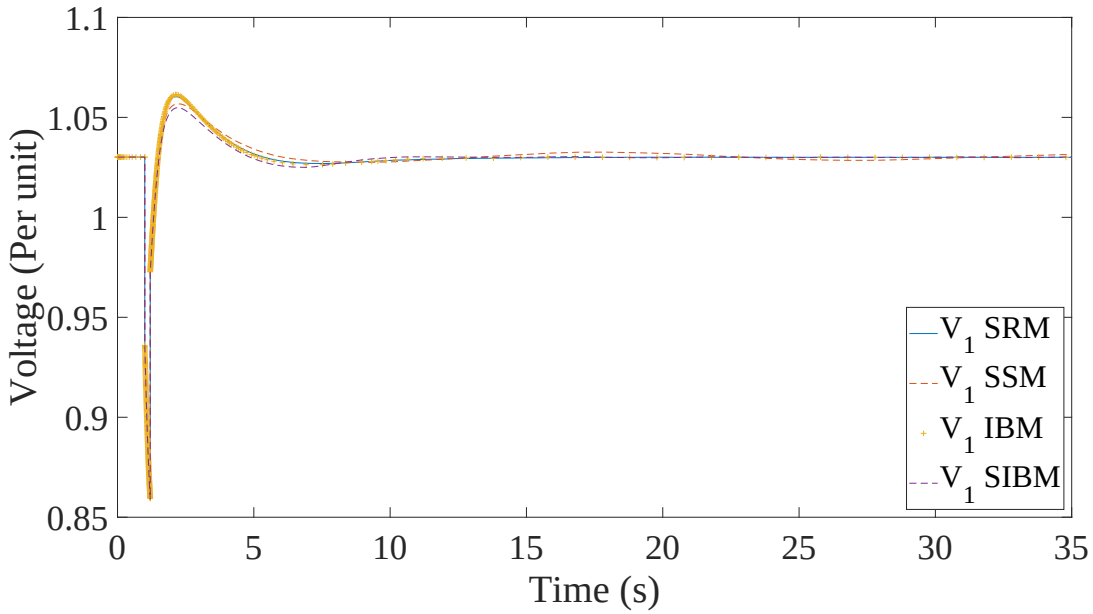

Figure 5.3: Voltage of bus 1 of the Kundur test system with slow controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

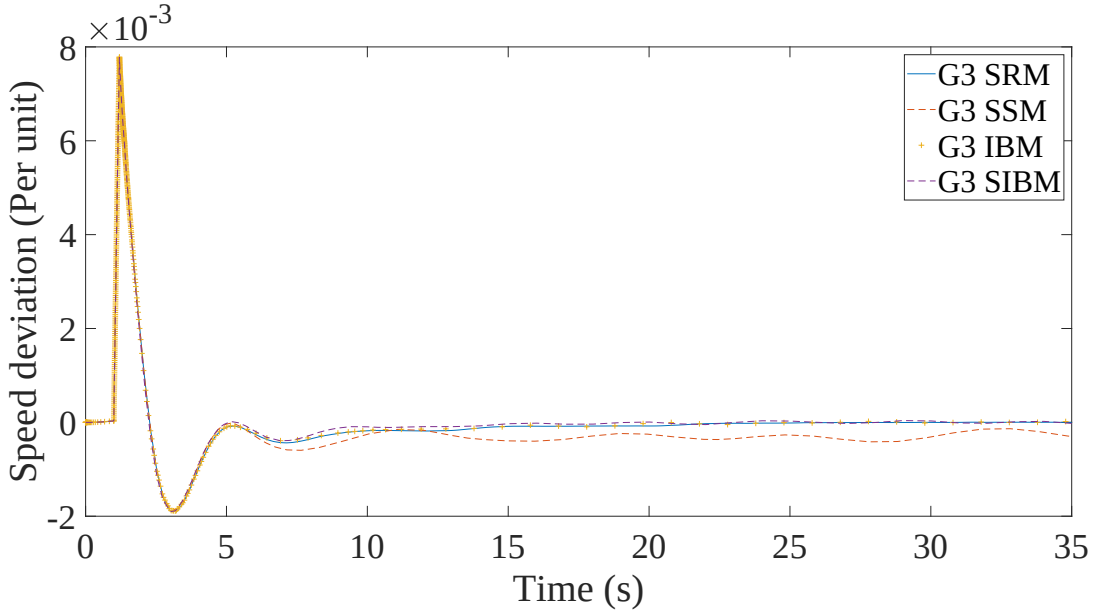

Figure 5.4: Speed deviation of generator 3 of the Kundur test system with slow controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

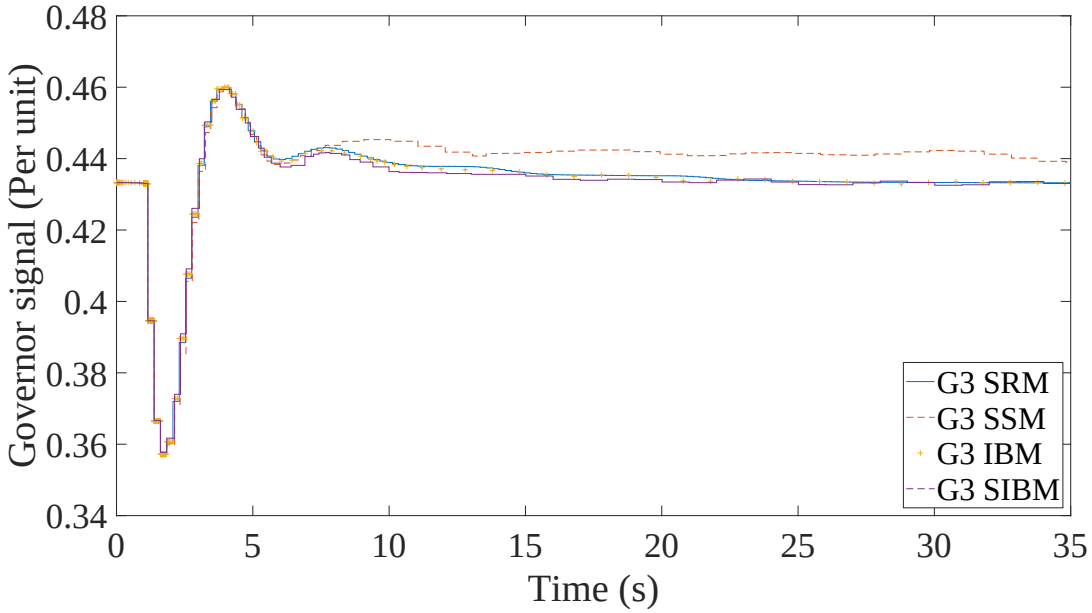

Figure 5.5: Governor output of generator 3 of the Kundur test system with slow controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

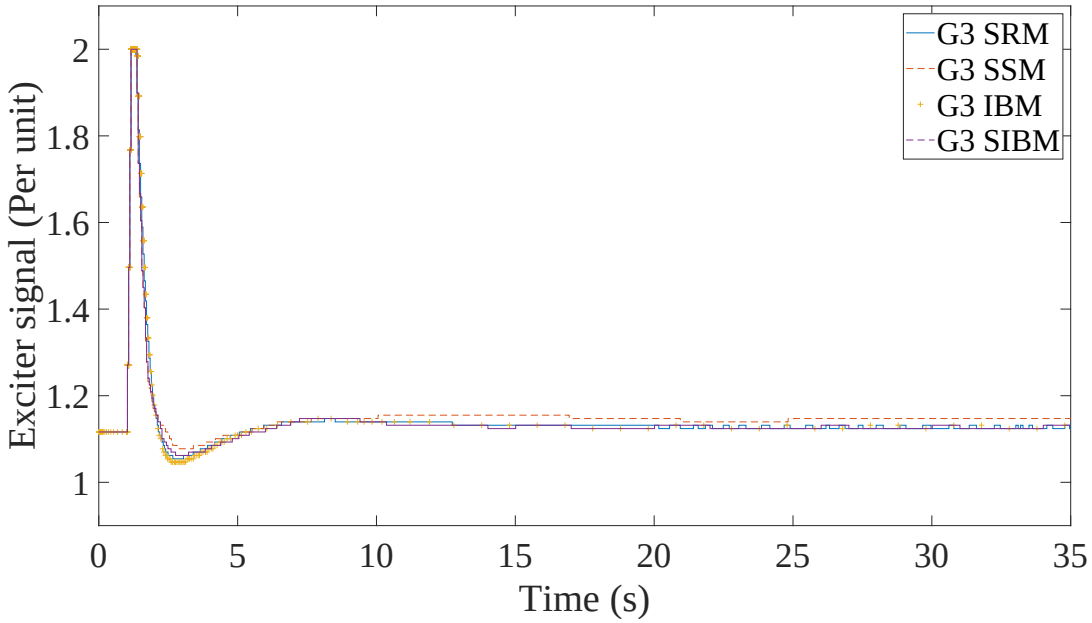

Figure 5.6: Exciter output of generator 3 of the Kundur test system with slow controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

The performance results of the simulations using all the methods are listed in Table 5.1, in terms of the number of Newton iterations and average run times. As expected, SSM is the fastest method. SIBM is the second fastest, followed by IBM, which solves a larger system with an extended state variables vector.

The step size results for the method are illustrated in Fig. 5.7. As evident, SSM reduces the step size

Table 5.1: Performance comparison between SRM, SSM, IBM, and SIBM for Kundur system with slow controllers in terms of the number of Newton iterations, and runtime

Method	SRM	SSM	IBM	SIBM
N. Newton iterations	16813	1198	1288	1214
Average runtime (s)	66.14	4.72	5.9	5.13

a couple of times due to accumulated error estimates. However, it hits the maximum time step first.

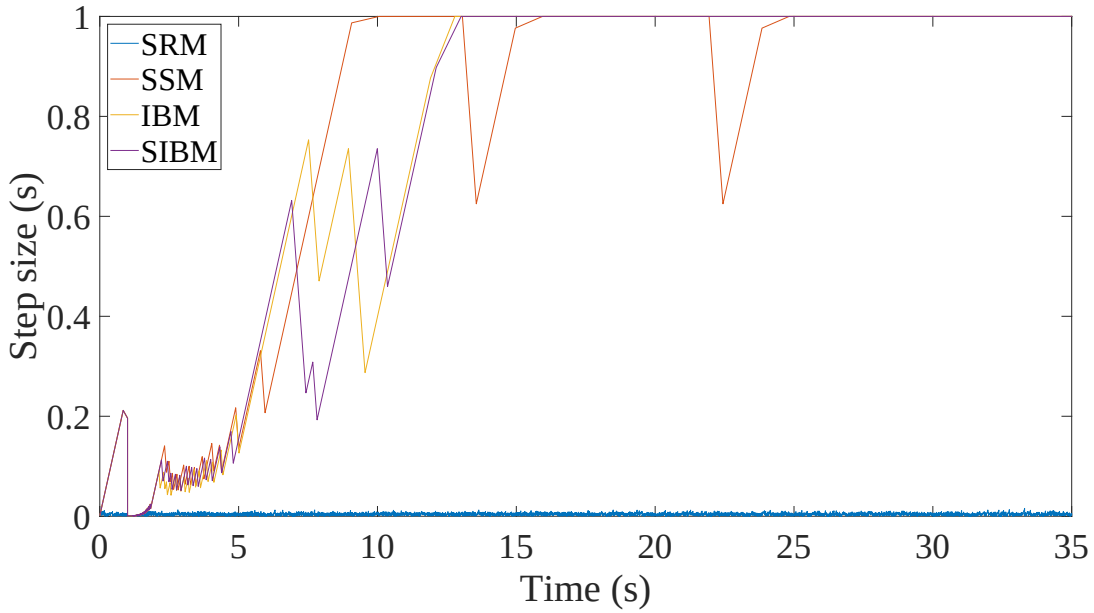

Figure 5.7: Step size results for all the methods simulating the Kundur system with slow controllers. The minimum and maximum time step size limits are equal to 1 ms and 1 s, respectively, for all the methods.

5.1.2.2 Fast controllers

The same simulation is repeated here with controllers with faster sampling rates. For this scenario, the sampling rate T of the digital controllers is set to 21 ms, 22 ms, 23 ms, and 24 ms for the governors, and 4.1 ms, 4.2 ms, 4.3 ms, and 4.4 ms for the exciters. In other words, each controller is 10 times faster than the previous scenario.

The simulation outputs are shown in Figs. 5.8, 5.9, 5.10, and 5.11, for the voltage of bus 1, the speed deviation of the third generator, the governor output of the third generator, and the exciter output of the third generator, respectively.

In this scenario, SSM struggles to find the steady-state and follow the fluctuations correctly. To solve this issue, the user can decrease the maximum time step to limit the number of events that can fall in each time step. However, in that case, SSM won't be a fast method after all. Nevertheless, SIBM shines, being able to follow the trajectories more accurately once more.

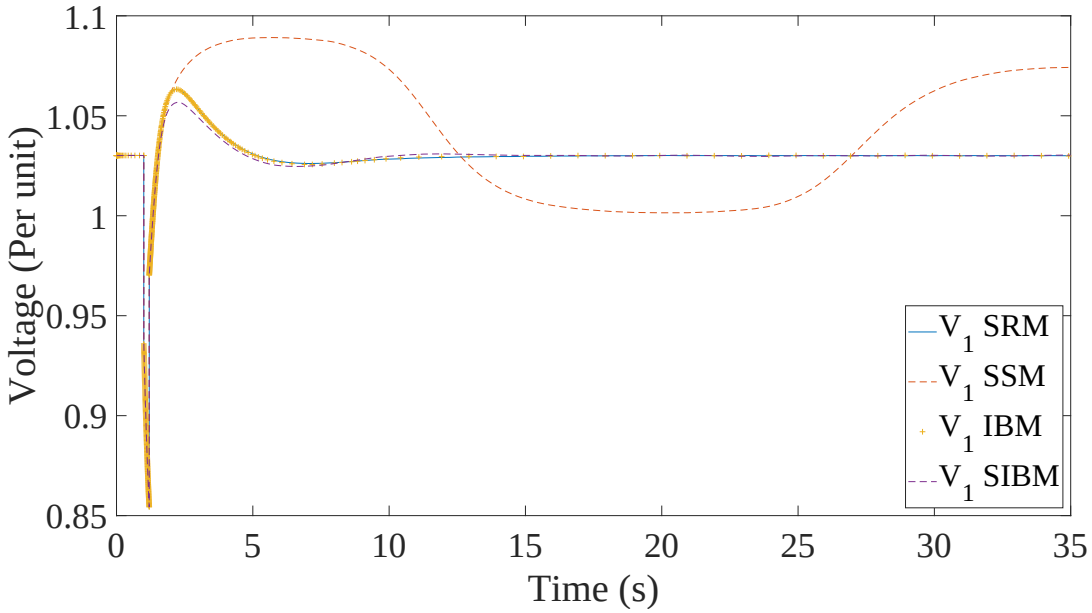

Figure 5.8: Voltage of bus 1 of the Kundur test system with Fast controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

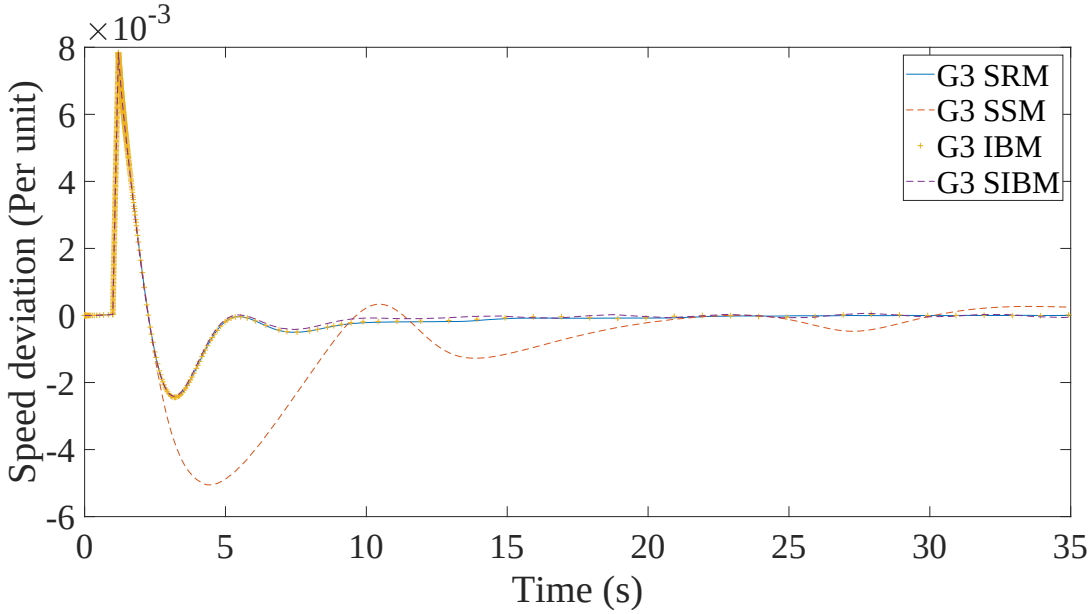

Figure 5.9: Speed deviation of generator 3 of the Kundur test system with fast controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

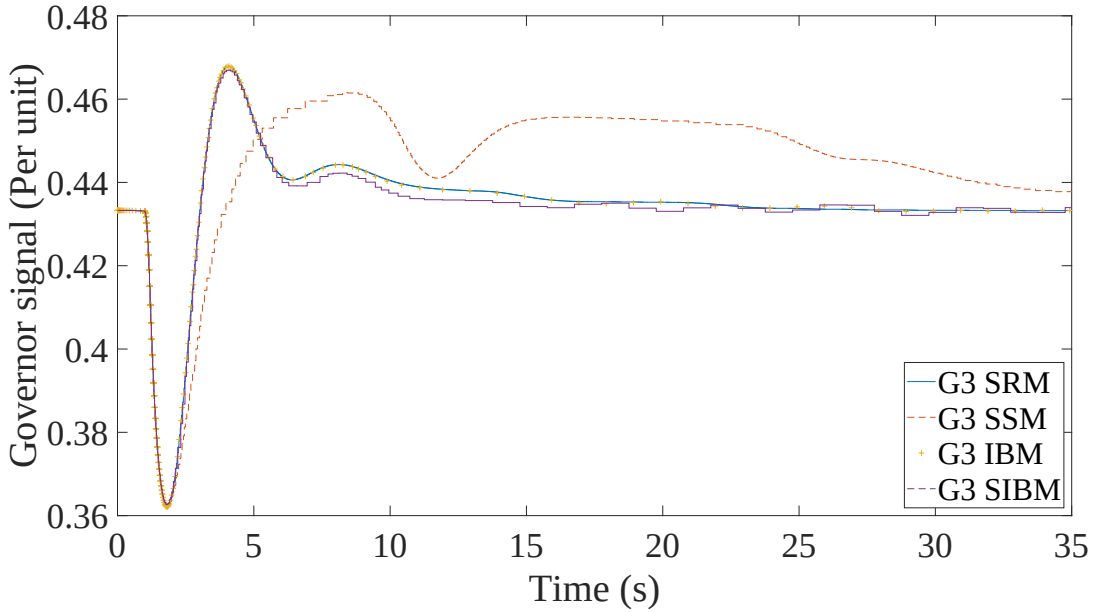

Figure 5.10: Governor output of generator 3 of the Kundur test system with fast controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

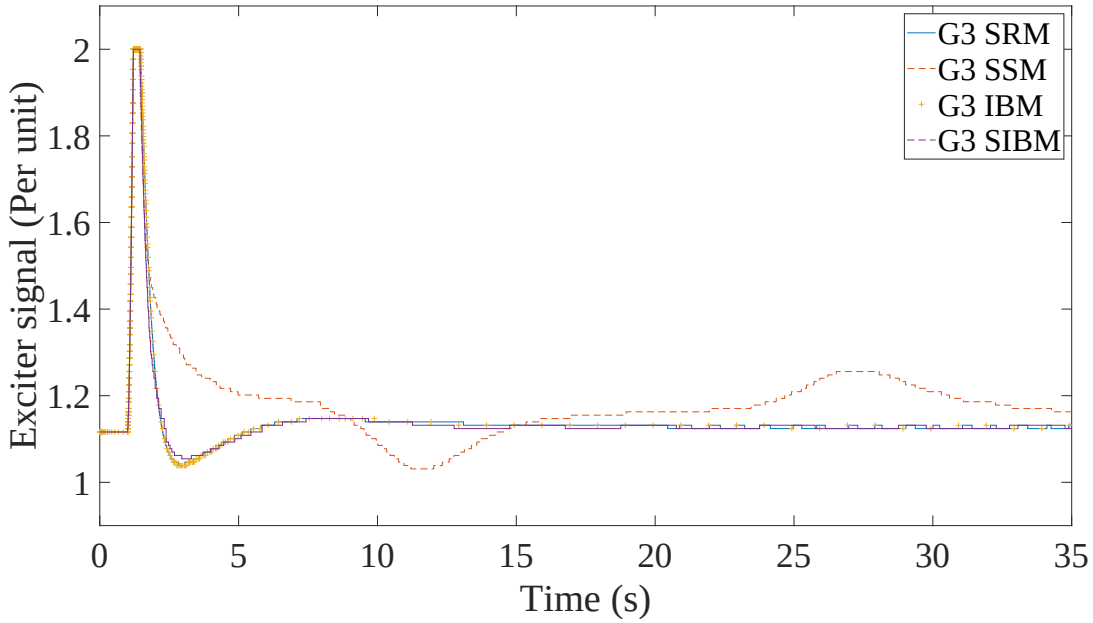

Figure 5.11: Exciter output of generator 3 of the Kundur test system with fast controllers. The blue solid line is SRM, a reference point as the most accurate approach, the yellow pluses represent the IBM, while the dashed red and purple lines are the SSM and SIBM, respectively.

The performance results for simulations of fast controllers are listed in Table 5.2. As can be seen, now the faster method is SIBM, as SSM has more Newton iterations. There are two reasons for extra Newton iterations of SSM. First, it needs more Newton iterations to reach convergence. In addition, SSM often reduces time steps to avoid high error estimates. This is evident in Figs. 5.12, which depicts the step size results for all four methods. As can be seen, SSM is busy decreasing the time steps periodically

to keep the error estimate below the threshold while finding the steady-state.

Table 5.2: Performance comparison between SRM, SSM, IBM, and SIBM for Kundur system with fast controllers in terms of the number of Newton iterations, and runtime

Method	SRM	SSM	IBM	SIBM
N. Newton iterations	164399	1685	1291	1241
Average runtime (s)	661.70	6.41	11.9	5.58

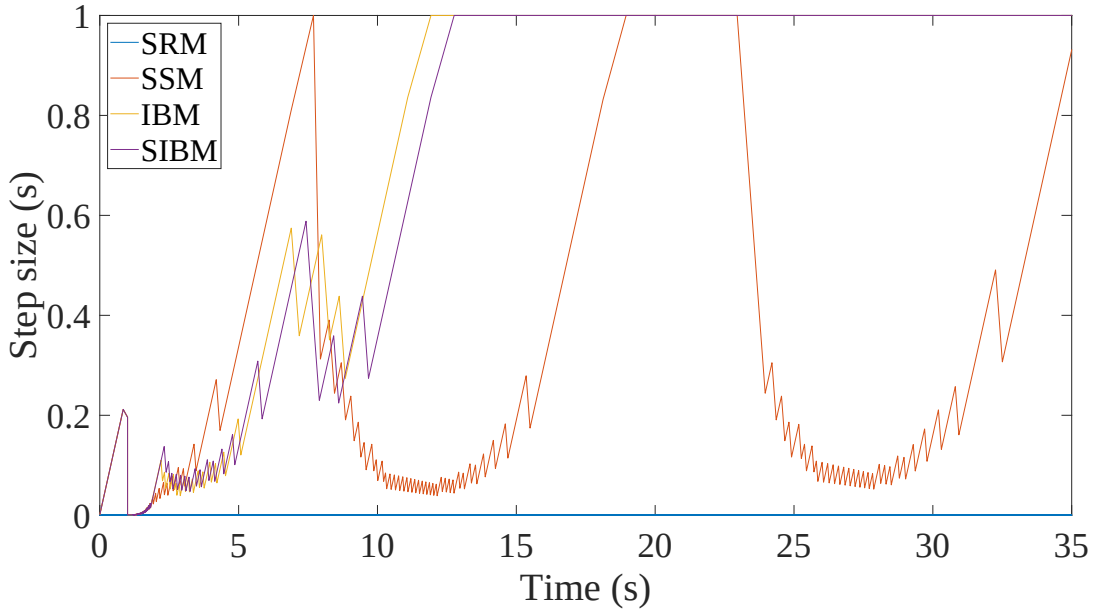

Figure 5.12: Step size results for all the methods simulating the Kundur system with fast controllers. The minimum and maximum time step size limits are equal to 1 ms and 1 s, respectively, for all the methods.

Another interesting point regarding the performance of the methods is that SRM is 10 times slower compared to the first scenario with slower controllers, as can be read from Table 5.3, listing the performance drops in terms of run times and Newton iterations. The reason, of course, is that the step sizes are heavily cramped by the tiny spaces between the controllers' samplings. However, the other 3 methods don't get that performance drop rate, losing only a few seconds, as they still jump over multiple discontinuities. Meanwhile, SIBM has the lowest performance drop ratio (in terms of runtime) since it solves a much smaller system compared to IBM.

Table 5.3: Performance drop rate between two scenarios for SRM, SSM, IBM, and SIBM for Kundur system in terms of the number of Newton iterations, and runtime

Method	SRM	SSM	IBM	SIBM
N. Newton iterations drop ratio	9.77	1.40	1.00	1.02
Runtime drop ratio	9.99	1.35	2.01	1.08

5.2 Light IBM

As was discussed so far, digital controllers introduce numerous discontinuities during the simulation that are challenging to handle. IBM was proposed to address this challenge by including the impact of the discontinuities at the end of a large time step using interpolation polynomials and including the controller outputs in the Newton solver process. However, IBM will suffer from a performance drop if computing the controller output is computationally heavy relative to a Newton iteration performed. The reason is that IBM relies on calling the controller multiple times instead of reducing the time step. In this section, a light version of IBM is proposed for handling computationally heavy controllers that covers the IBM performance drop while facing them.

5.2.1 Computationally heavy controllers

During the dynamic simulation of a system with digital controllers, the controller is called with feedback from the system that is injected into the controller block as an input. The computations that need to be executed inside the controller block to produce an output take some time. The time delay between feeding the input to the controller and resulting in an output from the controller is called the controller call cost here.

Let us assume the controller call cost is a, and solving each Newton iteration cost is b. IBM calls the controller function ζ once per sampling per Newton iteration, therefore, the total controller cost for IBM is equal to $tc_{IBM} = (a \times T \times m) + (b \times m)$, while SRM calls the controller once each sampling time, so $tc_{SRM} = (a \times T) + (b \times m^*)$. It should be noted that the number of Newton iterations for SRM is many times more than IBM, depending on the size of the time steps taken ($m \neq m^*$). If the controller call cost a is computationally light compared to the cost of solving each Newton iteration ($a \ll b$), the total cost of the simulation is $tc \cong b \times m$. In this case, IBM has better performance than SRM since it takes large time steps and the solver solves the system many times fewer than SRM ($m \ll m^*$). However, if the controller call cost is large and not negligible compared to b, then cc is no longer negligible and $tc = b + cc$. Now, since IBM calls the controller m times more than SRM, then $tc_{IBM} > tc_{SRM}$. In other words, simulating a computationally demanding controller with IBM may lead to slower simulation compared to SRM if the added cost of the extra controller calls is significant compared to the computations saved by not reducing the time step.

In addition, the size of the system to be solved increases per each sampling found in each time step for IBM. For example, the size of the Jacobian for a system with i equations and j controller equations each having r samples in a time step is equal to $(i + (j \times r)) \times (i + (j \times r))$ while it remains $i \times i$ for SRM

5.2.2 Controller calls

The first modification restricts calling the controller evaluation functions only in the first Newton iteration m. Therefore, the controller call cost in IBM becomes $cc_{IBM} = a \times T$, similar to SRM. In other words, the second term of (4.17), which is the controller function ζ remains constant for the rest of the iterations in each time step. This reduces the number of controller calls and the use of an interpolation polynomial by $m - 1$ times per controller per sampling in a time step.

The drawback of this modification is that the Jacobian and the controller mismatches are accurate only for the first Newton iteration. Therefore, minor inaccuracies are expected to occur.

5.2.3 Size of the Jacobian

The second modification aims to reduce the size of the system to be solved by including only the last controller output in each time step to form the Jacobian ($r = 1$). Therefore, the Jacobian size remains constant and equal to $(i+j)\times(i+j)$ for the time step (assuming that the controller has at least 1 sample in the time step taken). Therefore, less computation is needed to solve the system in every Newton iteration.

Using the modifications suggested, the number of calls to the controllers and the size of the system to be solved decrease, leading to an increase in performance.

In the next section, the performance increase and the impact of the modifications on the accuracy are demonstrated.

5.2.4 Simulation results

To compare the methods introduced in the previous section, a 3-bus network illustrated in Fig. 5.13 is considered. The synchronous generator is assumed to be under the control of a digital governor and a digital exciter again. For the governor, an equation-based controller with its continuous block diagram shown in I is considered. For the exciter, however, three different controllers are simulated as follows:

1. An equation-based AVR (EQAVR) with its continuous block diagram presented in I.
2. An AVR based on fuzzy logic (FAVR).
3. A machine learning-based AVR (MLAVR) trained based on the results obtained from the simulation of EQAVR.

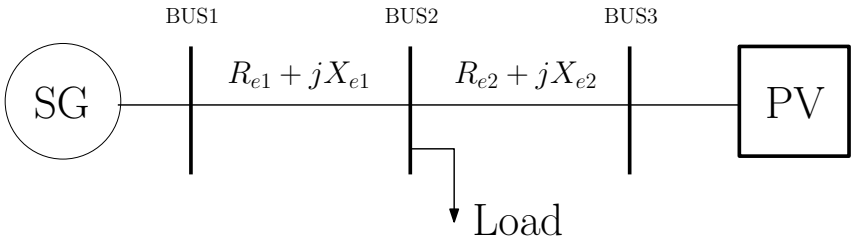

Figure 5.13: The schematic of the 3-bus network

The average call time of the controller for each of these controllers is listed in Table 5.4. Considering that each Newton iteration takes about 7×10^{-5} s to be solved, the MLAVR can be considered a heavy controller. Each controller is used for simulation with all three methods of SRM, IBM, and LIBM to compare the results in terms of accuracy and performance. 20 ms and 4 ms are considered for the sampling time of the governor and all AVRs, respectively.

Table 5.4: Average controller calling time for EQAVR, FAVR, and MLAVR

Controller	Average call time (s)
EQAVR	2.78e-06
FAVR	1.72e-04
MLAVR	3.7e-03

A short circuit, applied to bus 2 for 100 ms, is simulated for all three methods and all three variations of the AVR controller. The controller output for EQAVR, FAVR, and MLAVR, simulated with all three methods, is illustrated in Figs. 5.14, 5.15, and 5.16, respectively. Furthermore, the voltage of bus 1 for all three controllers EQAVR, FAVR, and MLAVR is depicted in Figs. 5.17, 5.18, and 5.19, respectively.

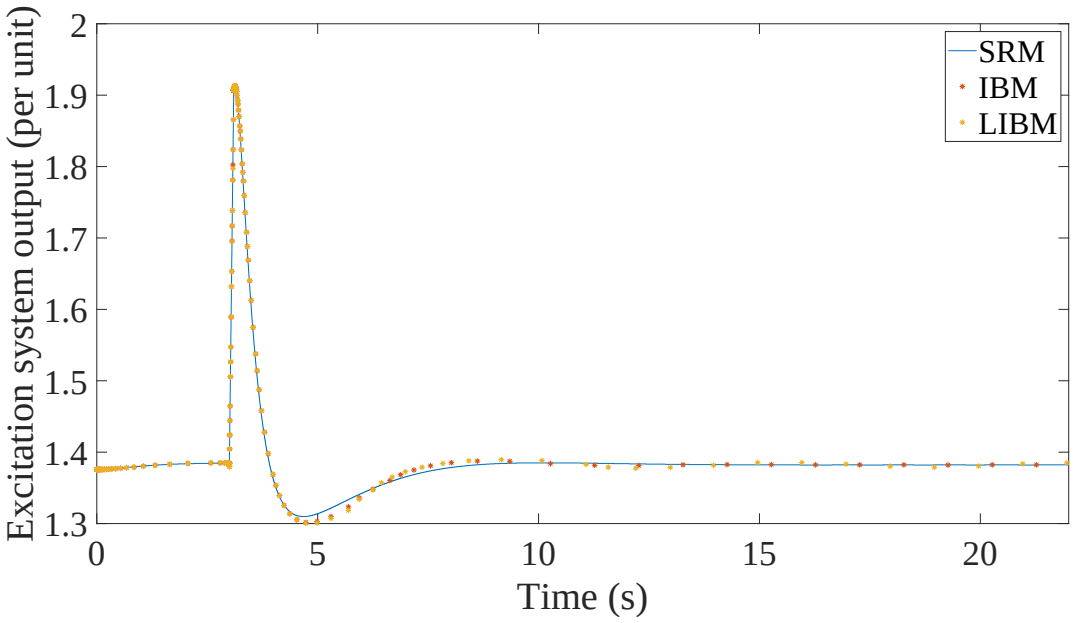

Figure 5.14: Output of the EQAVR using all three methods of SRM, IBM, and LIBM

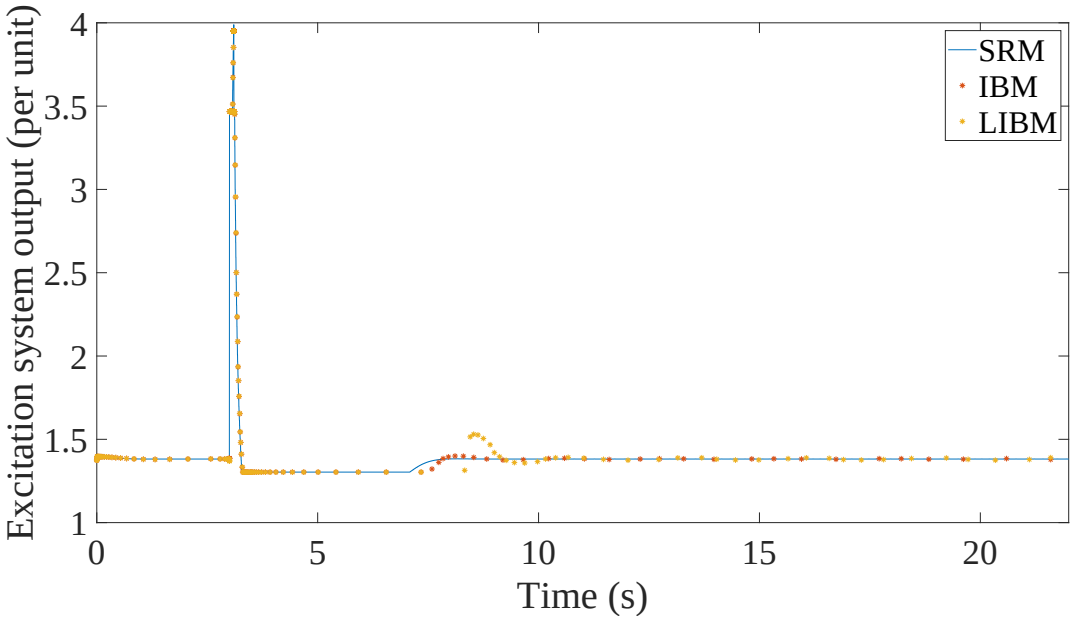

Figure 5.15: Output of the FAVR using all three methods of SRM, IBM, and LIBM

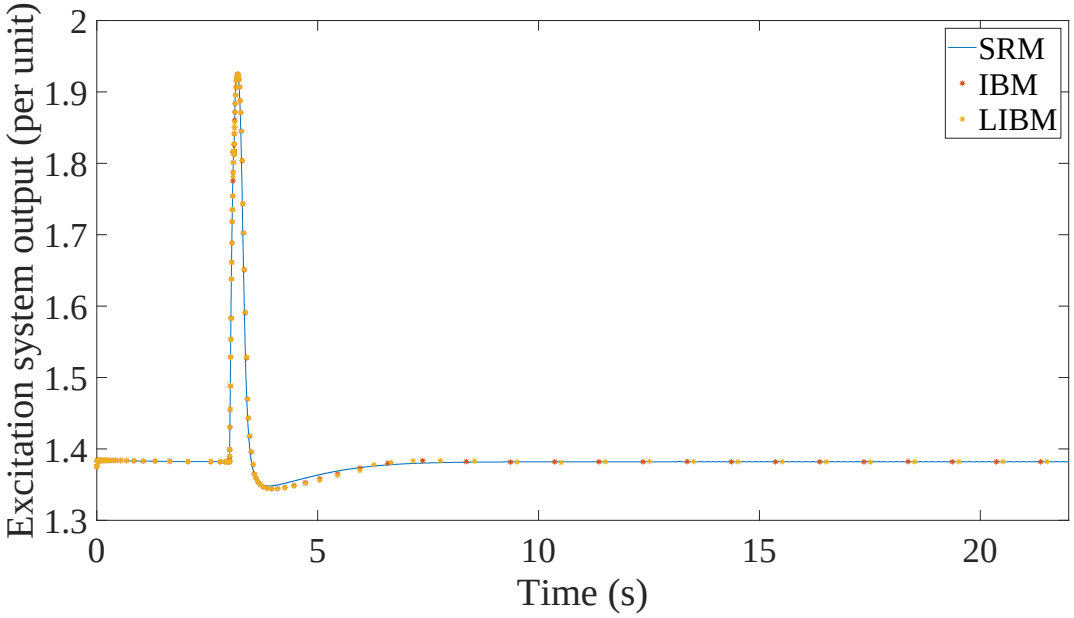

Figure 5.16: Output of the MLAVR using all three methods of SRM, IBM, and LIBM

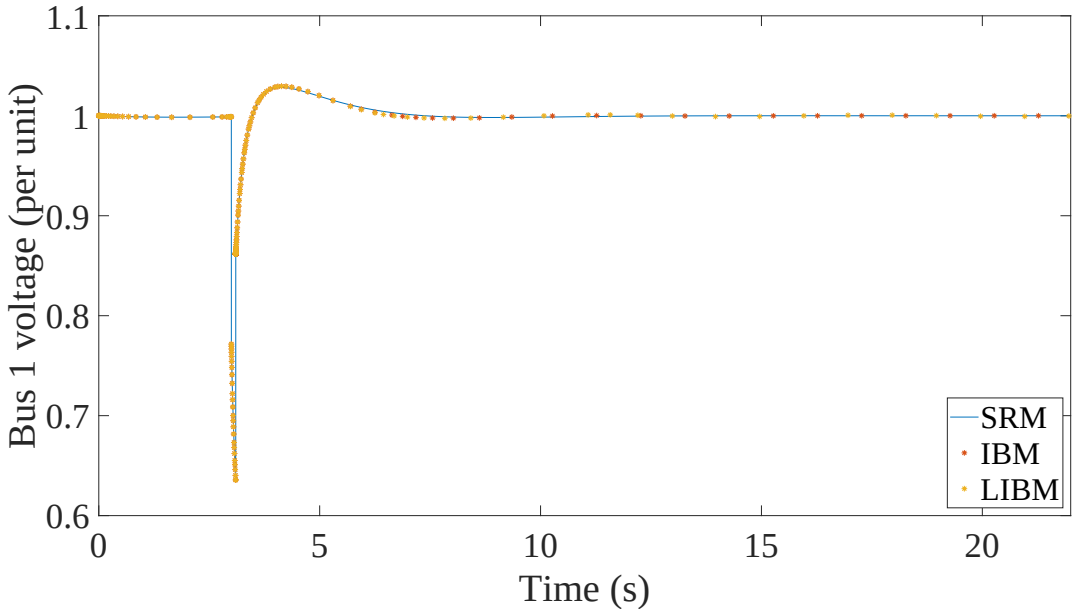

Figure 5.17: Voltage of bus 1 with the EQAVR using all three methods of SRM, IBM, and LIBM

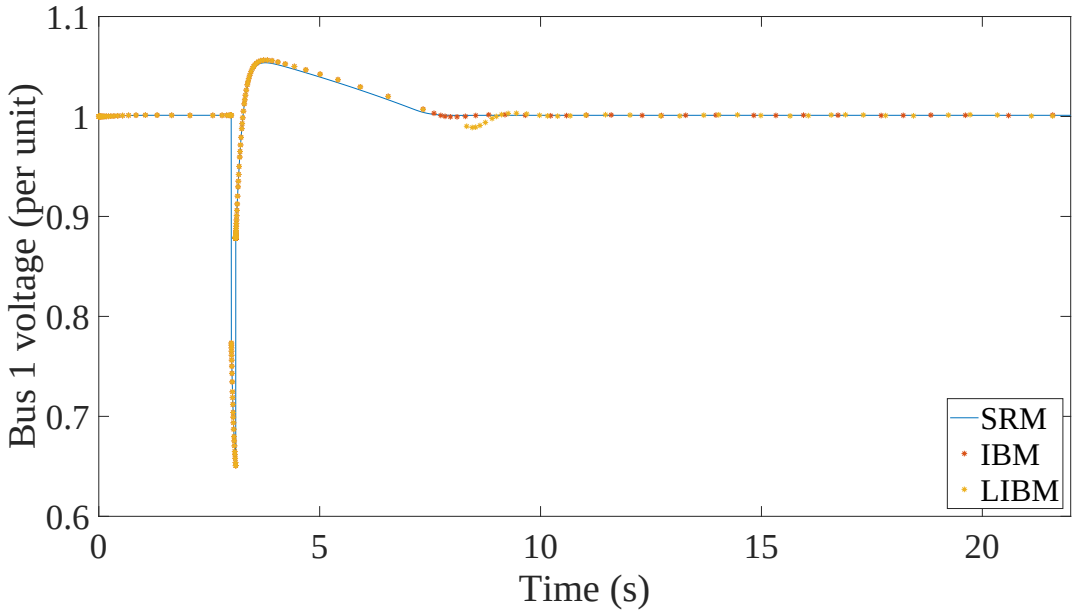

Figure 5.18: Voltage of bus 1 with the FAVR using all three methods of SRM, IBM, and LIBM

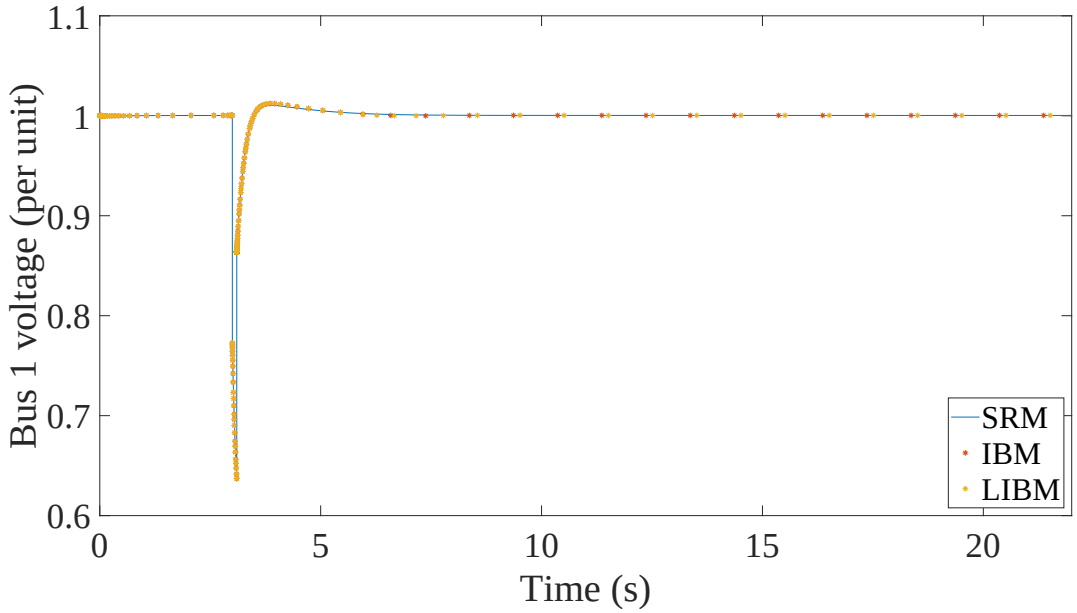

Figure 5.19: Voltage of bus 1 with the MLAVR using all three methods of SRM, IBM, and LIBM

The performance results in terms of the average of 10 runs for each simulation are listed in Table 5.5. The first row of the table shows that the runtime for MLAVR simulation using SRM is almost twice as that of the simulation of EQAVR and FAVR using the same method. However, this performance drop is more significant for IBM and LIBM, since controller calling is the heaviest computation for these methods. This can be deduced from Figs. 5.20, 5.21, and 5.22 that show the step size results for the simulation of EQAVR, FAVR, and MLAVR, respectively, using all three methods. In other words, although IBM still manages to take the maximum step size allowed for most of the simulation time of MLAVR, the run time decreases significantly due to the long time and numerous controller calls.

Table 5.5: Average runtime of the simulation of each controller and method in seconds.

	EQAVR (s)	FAVR (s)	MLAVR (s)
SRM	23.90	24.80	42.91
IBM	0.48	4.00	56.93
LIBM	0.36	2.34	34.46

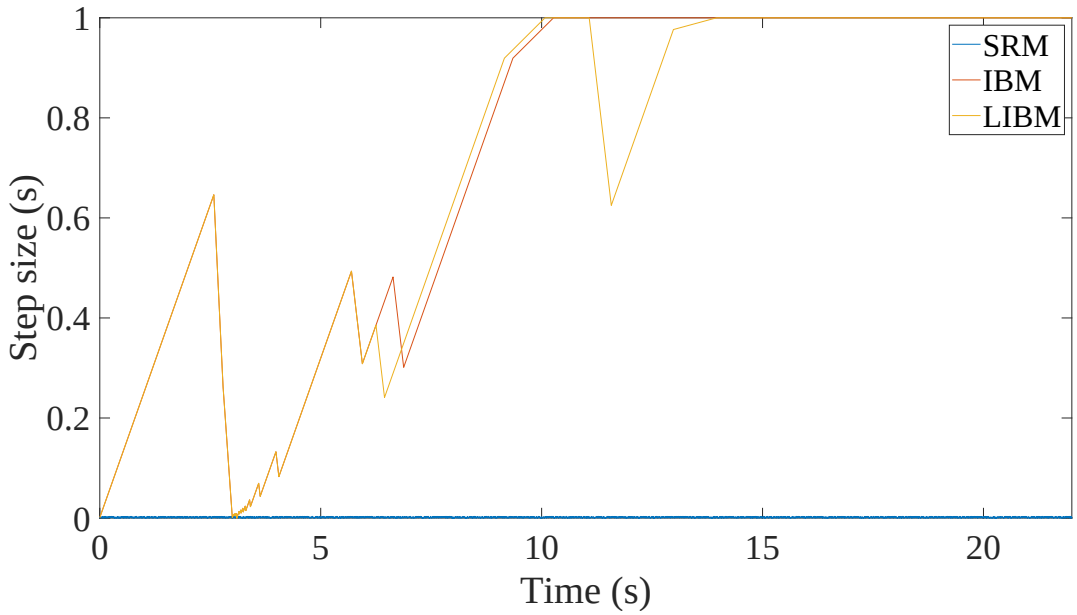

Figure 5.20: Step size results for the simulation of EQAVR using all three methods of SRM, IBM, LIBM

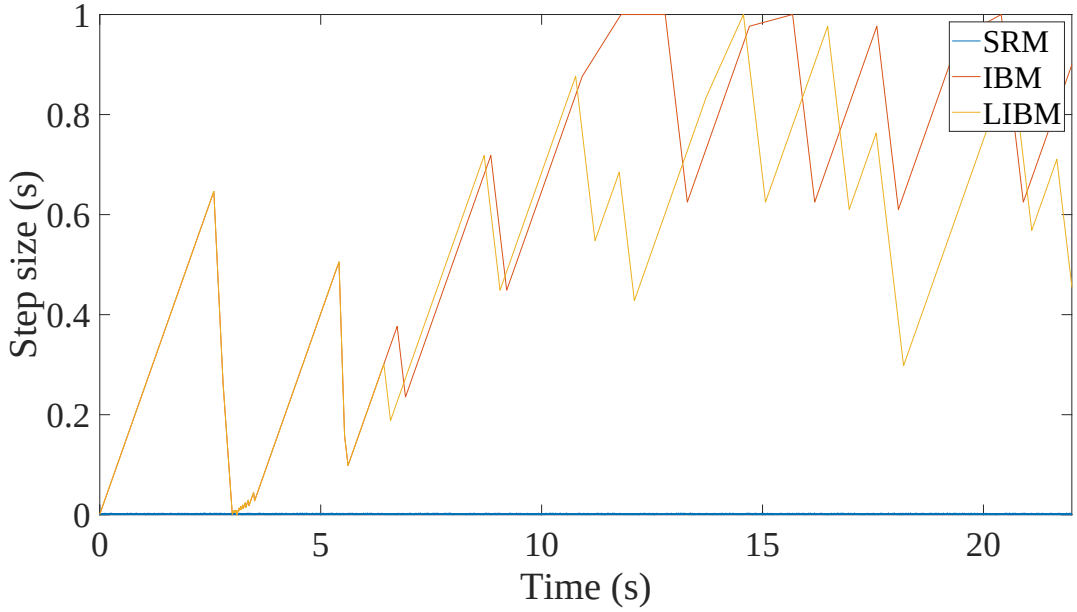

Figure 5.21: Step size results for the simulation of FAVR using all three methods of SRM, IBM, LIBM

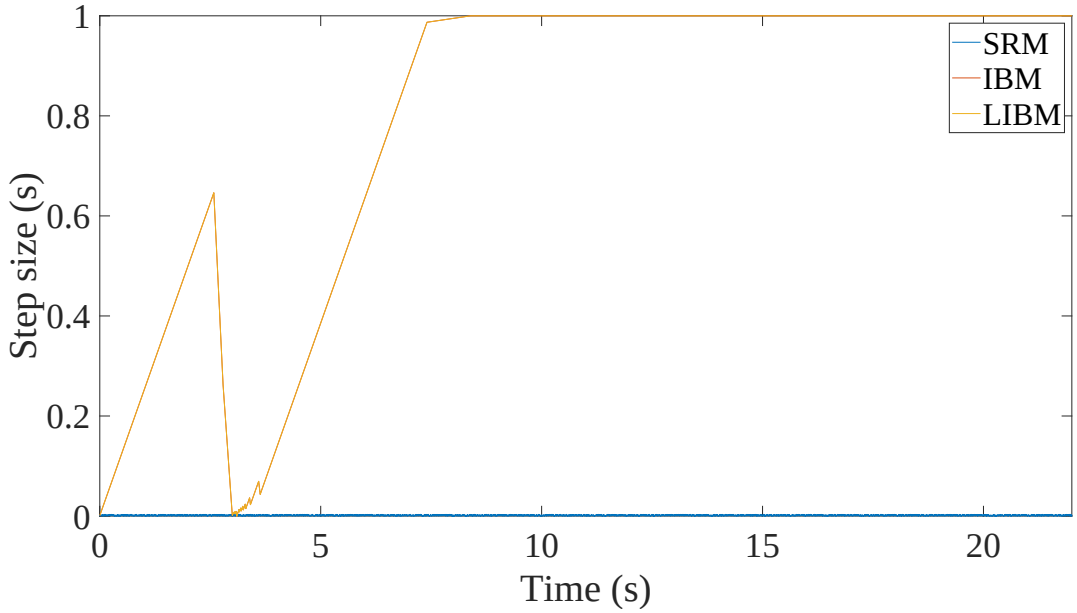

Figure 5.22: Step size results for the simulation of MLAVR using all three methods of SRM, IBM, LIBM

The main challenge is that IBM, which is many times faster than SRM for the simulation of EQAVR and FAVR, becomes 14 s slower than SRM for the simulation of MLAVR. However, LIBM is still 8 s faster than SRM because of fewer controller calls and solving smaller systems. It should be noted that this increase in performance is not significant for other controllers, so, in the case of normal controllers, IBM would still be the better choice, ensuring better accuracy.

The number of Newton iterations required for the simulation of each controller using each method is shown in Table 5.6. As can be seen, fewer Newton iterations are needed to solve the MLAVR system, which supports the fact that the number of controller calls leads to a performance drop. It can be seen that using LIBM leads to an increase in the number of Newton iterations since the Jacobian updates were done once per time step. Nevertheless, LIBM is faster due to a reduced number of controller calls and lower computation volume in each Newton iteration.

Table 5.6: Number of Newton iterations for the simulation of each controller and method.

	EQAVR	FAVR	MLAVR
SRM	54604	55136	54504
IBM	551	585	520
LIBM	554	598	522

5.3 Moved-Jacobian IBM

In this section, another novel method for handling the time events caused by the action of digital controllers in power systems is proposed. The method is similar to IBM but with a new structural design. The idea is to move the Jacobian formation to the controller's input instead of its output, which was implemented in IBM. The new method also allows integration within large time steps, without reducing it.

The highlights of the proposed method are as follows:

- Proposing a new design of IBM for the simulation of digital controllers in dynamic studies of power systems.
- In contrast to IBM, there is no element of the controller in the solver.
- The new method considers the controllers with the same input and sampling rate as one controller. Therefore, the performance is expected to increase for such settings in a system.

5.3.1 Formulation

It is assumed that, other than extended state variables that are moved from the controller output to its input, the integration method, convergence test, and time step selection schemes are similar to IBM. To discuss the formulation of the moved-Jacobian interpolation-based method (MJIBM), let's once more consider the continuous system under control by a digital controller as described using equations the following equations for the continuous system:

$$\begin{aligned} \dot{\boldsymbol{y}}(t) &= \boldsymbol{f}(\boldsymbol{y}(t), \boldsymbol{e}(t)) \\ \boldsymbol{y}(0) &= \boldsymbol{y_0},\ \boldsymbol{e}(0) = \boldsymbol{e}_0 \end{aligned} \tag{5.5}$$

and the digital controller:

$$\boldsymbol{e}_k = \zeta(\boldsymbol{e}_{k-1}, \boldsymbol{y}(kT)) \tag{5.6}$$

where the objective is to find the solution $\boldsymbol{y}_n$ at time t_n using the integration method chosen over the time step $h_n = t_n - t_{n-1}$. The residual function $\boldsymbol{g}$ to be solved for that purpose is as follows:

$$\boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{e}_{n,p_n}) = 0 \tag{5.7}$$

We need a set of initial values (an initial guess for each iteration) and the controller's signal for each time step. In the case of the first iteration, the initial guess is the predictor's output, while for the next iterations, the result of the previous iterations can be used. The latest controller's output value, however, depends on the previous ones and the state variables' values $\boldsymbol{y}_{n,g}$ in their intermediate time step (controller sampling times) $t_{n,g}$ as given by:

$$\begin{aligned} &\boldsymbol{e}_{n,1} = \zeta(\boldsymbol{e}_{n,0}, \boldsymbol{y}_{n,1}, t_{n,1}) \\ &... \\ &\boldsymbol{e}_{n,g} = \zeta(\boldsymbol{e}_{n,g-1}, \boldsymbol{y}_{n,g}, t_{n,g}) \\ &... \\ &\boldsymbol{e}_{n,p_n} = \zeta(\boldsymbol{e}_{n,p_{n-1}}, \boldsymbol{y}_{n,p_n}, t_{n,p_n}) \end{aligned} \tag{5.8}$$

A rough estimation of intermediate state variables $\boldsymbol{y}_{n,g}$ can be found using a polynomial interpolator function $\boldsymbol{w}_n$ like the following:

$$\begin{aligned}\boldsymbol{y}_{n,g} =& \boldsymbol{w}_n(t_{n,g}) = \boldsymbol{y}_{n-1} + h_{n,g}\dot{\boldsymbol{y}}_{n-1} \\ &+ \frac{h_{n,g}^2}{h_n^2}(\boldsymbol{y}_n - \boldsymbol{y}_{n-1} - h_n\dot{\boldsymbol{y}}_{n-1})\end{aligned} \tag{5.9}$$

which is a second-order Taylor expansion function similar to IBM. Instead of including the controller's outputs in the solver and updating them in each Newton iteration (IBM), the controller inputs are included in the solver alternatively. In other words, instead of extending the problem with controller outputs, we move the extended part of the problem to the controller inputs and form the Jacobian in that position (moved Jacobian).

To do so, a new state variable vector $\boldsymbol{z}_n$ is defined:

$$\boldsymbol{z}_n = \begin{bmatrix}\boldsymbol{z}_{n,1} \\ \boldsymbol{z}_{n,2}\end{bmatrix} \tag{5.10}$$

with:

$$\boldsymbol{z}_{n,1} = \boldsymbol{y}_n \tag{5.11}$$

$$\boldsymbol{z}_{n,2} = \begin{bmatrix}\boldsymbol{x}_{n,1} & \boldsymbol{x}_{n,2} & \dots & \boldsymbol{x}_{n,g} & \dots & \boldsymbol{x}_{n,p_n}\end{bmatrix}^T \tag{5.12}$$

where $\boldsymbol{x}_{n,g}$ is a subset of $\boldsymbol{y}_{n,g}$ that the controller has under monitor (the feedback). For example, in the case of a simple generator's exciter, it would be the generator's terminal voltage. This is a crucial point to be noted since in the case of large-scale systems, including all the state variables for each sampling time leads to solving an enormous size of Jacobian in each iteration, many times bigger than the size of the system itself. It can be noted that the first part of the state variable vector, called $\boldsymbol{z}_{n,1}$ is identical to IBM, while the extended part $\boldsymbol{z}_{n,2}$ is expanded based on the controller's input.

For the new state variable vector defined, a new residual vector is considered:

$$\tilde{\boldsymbol{g}} = \begin{bmatrix}\tilde{\boldsymbol{g}}_1 \\ \tilde{\boldsymbol{g}}_2\end{bmatrix} \tag{5.13}$$

with:

$$\tilde{\boldsymbol{g}}_1(\boldsymbol{z}_n) = \boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{e}_{n,p_n}) \tag{5.14}$$

$$\tilde{\boldsymbol{g}}_2(\boldsymbol{z}_n) = \begin{bmatrix}\boldsymbol{x}_{n,1}^s - \boldsymbol{x}_{n,1}^w \\ \dots \\ \boldsymbol{x}_{n,g}^s - \boldsymbol{x}_{n,g}^w \\ \dots \\ \boldsymbol{x}_{n,p_n}^s - \boldsymbol{x}_{n,p_n}^w\end{bmatrix} \tag{5.15}$$

where superscripts s and w mean $\boldsymbol{x}$ is the output result of the solver and the interpolator, respectively. Contrary to IBM, the interpolated values $\boldsymbol{x}_{n,g}^w$ are only used to form the residual vector and not for the computation of the controller's output. Instead, the solver's output $\boldsymbol{x}_{n,g}^s$ is replaced in (5.8) sequentially

until $\boldsymbol{e}_{n,p_n}$ is calculated to be fed to the system's equations for the next iteration. In other words, instead of solving an iteration to find the controller output, it finds and updates the value for its input, which is used to update and find the new output.

Now the m-th Newton iteration to be solved is:

$$\boldsymbol{J}_n^{(m)}(\boldsymbol{z}_n^{(m+1)} - \boldsymbol{z}_n^{(m)}) = -\tilde{\boldsymbol{g}}(\boldsymbol{z}_n^{(m)}) \tag{5.16}$$

that results in new values for the system's state variables $\boldsymbol{z}_{n,1}$ and the system's state variables under monitoring by the controller for each intermediate step $\boldsymbol{z}_{n,2}$. Also, $\boldsymbol{J}_n^{(m)}$ is the Jacobian matrix of the extended system at the m-th Newton iteration, given by:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \end{bmatrix} \tag{5.17}$$

Since the computation of $\boldsymbol{J}_n^{(m)}$ can be extremely complex, a simplified Jacobian is considered:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{0} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} \approx \boldsymbol{0} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \approx \boldsymbol{I}_{p_n} \end{bmatrix} \tag{5.18}$$

In comparison to IBM, it can be noted that the extended part of the Jacobian is moved from the controller's outputs to its inputs. Contrary to IBM, it can be seen that absolutely no element of the controller is involved in the Jacobian formation; all the state variables in the extended vector are coming from the system's equations via an interpolation polynomial, only at different points in time. From the viewpoint of the solver, no controller is seen in the system, but as an input to the system's equations. Even in the case of using the continuous equivalent model of the controllers in traditional common methods, the controllers' differential-algebraic equations are integrated alongside the system's equations. This means, similar to IBM and in a more suitable way, even non-equation-based digital controllers can be numerically included and solved without any step size reductions using the proposed method.

The flowchart of MJIBM for one time step is presented in Fig. 5.23

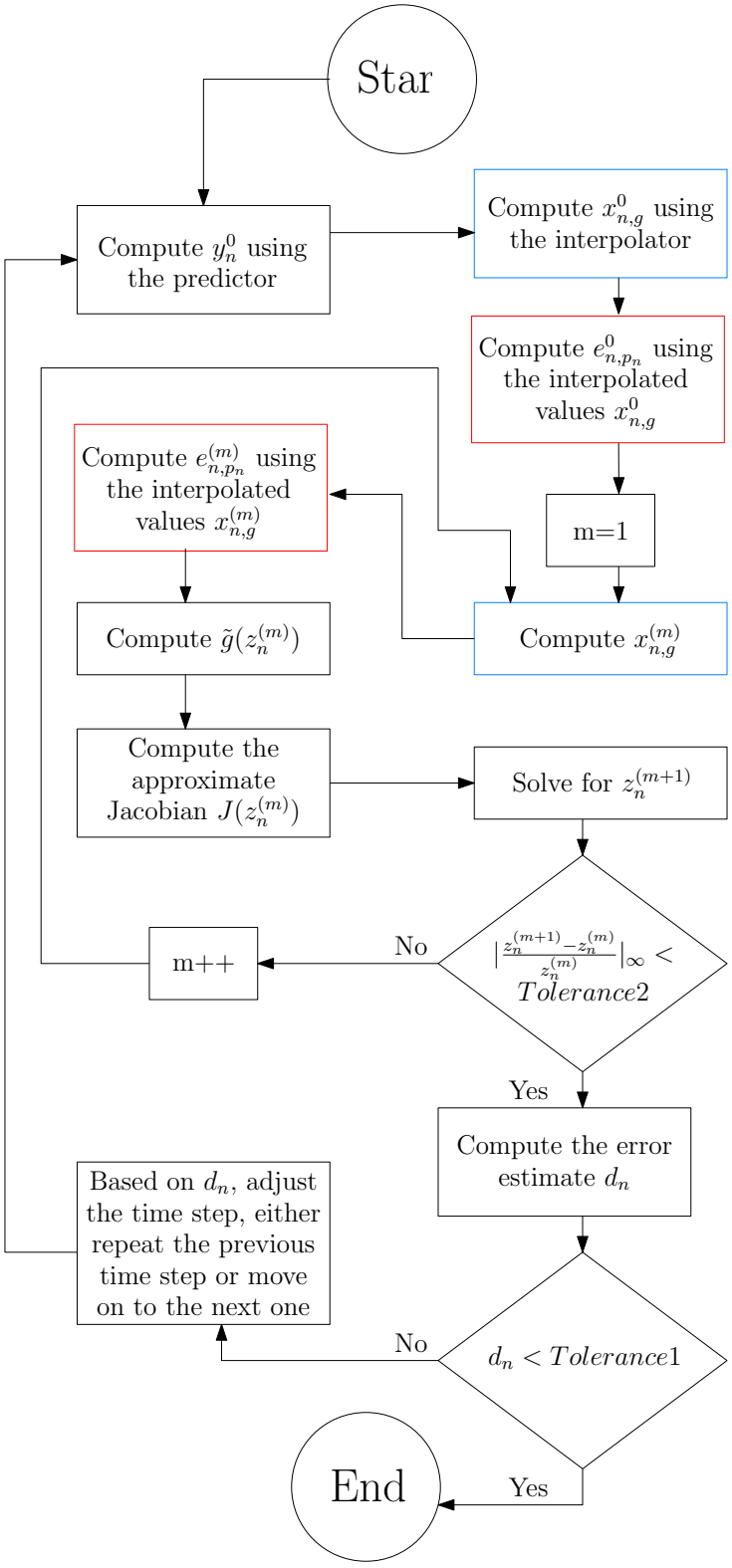

Figure 5.23: Flowchart of the moved Jacobian interpolation-based method for one time step. The blue blocks have the interpolation functions, the red blocks are related to controller computations, and the gray blocks are part of the continuous solver.

5.3.2 Simulation results

In this section, three test systems are considered to assess the accuracy and performance of MJIBM in comparison with the SRM and IBM:

- The two-state system under control by a single integral controller.
- The two-state system under control by multiple integral controllers.
- The kundur system.

5.3.2.1 Single integral controller

In this section, a test system containing two differential equations under the control of a single integral controller is considered.

The simulation results for MJIBM in comparison with IBM and SRM are illustrated in Figs. 5.24 and 5.25 for the stable and unstable systems, respectively. In terms of accuracy, as is visible, no difference can be detected between the solution trajectories obtained with all 3 methods.

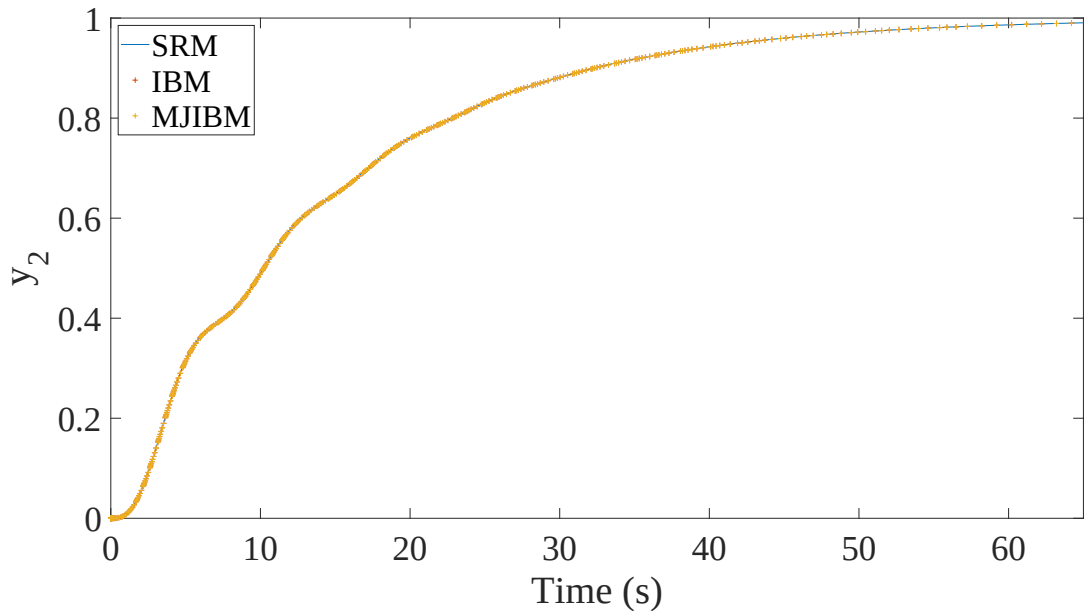

Figure 5.24: Output results for the stable system with one integral controller for all three methods, MJIBM, IBM, and SRM

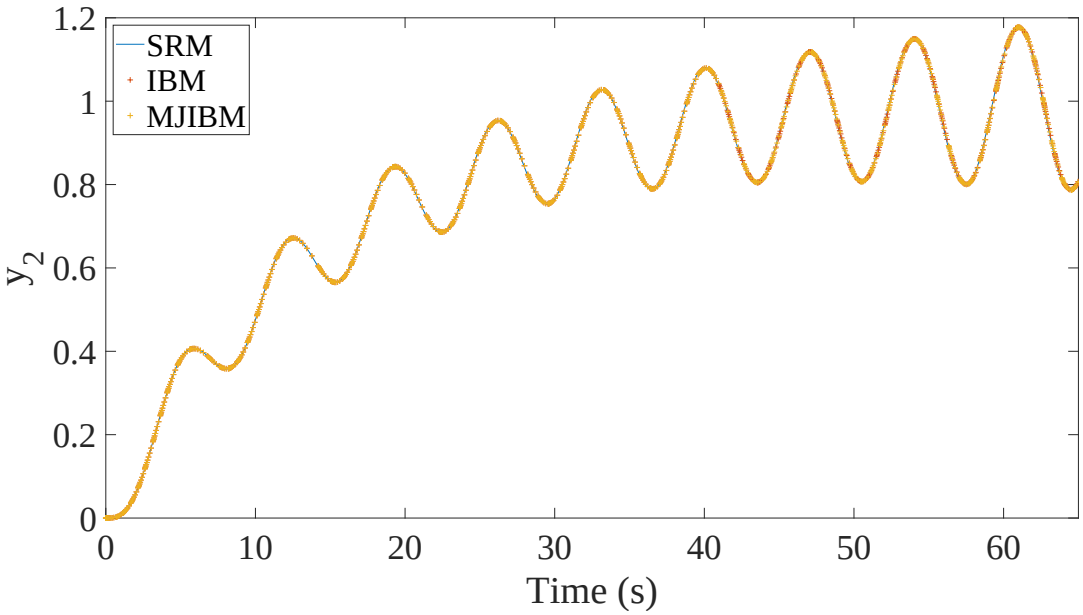

Figure 5.25: Output results for the unstable system with one integral controller for all three methods, MJIBM, IBM, and SRM

The number of Newton iterations required to solve the system is used as a criterion to assess the performance of the proposed method. These numbers are summarized in Table 5.7. It can be seen that MJIBM is slightly faster than IBM, and consequently, significantly faster than SRM for both stable and unstable scenarios. Since the step sizes taken by MJIBM and IBM are almost the same, as shown in Fig. 5.26 and Fig. 5.27.

Table 5.7: Performance comparison between MJIBM, IBM, and SRM for a system containing one Integral controller in terms of the number of Newton iterations

System parameters	SRM	IBM	MJIBM
Stable case	5473	1133	1031
Unstable case	5333	2403	2372

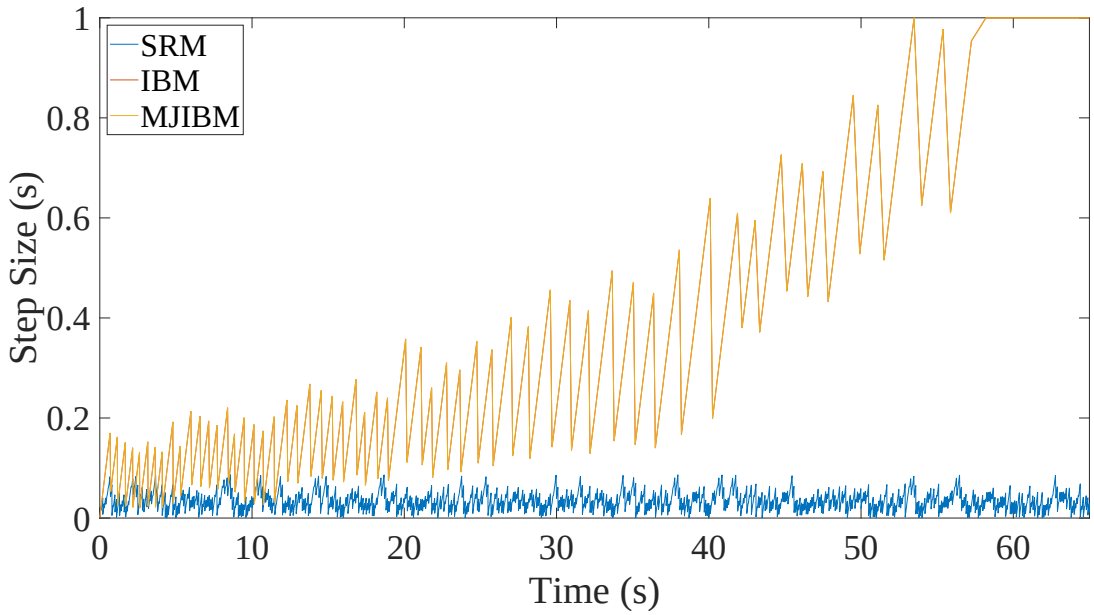

Figure 5.26: Step size results for the simulation of the stable system with the integral controller using all three methods, MJIBM, IBM, and SRM

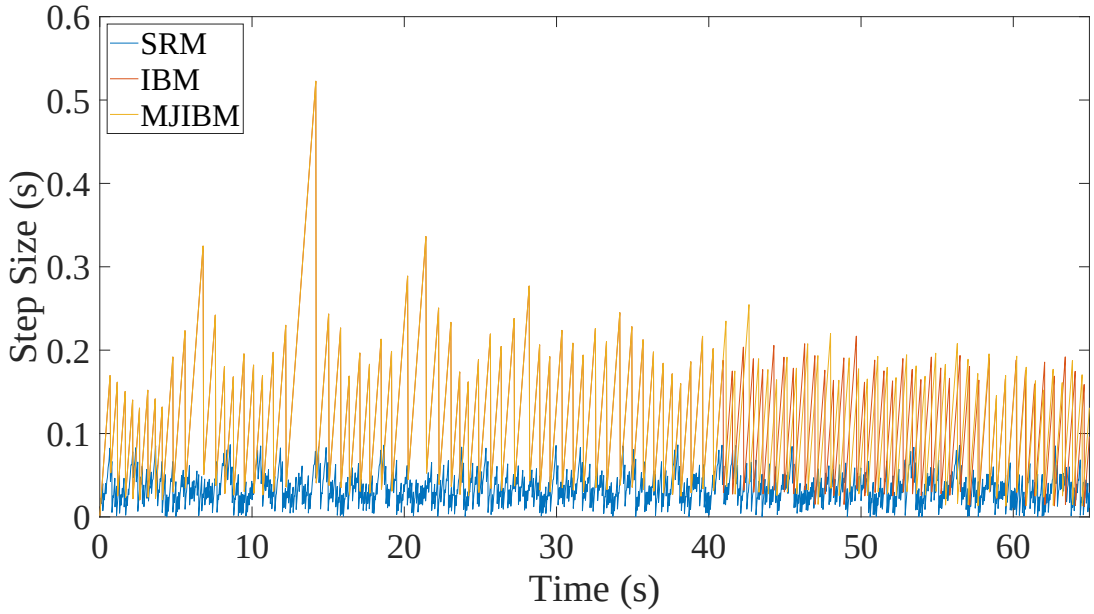

Figure 5.27: Step size results for the simulation of the unstable system with the integral controller using all three methods, MJIBM, IBM, and SRM

5.3.2.2 Multiple integral controllers

In this section, the same continuous system under control by three integral controllers is considered as illustrated in Fig. 5.28. It is assumed that the controllers have a similar sampling rate T equal to 1 ms. Therefore, their input is the same for all the samplings. The aim of this study is to showcase the performance increase happening for MJIBM compared to IBM when multiple controllers have the same inputs. The reason is that MJIBM extends the state variable once for all the identical inputs, while IBM

adds 3 since the outputs are different due to different parameters of the controllers. For example, the gains of controllers are considered equal to 0.05, 0.06, and 0.07 in this case.

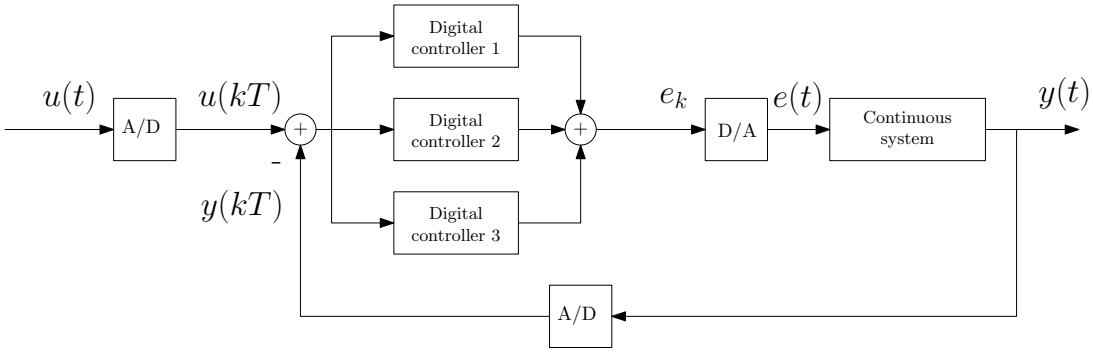

Figure 5.28: A scheme of a continuous system under control by 3 digital controllers

The results using all three methods are depicted in Fig. 5.29 and Fig. 5.30 for stable and unstable systems, for 265 s of simulation. The elongated simulation time highlights the performance gap between the methods more vividly. It can be seen that IBM and MJIBM have similar accuracy.

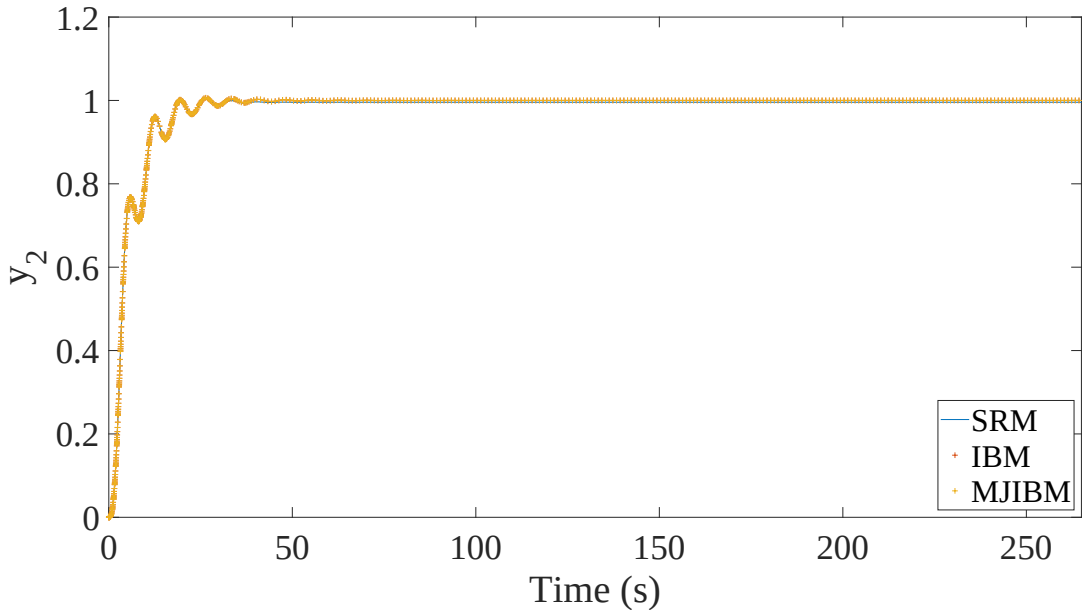

Figure 5.29: Output results for the stable system with three integral controllers for all three methods, MJIBM, IBM, and SRM

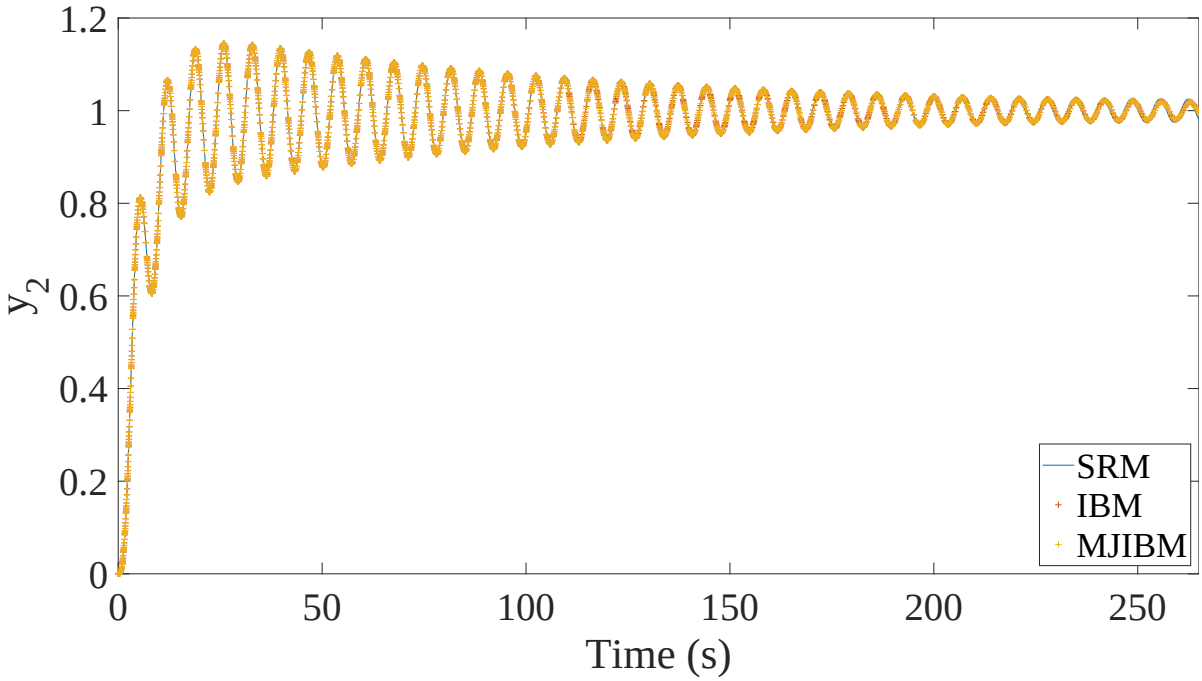

Figure 5.30: Output results for the unstable system with three integral controller for all three methods, MJIBM, IBM, and SRM

The average runtime for IBM and MJIBM is summarized in Table 5.8. It can be seen that MJIBM is more than 20 percent faster than IBM since it solves a smaller system in each Newton iteration. Finally, the step size results for the stable and unstable cases are depicted in Fig. 5.31 and Fig. 5.32, respectively, showing similar time stepping once again.

Table 5.8: Performance comparison between MJIBM and IBM for a system containing three integral controllers in terms of the runtime

System parameters	IBM	MJIBM
Stable case	0.89 s	0.64 s
Unstable case	1.54 s	1.25 s

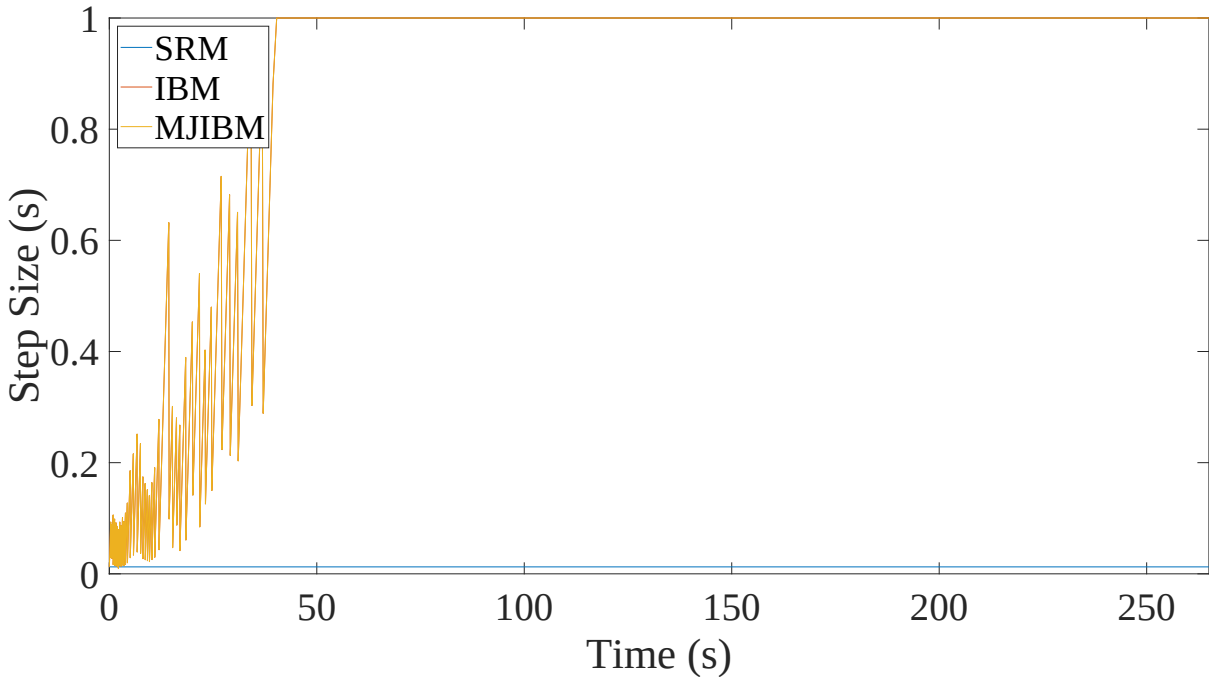

Figure 5.31: Step size results for the simulation of the stable system with three integral controllers using all three methods, MJIBM, IBM, and SRM

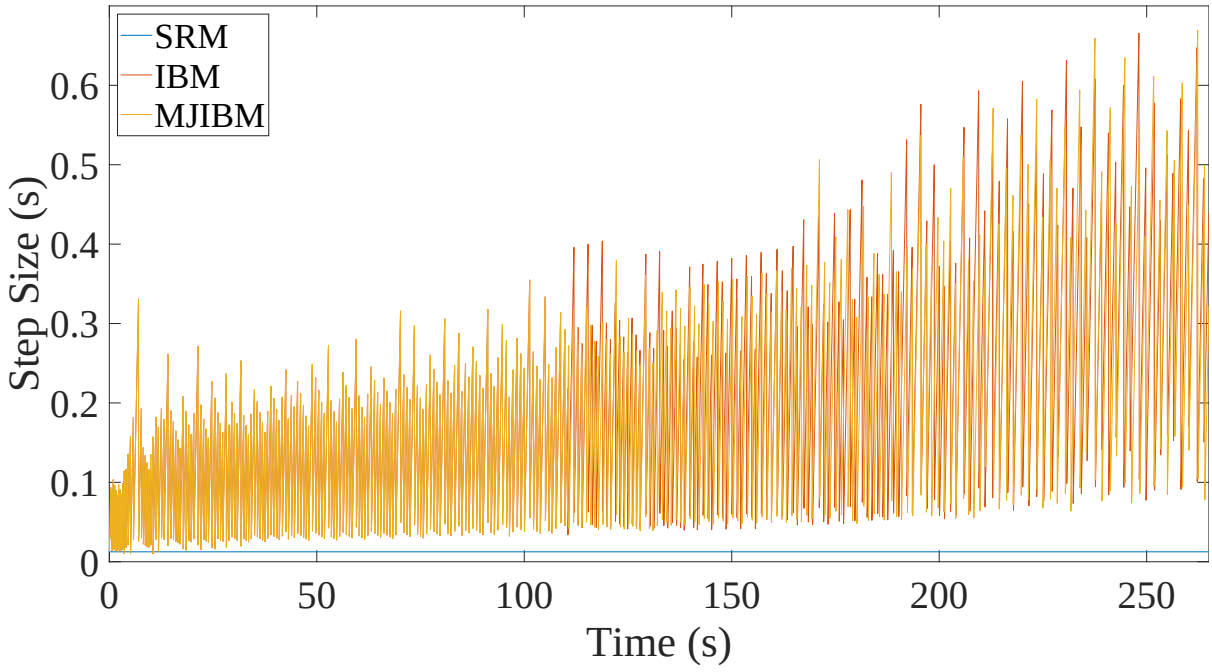

Figure 5.32: Step size results for the simulation of the unstable system with three integral controllers using all three methods, MJIBM, IBM, and SRM

5.3.2.3 Kundur system

Here, we consider the Kundur system with 8 digital controllers, one digital AVR, and one governor for each synchronous generator. The sampling times of digital governors and digital exciters for generators

1 to 4 are considered 210, 220, 230, 240, and 41, 42, 43, 44 ms, respectively.

We consider a step change reduction in the active power consumption of the load on bus 7 $\Delta P_7 = -300$ MW at $t = 1$ s, which is restored at $t = 12$ s. The bus 1, 4, and 9 voltages, speed deviation, AVR, and governor signal of generators are illustrated in Figs. 5.33, 5.34, 5.35, and 5.36, respectively. It is shown that the simulated results are almost identical.

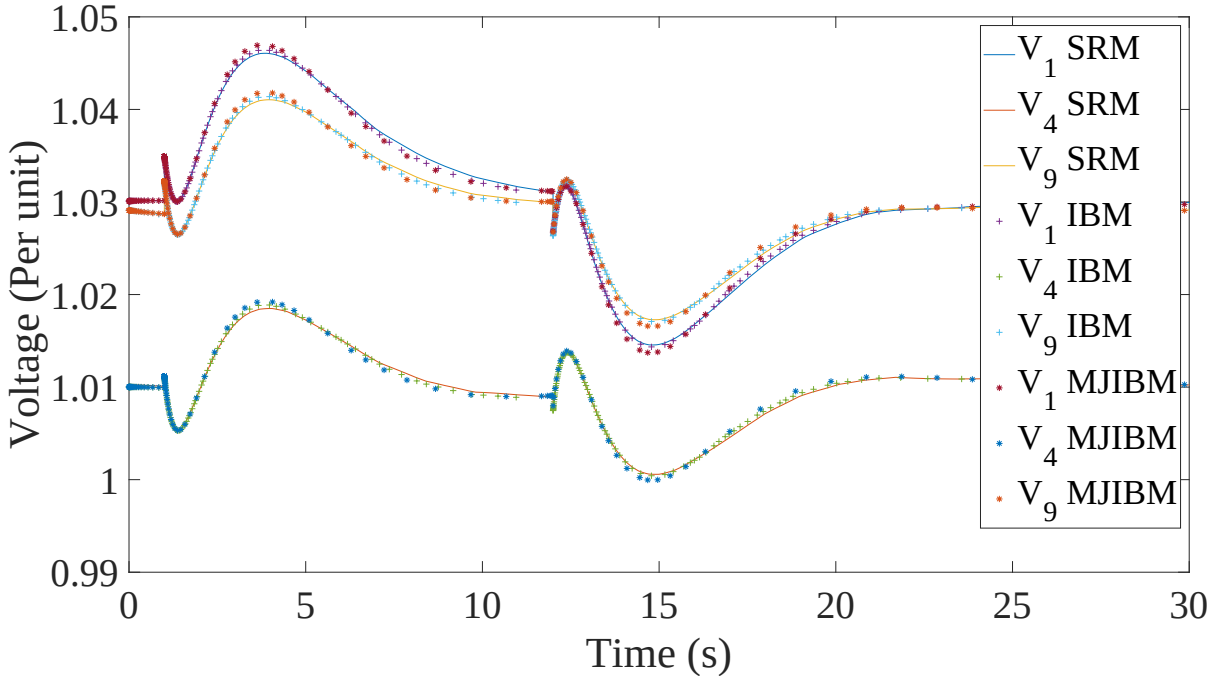

Figure 5.33: Bus 1, 4, and 9 voltage magnitude of the Kundur system simulated using all three methods, MJIBM, IBM, and SRM

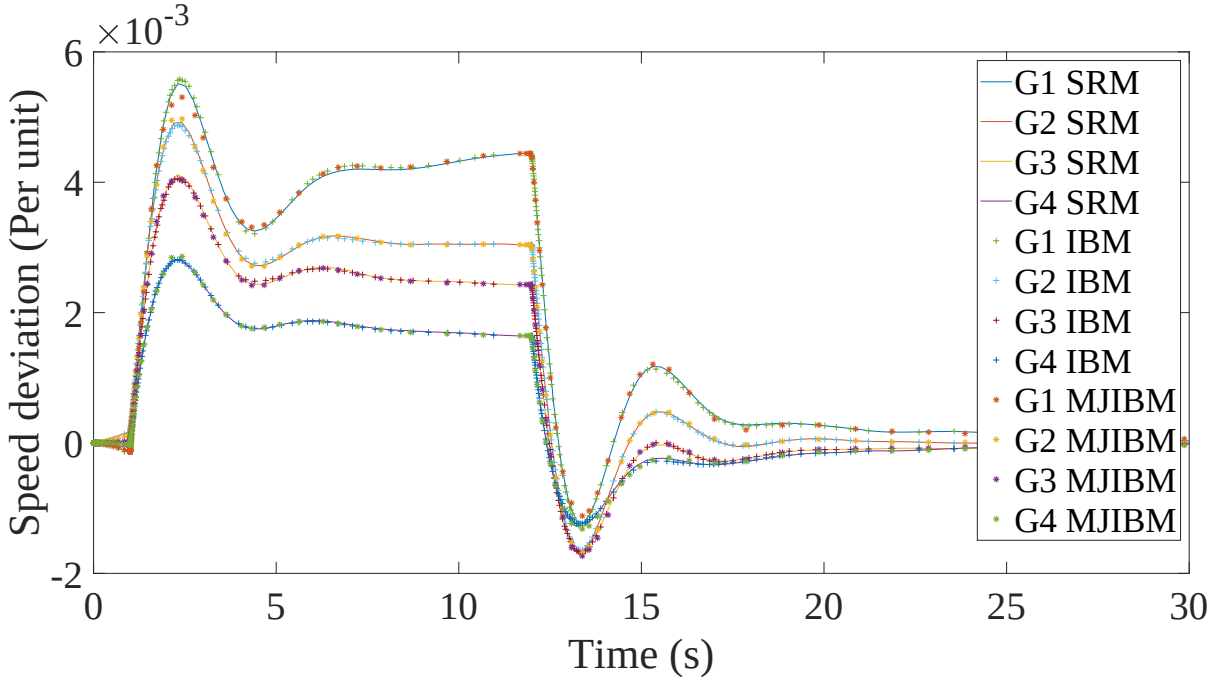

Figure 5.34: Speed deviation of the generators of the Kundur system simulated using all three methods, MJIBM, IBM, and SRM

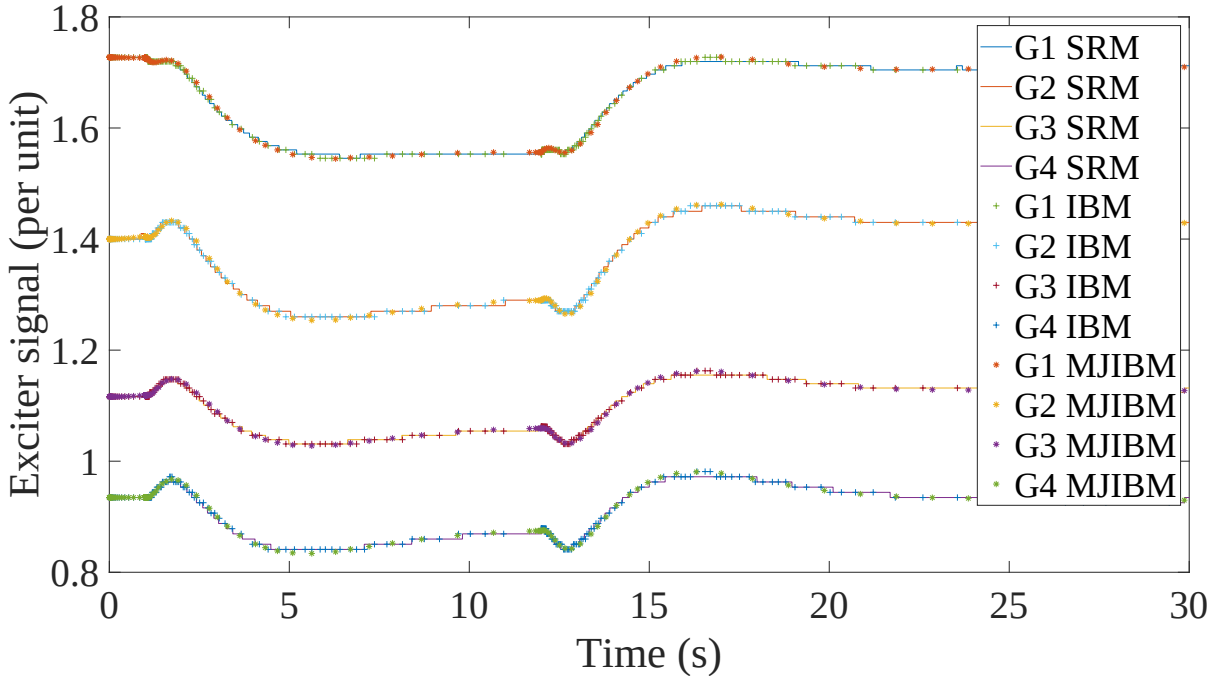

Figure 5.35: The output signal of the digital excitation system of the generators of the Kundur system simulated using all three methods, MJIBM, IBM, and SRM

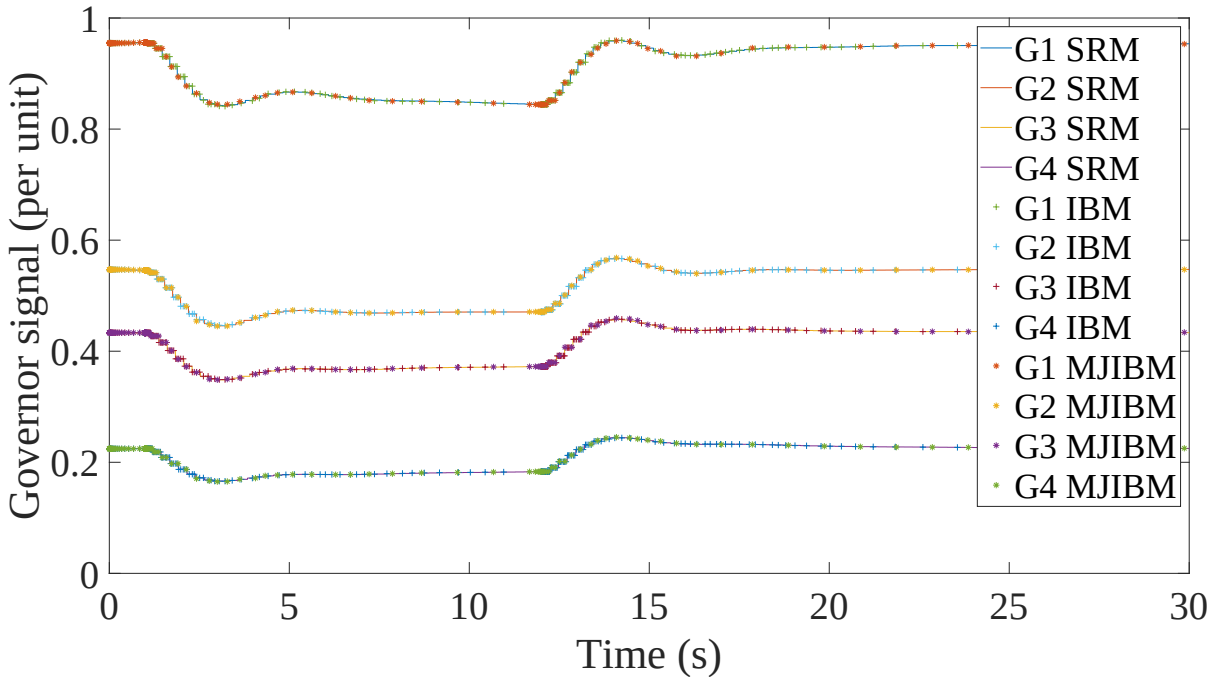

Figure 5.36: Output of the digital governor of the generators of the Kundur system simulated using all three methods, MJIBM, IBM, and SRM

The performance of MJIBM is compared using the number of Newton iterations and the execution time required to solve the Kundur test network, and the results are summarized in Table 5.9. It should be noted that all the run times reported are the average time of 5 runs. Again, it can be seen that IBM is much faster than SRM (about 12 times), as justified by the time-step size shown in Fig. 5.37. Additionally, MJIBM outperforms IBM in both runtime and Newton iterations, as it is able to reach the maximum time step size limit and maintain it for the remainder of the simulation, whereas IBM generates more error estimates, which often necessitate reducing the step size. This leads to MJIBM being about 18 percent faster than IBM.

Table 5.9: Performance comparison between IBM and SRM for Kundur test system

Method	Number of Newton iterations	Run time (s)
SRM	13598	45.4
IBM	642	2.41
MJIBM	461	1.98

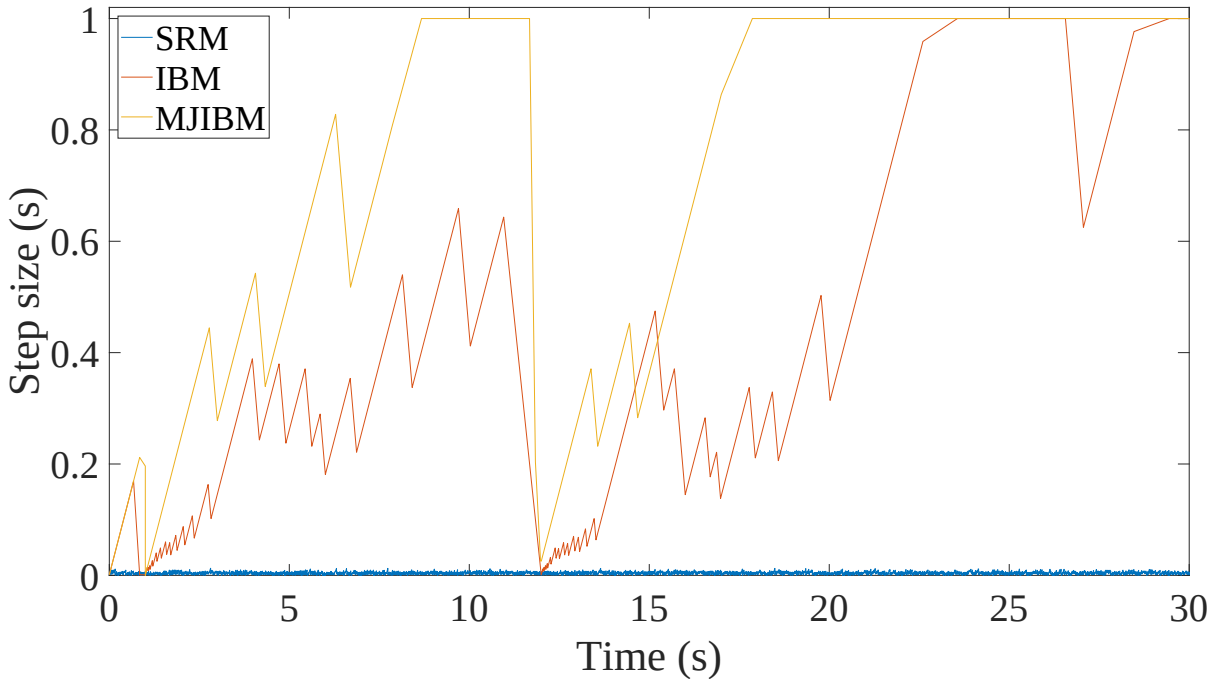

Figure 5.37: Step size results for Kundur system simulation using all three methods, MJIBM, IBM, and SRM

5.4 Summary

In this chapter, extensions and modifications introduced to IBM were discussed to make it suitable for different applications. The highlights can be listed as follows:

- The traditional simplified simulation method loses accuracy facing multiple fast digital controllers since it cannot shift more than one event per controller per time step to the end of the time step.
- The simplified IBM was proposed for simplified simulation of digital controllers, which is able to shift all the controllers to the end of the time step. It also utilizes interpolation to have a good estimate of the feedback coming from the system, leading to an accuracy improvement (see Fig. 5.8).
- IBM struggles to simulate systems with computationally demanding controllers. The reason is that IBM relies on calling the controllers many more times compared to SRM.
- The light version of IBM was proposed, which is able to handle computationally demanding digital controllers faster than both SRM and IBM. This is possible by reducing the size of the system to be solved at each time step by extending the state variables vector only for the last output, and calling the controller only once per time step per sampling.
- Consequently, LIBM is able to simulate systems with computationally heavy controllers faster than both SRM and IBM by 20 and 40 percent, respectively (see Table 5.5).

Also, the moved-Jacobian IBM was proposed that extends the state variables vector based on the controller inputs instead of its outputs. The highlights of MJIBM can be listed as follows:

- MJIBM forms the extended part of the state variables vector on the controller inputs instead of its

outputs. This means that no new type of variable is introduced to the solver. In other words, the same variables are added to the vector only at different points in time. It was shown that MJIBM is about 18 percent faster than IBM (see Table 5.9).

- MJIBM solves a smaller system compared to IBM if the controllers have the same inputs, while IBM always extends the state variables vector by unique controller outputs. It was shown that for such a system, MJIBM is at least about 19 percent faster than IBM (see Table 5.8).
- No element of controllers is included in the solver since the controller inputs are the continuous system's state variables only at different points in time.

6 Modelica Implementation of IBM

Modelica is a powerful open-access tool for modeling and simulating the dynamics of systems governed by differential-algebraic equations (DAEs). It is an object-oriented, equation-based programming language that facilitates acausal modeling, supporting ease of use and the reuse of components [66]. Like many other simulation environments, Modelica typically employs the step reduction method (SRM) to handle discontinuities [16]. While SRM is sufficiently fast for systems with only a few discontinuities, its performance degrades significantly in scenarios involving numerous discontinuities, such as systems incorporating multiple digital controllers. In such cases, integrating the interpolation-based method (IBM) into Modelica could enhance computational efficiency while preserving the advantages of the Modelica language.

However, it would be challenging since implementing IBM requires significant modifications to the solver, from time stepping to Jacobian computation. This section proposes a fixed-step IBM implementation using Modelica for the dynamic simulation of systems containing digital controllers. The method is devised in a way that all the changes required are implemented in the controller block without the need to modify the solver. Therefore, the remainder of the system remains the same as before, and the user only needs to replace the controller with its IBM equivalent.

6.1 Interpolation

Fig. 6.1 shows the two-state system under control by an integral controller. The goal is to build the IBM version of the same system as illustrated in Fig. 6.2, where the interpolation and the extended variables, i.e., the controller outputs at the sampling points of time, are defined in the IBM block of the controller. In this way, only the controller and the sampler are replaced with the IBM controller, while the remaining system and simulation environment are the same.

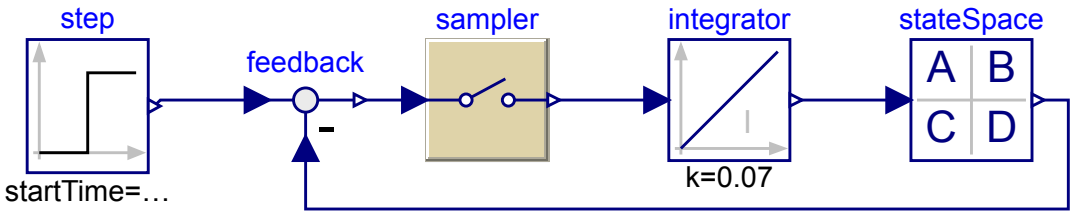

Figure 6.1: A continuous system under control by a digital integral controller modeled in Modelica

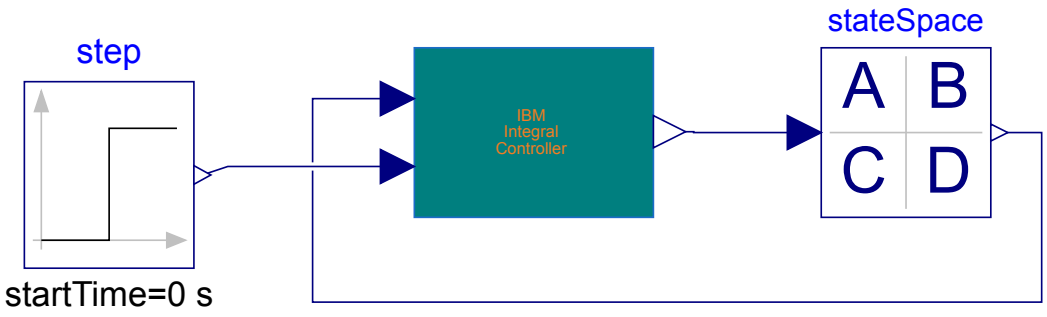

Figure 6.2: A continuous system under control by the IBM version of digital integral controller modeled in Modelica

To do so, the first challenge to tackle is the interpolation formula implementation. As an example, let's assume a first-order interpolator as follows:

$$\boldsymbol{y}_{n,g}^{(m)} = \boldsymbol{w}_{n}^{(m)}(t_{n,g}) = h_{n,g}\dot{\boldsymbol{y}}_{n-1} + \boldsymbol{y}_{n-1} \tag{6.1}$$

where $h_{n,g} = t_{n,g} - t_{n-1}$ is the step of the g-th sample within the time step h_n from the beginning of the time step. As evident, the feedback from the system and its derivative $\dot{\boldsymbol{y}}_{n-1}$ are required for the interpolation function.

There are two approaches to provide the derivation of the feedback variables to the controller. In the first approach, a derivative block can be placed between the feedback and the controller input. Therefore, the controller inputs will double by the number of its inputs. In this approach, the feedback from the system is connected directly to the IBM controller as well as its derivative, which will be computed outside of the controller block. This approach calls for rewiring and adding derivative blocks to the system design.

The second approach performs the derivative computation inside the controller block using a differentiation formula, removing the need for rewiring the system blocks. In this path, the derivative of the inputs will be built inside the controller block using the following formula:

$$\dot{\boldsymbol{y}}_n = \frac{\boldsymbol{y}_n - \boldsymbol{y}_{n-1}}{h_n} \tag{6.2}$$

The issue with this approach is that the accuracy of the derivative provided is limited to the size of the time step taken. This approach is, however, simpler to implement since the calculations happen inside the controller block. Therefore, the rest of the system remains intact while choosing to use IBM. Nevertheless, the accuracy of the derivatives can always be enhanced by adjusting the fixed time step size.

Finally, it should be noticed that an IBM equivalent of the digital controller doesn't have the sampler. This means that the solver will see the system as a continuous system. In other words, no time events are imposed by the IBM controller.

6.2 Time stepping

As mentioned, the overall system with the IBM controller replaced the digital controller will be seen as a continuous system by the solver. Therefore, there is a need for fake events to pause the solver and compute the interpolations, and add the controller outputs as the state variables. These fake events act as the fixed time steps that provide a chance to run an algorithm of feedback derivative computation, counting the number of controller sampling actions, running the interpolation calculation, and adding the extended state variables.

The time step size can be chosen by the user by modifying the parameter of a `Clock` function. It should be noted that there is no need for the user to select a time step size that is an integer multiple of the controller sampling rate to make sure the time step lands on the samplings. Employing a ZOH approach, the output of the controllers remains constant between the samplings.

Time events forced on the solver are different from the time events imposed by the digital controller. For example, simulating a digital controller with sampling time $T = 10$ ms using SRM will impose time events every 10 ms. On the other hand, with the IBM controller, the only time event happens at the time step. In other words, the IBM version of the controller is handled as a continuous model, and no time events are created in between the time steps because of the controller's discrete nature. Therefore, the solver stops only at the fixed time steps specified by the user.

After a fake event caused by the clock function imposed, first, the derivative of the feedback will be calculated. Then, using auxiliary discrete variables, the number of sampling actions falling inside the fixed time step for the controller will be counted. Now, the interpolation formula will be executed, and its output will be used to compute the controller outputs at sampling times inside the time step. These outputs can be saved in a new vector of variables that will be extended (automatically by the solver) to the previous system state variables. Since the size of this vector must be defined before running the algorithm, an oversized array can be defined to have enough space to save all the controller outputs. This can be estimated since both the time steps and the controller sampling rate are fixed.

In case of having multiple controllers, it should be noted that all the clocks in all the controller blocks must be synchronised, otherwise each one imposes a different size of fixed time step, which limits the time step overall. This may cause accuracy loss and limit the scalability of the approach if the difference between controller sampling rates is relatively large. The reason is that a controller might have a large number of samplings falling in one time step, while the other ones have only a few.

6.3 IBM-Modelica implementation

Based on the technicalities discussed previously, the IBM version of the integral controller can be modeled as the following listing:

Listing 6.1: The IBM-Modelica implementation of the digital integral controller

```
1 Clock c1=Clock(1,20);
2 algorithm
3   when Clock(c1,"ImplicitTrapezoid") then
4   i:=0;
```

```
5   prett:=pre(tt);
6   ttfloor:=floor(tt/T);
7   prettfloor:=floor(prett/T);
8   ttfloordiff:=ttfloor - prettfloor;
9   ydot=y-pre(y)/tt-pre(tt)
10  while i < ttfloordiff loop
11    i:=i + 1;
12    w:=(prettfloor*T+i*T-prett)*pre(ydot)+pre(y)+
13    ((prettfloor*T+i*T-prett)^2/(tt-prett)^2)*(y-pre(y)-(tt-prett)*pre(ydot));
14    e:=e + Ki*T*(u - w);
15    earray[1,i]:=e;
16    if i==ttfloordiff then
17      earray[1,maxstep]:=earray[1,i];
18    end if;
19  end while;
20  equation
21 z2=earray[1,maxstep];
```

The first and third lines of the code impose the time events at fixed time steps to provide the opportunity to run the algorithm containing the IBM approach. The variable `tt` holds the value of time t_n at each time step. Therefore, the fifth line defines the previous time step t_{n-1}. Lines 6 to 8 have the duty of making auxiliary variables for counting the number of controller samples in the time step to set the number of times the loop has to repeat. Depending on the fixed time step size, there may be a different number of controller actions in each time step. The 9th line computes the derivative of the feedback.

Line 12 accounts for a second-order interpolation function of the feedback from the dynamical system at the sampling points of the controller, where $ydot$ and y address the derivative of the feedback and the feedback itself, respectively. As can be noted from the code, the size of $h_{n,g}$ is considered a variable that depends on the number of samplings that fall into the time step. Line 13 has the integral controller equation, and the next line saves the calculated controller output in an array, which represents $\boldsymbol{z}_{n,2}$ defined in (4.14).

It should be noted that the size of the array `earray` defined by the parameter `maxstep` for saving the controller outputs is larger than it needs to be, since it relieves the user from modifying it every time based on the sampling rate of the controller and fixed time step. Therefore, lines 15 to 17 have the task of saving the last controller output in the last cell of the array to be fed to the output of the controller block using line 20. In this approach, the user can actually use any solver with any settings, and the code itself forces the time events to the solver at the times required and runs the IBM algorithm.

6.4 Simulation results

In this section, two test cases are used to showcase the accuracy and performance of the IBM implementation in Modelica versus the regular Modelica modeling of the digital controller:

1. An integral controller controlling a state space system with two ordinary differential equations (ODE).

2. A single-machine infinite-bus (SMIB) system under control by digital excitation and digital governor systems.

6.4.1 Integral controller

For the first test system, a continuous state-space system represented by the following set of ODEs is considered:

$$\underbrace{\begin{bmatrix}\dot{y_1}\\ \dot{y_2}\end{bmatrix}}_{\dot{\boldsymbol{y}}} = \underbrace{\begin{bmatrix}a & b\\ -b & 0\end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix}y_1\\ y_2\end{bmatrix}}_{\boldsymbol{y}} + \underbrace{\begin{bmatrix}-b\\ 0\end{bmatrix}}_{\boldsymbol{B}} e(t) \tag{6.3}$$

where an integral controller is used to put the system under control:

$$e_k = e_{k-1} + K_I T(u(kT) - y_2(kT)) \tag{6.4}$$

where it has the gain of $K_I = 0.07$ and sampling time of $T = 0.01$ s. The schematic of the test system is sketched in Fig. 6.1. The Modelica-IBM equivalent of the system can be modeled as shown in Fig. 6.2. The results of the simulation of both systems for the second output of the continuous state-space system using the RK2 solver are shown in Fig. 6.3.

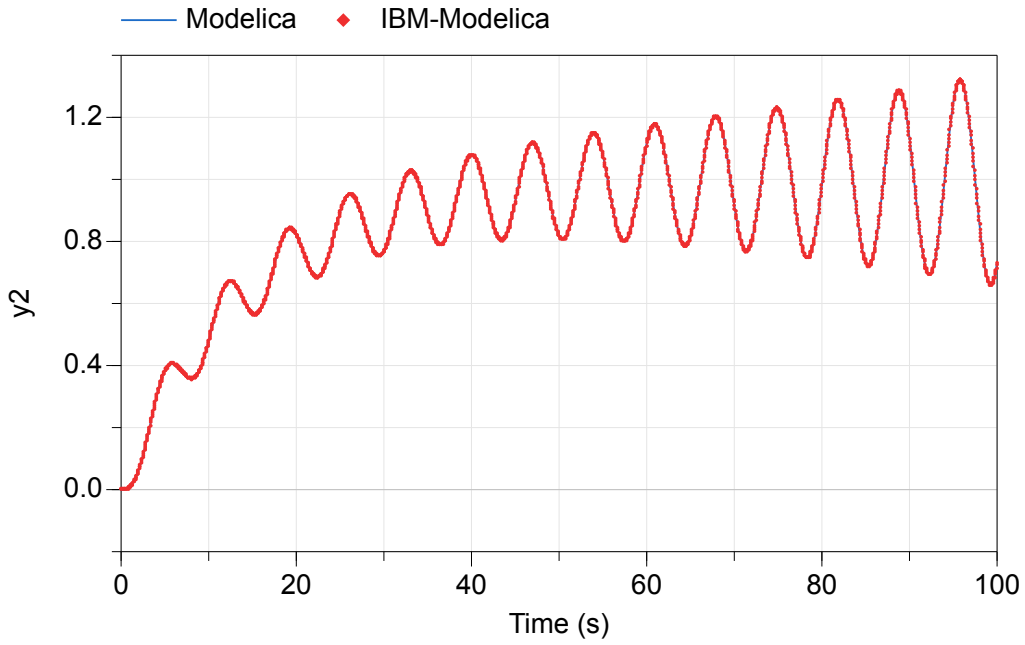

Figure 6.3: Simulation results for regular Modelica modeling and IBM-Modelica modeling of y_2 using the RK2 solver

The CPU runtime for the simulation of both controller models using the RK2 solver is listed in Table. 6.1. The fixed time-step for the IBM-Modelica simulation is considered 0.05 s. The fixed time step size for the regular Modelica controller model is limited to 0.01 s, which is equal to the controller sampling rate. It can be seen that IBM-Modelica is 3 times faster than regular Modelica modeling.

Table 6.1: Test System 1: Performance results using the RK2 solver

Model	CPU Runtime (s)
Modelica	**0.022**
IBM-Modelica	**0.007**

The same simulation is performed again, this time with the variable step solver Dassl with the default settings. The accuracy is the same as before, as can be deducted from Fig. 6.4. The performance results are listed in Table. 6.2. As can be seen, solving the regular Modelica system with the Dassl solver is significantly slower compared to RK2. The reason is that the solver is constantly trying to reduce the time steps to land on the time events of the controller, while it wasn't the case for the RK2 solver since the fixed time steps were chosen to land perfectly on the controller samplings. However, the same solver has great performance results for the IBM-Modelica model.

Table 6.2: Test system 1: performance results using the Dassl solver

Model	CPU Runtime (s)
Modelica	**0.16**
IBM-Modelica	**0.005**

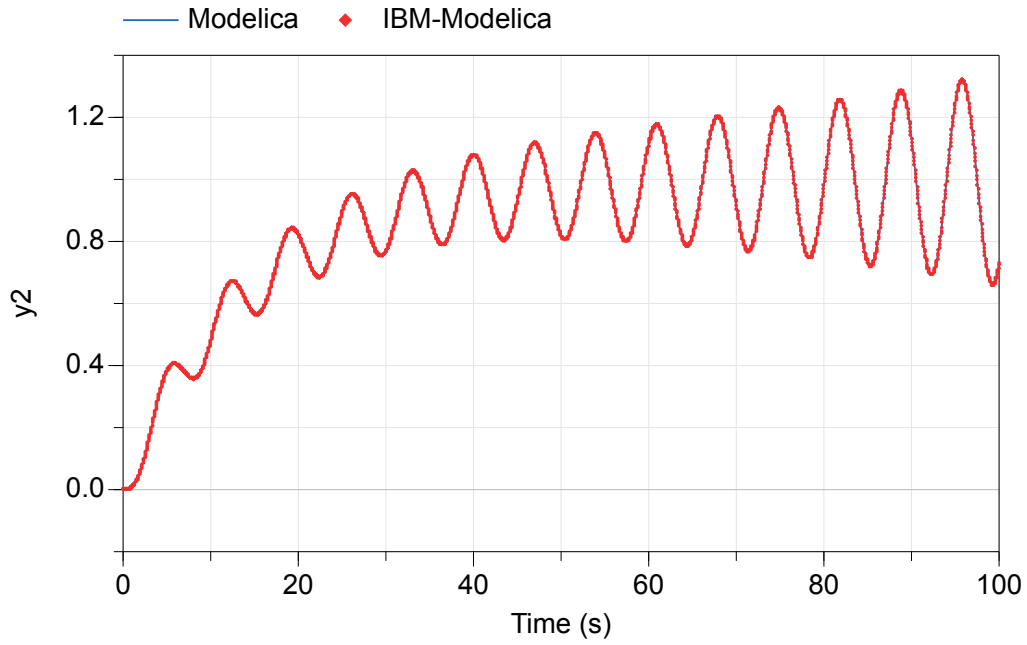

Figure 6.4: Simulation results for regular Modelica modeling and IBM-Modelica modeling of y_2 using the Dassl solver

6.4.2 SMIB

For the second test system, a fourth-order synchronous generator connected to an infinite bus is considered as illustrated in Fig. 6.5. The open-source library OpenIPSL, developed using the Modelica language for

power system dynamic studies, is utilized to construct the test system [67]. The generator is under control by a digital AVR and a digital governor, both having a sampling rate equal to 1 ms. The same system with IBM modeling is shown in Fig. 6.6. It should be noted that the output of the digital AVR first goes to an exciter (continuous device), modeled by a first-order block, and then to the generator.

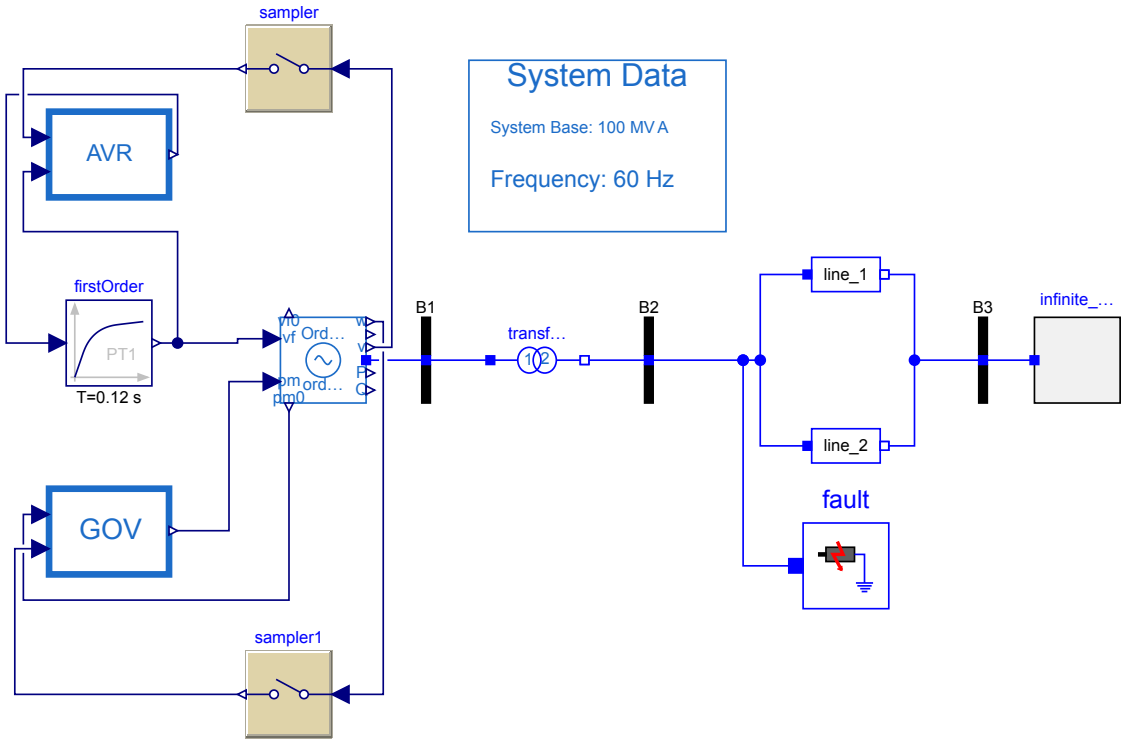

Figure 6.5: The schematics of the SMIB test system modeled in Modelica

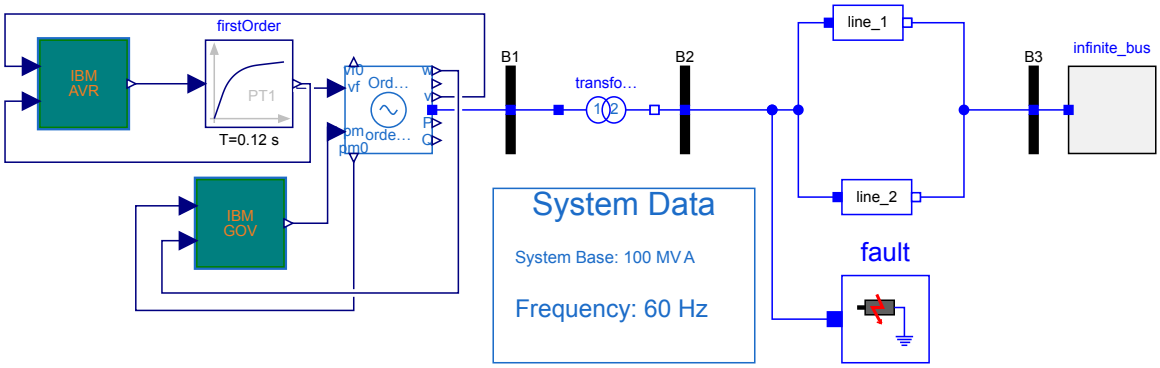

Figure 6.6: The schematics of the SMIB test system with IBM controllers modeled in Modelica

The simulation is performed using the Dassl solver, and the generator voltage, AVR output, and governor output are depicted in Figs. 6.7, 6.8, and 6.9, respectively. As shown, the dynamic response is identical for both models.

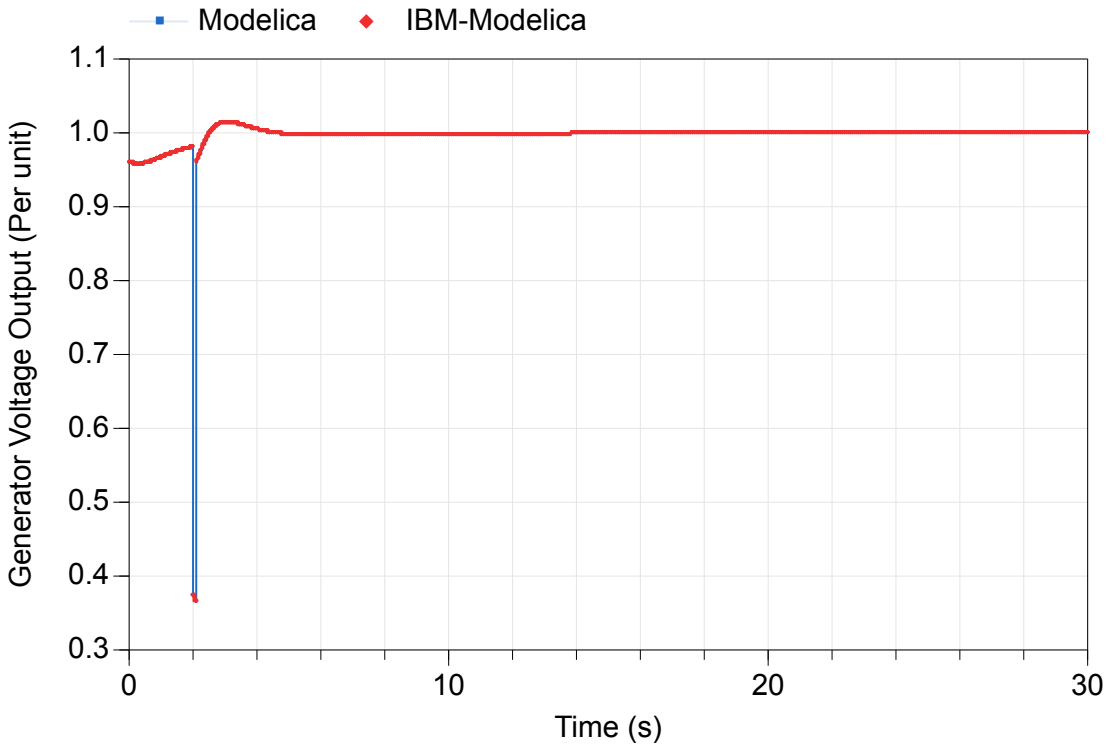

Figure 6.7: Generator voltage output results for both models

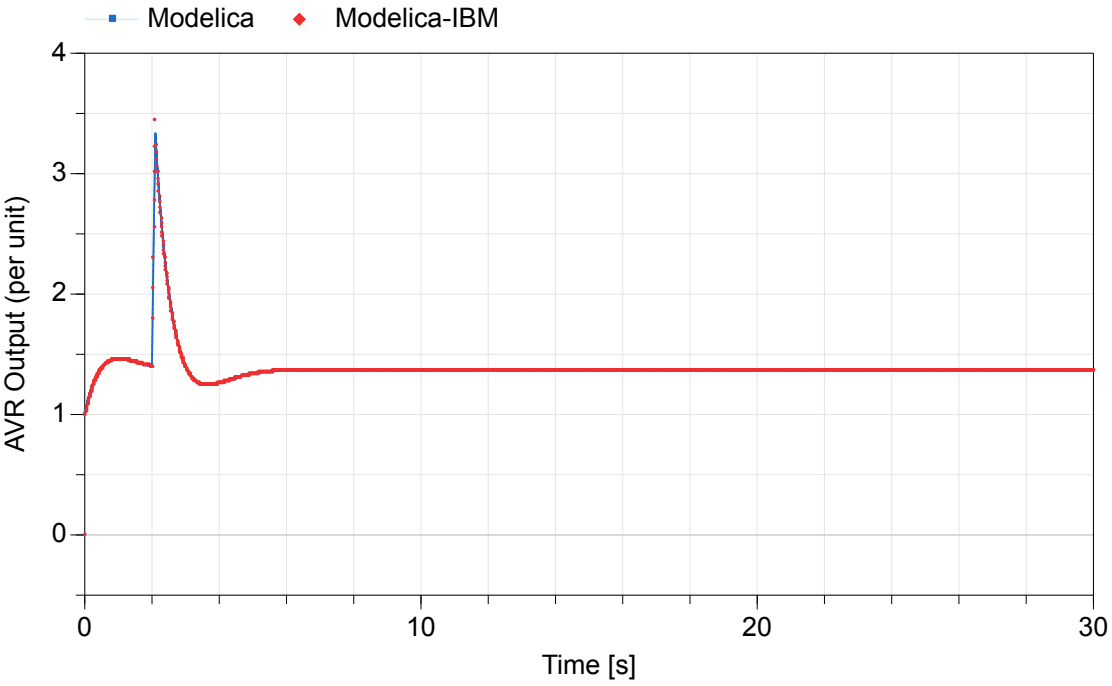

Figure 6.8: AVR output results for both models

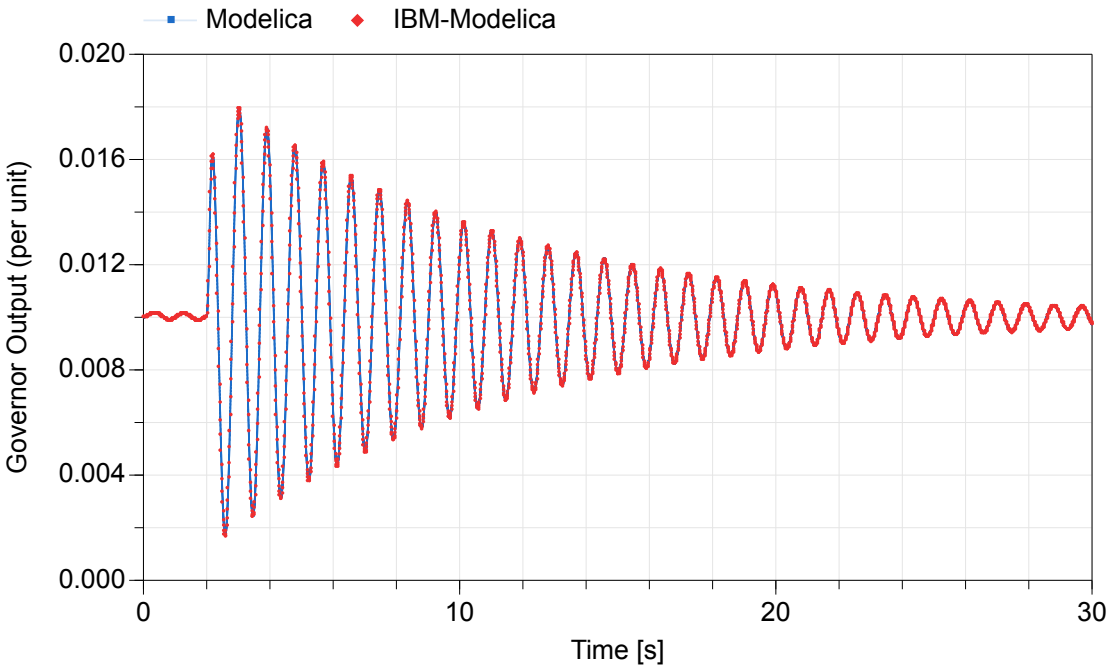

Figure 6.9: Governor output results for both models

The number of controller samples falling within each time step during the simulation is shown in Fig. 6.10. This emphasizes the fact that the user doesn't need to satisfy any condition for choosing the time step size.

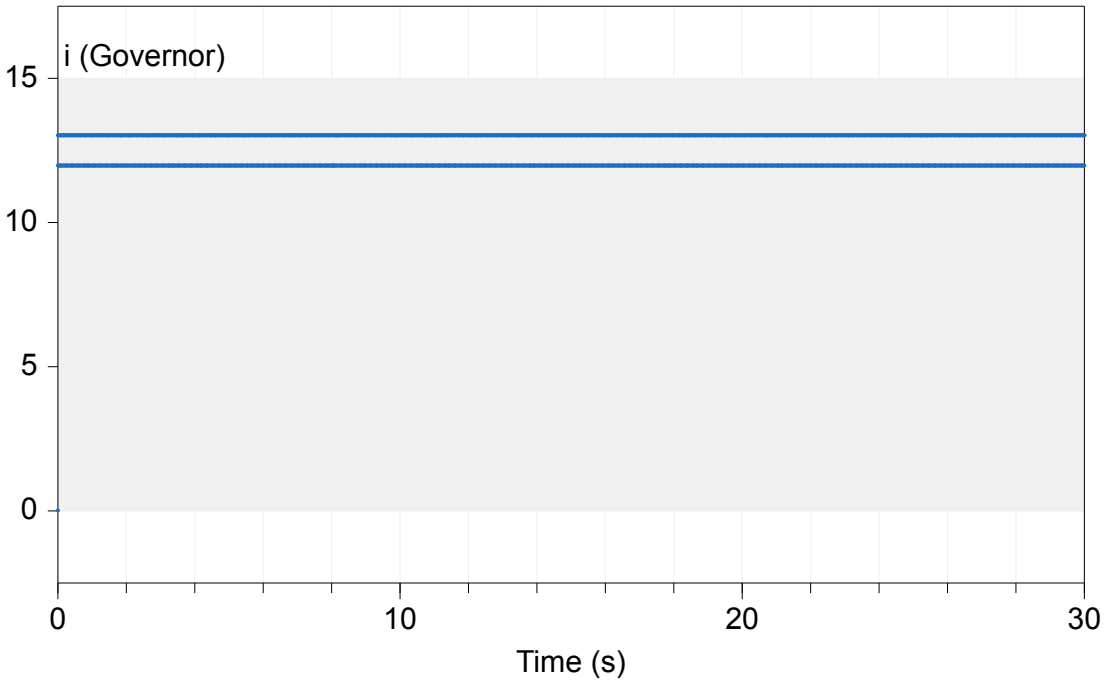

Figure 6.10: Number of Sampling actions of the governor in each fixed time step

The performance results of the simulation of the SMIB system are summarized in Table. 6.3. As can be seen, the IBM-Modelica system is solved almost 3 times faster.

Table 6.3: Test System 2: Performance results using the Dassl solver

Model	Step size (s)	CPU Runtime (s)
Modelica	**0.001**	**0.39**
IBM-Modelica	**0.0125**	**0.151**

6.5 Summary

In this chapter, an IBM-based Modelica implementation is proposed for the fast and accurate simulation of systems with digital controllers. Two case studies were used to showcase the accuracy and performance of the method compared to regular Modelica modeling. It was shown that the IBM-Modelica method has similar accuracy to the regular Modelica modeling, while the performance increases.

The method is designed in a way that the user only needs to replace the digital controller and the sampler with their IBM equivalent, and no further changes are needed. Also, the user is able to choose any fixed time step, whether they land on the controller samplings or not, by altering the clock parameter inside the controller block.

7 Summary of Findings

7.1 Summary of work and main contributions

In this work, the integration of digital controllers in time-domain power system dynamic simulations and the challenges that may arise are discussed. Also, a range of novel methods for the dynamic simulation of power systems with multiple digital controllers capable of tackling their challenges has been proposed.

First, different approaches to modeling and simulation of digital controllers were discussed. The impact of continuous and discrete modeling of controllers in power system dynamic simulations on accuracy and performance was demonstrated using case studies. It was shown how the simulation of a system with multiple digital controllers limits the time step, leading to a performance drop and a slow simulation.

Then, the conventional methods capable of simulating power systems alongside the digital controllers have been reviewed. The analysis revealed that the step reduction method becomes very slow when facing systems with multiple digital controllers. For instance, simulating the kundur system with 8 digital controllers was 80 percent slower than simulating it with continuous equivalent controllers (see Table 4.9). Also, simplified simulations fail in accuracy while facing multiple with relatively fast sampling rates. Analog treatment of controllers also suffers from accuracy loss, and it is not able to simulate smart modern controllers, as they cannot be modeled with conventional differential-algebraic equations.

The proposed interpolation-based method (IBM) stands between the step-reduction method and the simplified simulation approach, having a fast yet accurate response. This was achieved by taking large time steps and including the controller outputs in the Newton solver loop, thanks to an interpolation function that estimates the controller inputs at each sampling time of the controller. Multiple case studies were introduced to test the proposed method and compare it against the traditional approaches. For instance, a 16 times faster performance compared to SRM was noticed while simulating the modified Kundur system with 9 digital controllers.

A decoupled version of IBM was developed that solves the controller equations and the system's DAE separately, and uses the results of each process for the other. It was demonstrated that simulating the test system using DIBM leads to a 7 percent performance enhancement compared to IBM with the most simplified Jacobian.

Although effective for normal digital controllers, it was discussed that IBM may have a slow performance compared to SRM in the case of simulation computationally demanding controllers, as it relies on calling the controllers many more times. For instance, IBM was 14 s slower than SRM simulating a 3-bus system with a computationally heavy controller included. Thus, LIBM was proposed, appropriate for the simulation of systems containing such controllers. The light IBM approach updates the controller output only at the first Newton iteration and includes only the last controller output found in the time step in each extended state variable vector. This led to 12 s faster performance compared to IBM simulating the same system.

In addition, a novel simplified simulation method (SIBM) suitable for the simulation of power systems with fast digital controllers has also been proposed. To cover the accuracy loss of the traditional

simplified simulation approach (SSM) in the case of controllers with fast sampling rates, the proposed simplified approach uses an interpolation function to estimate the controller inputs at the sampling points in time, and computes the outputs recursively as they depend on the previous one. It was shown that SIBM is faster than both IBM and SSM by about 53 and 13 percent, respectively, while maintaining better accuracy than SSM.

Another version of IBM has also been proposed, extending the state variable vector by the controller inputs instead of its outputs. This approach has advantages such as not introducing any new type of variable to the solver, and reducing the number of controllers from the solver's perspective if they have the same input. Due to these advantages, it was shown that MJIBM was faster than IBM by 18 percent while simulating the Kundur system.

Finally, a fixed-step implementation of IBM using Modelica has been proposed, making it reachable to others. The approach has been devised in a way that the least modifications are needed to the system design. Only the controllers need to be replaced with their IBM equivalents, and all the extra computations are imposed on the solver without manipulating it. Case studies revealed that simulating the IBM implementation of the controller in Modelica leads to 3 times faster performance for a system with just one integral controller.

7.2 Future work

The presented methods may be expanded and further improved along the following paths:

- IBM can be tested for large-scale simulations to evaluate further and verify the method's accuracy and performance. In this case, limitations may need to be imposed on the method, such as limiting the time step size allowed to be taken by the solver to limit the number of events falling in each time step. In a large-scale simulation with numerous digital controllers, too many events arising in large time steps may impact accuracy.
- The performance and accuracy of SIBM facing computationally demanding controllers can be evaluated. A comparison between SIBM and LIBM in terms of accuracy and performance facing computationally demanding controllers may be interesting.
- A hybrid SRM-IBM approach can be investigated to achieve a comprehensive solution for dynamic simulation of power systems containing both normal digital controllers and computationally demanding ones. The normal controllers can be treated using IBM, while the heavy controllers can be handled by SRM.
- The Modelica implementation of IBM can be expanded to a variable time step approach. This can be achieved by implementing a time-dependent varying clock function that imposes events according to the variable in size time steps taken by the solver. As a suggestion, the clock must act based on a variable that becomes true at each variable time step taken by the solver, instead of imposing fixed time steps on the solver as it is now. This provides the opportunity for extra computations to be carried out in the halts forced by the variable in fake time events.
- The family of the proposed methods can be extended to the EMT simulations of systems with discrete time events for broader applications, catching the transients in different time scales faster.

- Hardware-in-the-loop simulation can be conducted using the proposed methods for validation of the simulation results, and identifying the real-world applications and challenges.

BIBLIOGRAPHY

[1] M. B. Carver, "Efficient integration over discontinuities in ordinary differential equation simulations," *Mathematics and Computers in Simulation*, vol. 20, no. 3, pp. 190–196, 1978.

[2] IEEE, "IEEE recommended practice for excitation system models for power system stability studies," *IEEE Std 421.5–2016.(Revision of IEEE Std 421.5-2005)*, pp. 1–207, 2016.

[3] F. Zhang, M. Yeddanapudi, and P. J. Mosterman, "Zero-crossing location and detection algorithms for hybrid system simulation," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 7967–7972, 2008.

[4] T. Park and P. I. Barton, "State event location in differential-algebraic models," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 6, no. 2, pp. 137–165, 1996.

[5] G. Mao and L. R. Petzold, "Efficient integration over discontinuities for differential-algebraic systems," *Computers & Mathematics with Applications*, vol. 43, no. 1-2, pp. 65–79, 2002.

[6] G. Grabner, R. Kittinger, and A. Kecskeméthy, "An Integrated Runge-Kutta and Polynomial Root Finding Method for Reliable Event-Driven Multibody Simulation," *IFAC Proceedings Volumes*, vol. 36, no. 2, pp. 251–256, 2003.

[7] M. Jafari, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, and P. Aristidou, "Modeling of Digital Controllers in Electric Power System Dynamic Simulations," in *2023 IEEE Belgrade PowerTech*, 2023.

[8] ——, "Methods for incorporating digital controllers in power system dynamic simulations," *Electric Power Systems Research*, vol. 235, p. 110827, 2024.

[9] ——, "Decoupled interpolation-based method for numerical simulation of digital controllers," in *2024 IEEE International Systems Conference (SysCon)*. IEEE, 2024, pp. 1–6.

[10] ——, "A modelica ibm implementation for fast simulation of digital controllers in power systems," in *2024 Open Source Modelling and Simulation of Energy Systems (OSMSES)*. IEEE, 2024, pp. 1–6.

[11] ——, "Handling of computationally demanding digital controllers in power system dynamic simulations," in *2024 3rd International Conference on Energy Transition in the Mediterranean Area (SyNERGY MED)*. IEEE, 2024, pp. 1–5.

[12] ——, "Fast and accurate simulation of smart digital controllers in power system dynamic studies," *Sustainable Energy, Grids and Networks*, vol. 42, p. 101665, 2025.

[13] B. Zupancic, R. Karba, M. Atanasijevic-Kunc, and J. Music, "Continuous systems modelling education–causal or acausal approach?" in *ITI 2008-30th International Conference on Information Technology Interfaces*. IEEE, 2008, pp. 803–808.

[14] IEEE Power System Dynamic Performance Committee, "Dynamic models for turbine-governors in power system studies," IEEE, Tech. Rep., 2013.

[15] M. A. A. Murad, B. Hayes, and F. Milano, "Application of Filippov theory to the IEEE Standard 421.5-2016 anti-windup PI controller," in *2019 IEEE Milan PowerTech*, 2019, pp. 1–6.

[16] S. E. Mattsson, H. Elmqvist, and M. Otter, "Physical system modeling with Modelica," *Control engineering practice*, vol. 6, no. 4, pp. 501–510, 1998.

[17] M. Otter, H. Elmqvist, and S. E. Mattsson, "Hybrid modeling in modelica based on the synchronous data flow principle," in *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No. 99TH8404)*. IEEE, 1999, pp. 151–157.

[18] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[19] D. Ellison, "Efficient automatic integration of ordinary differential equations with discontinuities," *Mathematics and Computers in Simulation*, vol. 23, no. 1, pp. 12–20, 1981.

[20] I. A. Hiskens, "Dynamics of type-3 wind turbine generator models," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 465–474, 2011.

[21] ——, "Trajectory deadlock in power system models," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 2721–2724.

[22] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. Siam, 1998, vol. 61.

[23] C. W. Gear and O. Osterby, "Solving ordinary differential equations with discontinuities," *ACM Transactions on Mathematical Software (TOMS)*, vol. 10, no. 1, pp. 23–44, 1984.

[24] F. E. Cellier, "Combined continuous/discrete system simulation by use of digital computers: techniques and tools," 1979.

[25] C. C. Pantelides, "SpeedUp—recent advances in process simulation," *Computers & chemical engineering*, vol. 12, no. 7, pp. 745–755, 1988.

[26] F. Milano, "Semi-implicit formulation of differential-algebraic equations for transient stability analysis," *IEEE Transactions on Power Systems*, vol. 31, no. 6, pp. 4534–4543, 2016.

[27] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM, 1995.

[28] N. Guglielmi and E. Hairer, "Computing breaking points in implicit delay differential equations," *Advances in Computational Mathematics*, vol. 29, no. 3, pp. 229–247, 2008.

[29] J. M. Esposito, V. Kumar, and G. J. Pappas, "Accurate event detection for simulating hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2001, pp. 204–217.

[30] A. Ralston and P. Rabinowitz, *A first course in numerical analysis*. Courier Corporation, 2001.

[31] L. F. Shampine, I. Gladwell, and R. W. Brankin, "Reliable solution of special event location problems for ODEs," *ACM Transactions on Mathematical Software (TOMS)*, vol. 17, no. 1, pp. 11–25, 1991.

[32] C. Moler, "Are we there yet?" *Zero crossing and event handling for differential equations, Matlab News & Notes*, pp. 16–17, 1997.

[33] P. Rabinowitz, *Numerical methods for nonlinear algebraic equations*. Gordon & Breach Science Pub, 1970.

[34] L. F. Shampine, "Solving ordinary differential equations for simulation," *Mathematics and Computers in Simulation*, vol. 20, no. 3, pp. 204–207, 1978.

[35] J. L. Hay, R. E. Crosbie, and R. I. Chaplin, "Integration routines for systems with discontinuities," *The Computer Journal*, vol. 17, no. 3, pp. 275–278, 1974.

[36] F. E. Cellier and D. F. Rufer, "Algorithm suited for the solution of initial value problems in engineering applications," in *Proceedings of the international symposium and course SIMULATION*, 1975, pp. 160–165.

[37] R. Serban, C. Petra, A. C. Hindmarsh, C. J. Balos, D. J. Gardner, D. R. Reynolds, and C. S. Woodward, "User Documentation for idas v4. 7.0 (sundials v5. 7.0)," 2020.

[38] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 363–396, 2005.

[39] L. G. Birta, T. I. Oren, and D. L. Kettenis, "A robust procedure for discontinuity handling in continuous system simulation," *Transactions of The Society for Computer Simulation International*, vol. 2, no. 3, pp. 189–205, 1985.

[40] D. Fabozzi, A. S. Chieh, P. Panciatici, and T. Van Cutsem, "On simplified handling of state events in time-domain simulation," *Proceedings of the 17th PSCC*, 2011.

[41] P. J. Mosterman, "An overview of hybrid simulation phenomena and their support by simulation packages," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 1999, pp. 165–177.

[42] P. J. Mosterman, F. Zhao, and G. Biswas, "Sliding mode model semantics and simulation for hybrid systems," in *International Hybrid Systems Workshop*. Springer, 1997, pp. 218–237.

[43] A. D. Ames, H. Zheng, R. D. Gregg, and S. Sastry, "Is there life after Zeno? Taking executions past the breaking (Zeno) point," in *2006 American control conference*. IEEE, 2006, pp. 6–pp.

[44] K. H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, "On the regularization of Zeno hybrid automata," *Systems & control letters*, vol. 38, no. 3, pp. 141–150, 1999.

[45] H. Zheng, "Simulating Zeno hybrid systems beyond their Zeno points," Tech. Rep., 2006.

[46] J. M. Esposito and V. Kumar, "A state event detection algorithm for numerically simulating hybrid systems with model singularities," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 17, no. 1, pp. 1–es, 2007.

[47] P. G. O'Regan, "Step size adjustment at discontinuities for fourth order Runge-Kutta methods," *The Computer Journal*, vol. 13, no. 4, pp. 401–404, 1970.

[48] A. F. Filippov, *Differential equations with discontinuous righthand sides: control systems*. Springer Science & Business Media, 2013, vol. 18.

[49] L. Dieci, C. Elia, and L. Lopez, "On Filippov solutions of discontinuous DAEs of index 1," *Communications in Nonlinear Science and Numerical Simulation*, vol. 95, p. 105656, 2021.

[50] J. Na, H. Kim, H. Zhao, A. Gole, and K. Hur, "An improved high-accuracy interpolation method for switching devices in emt simulation programs," *Electric Power Systems Research*, vol. 223, p. 109630, 2023.

[51] M. Faruque, V. Dinavahi, and W. Xu, "Algorithms for the accounting of multiple switching events in digital simulation of power-electronic systems," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 1157–1167, 2005.

[52] P. Aristidou, S. Lebeau, and T. V. Cutsem, "Power system dynamic simulations using a parallel two-level schur-complement decomposition," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3984–3995, Sept 2016. [Online]. Available: http://orbi.ulg.ac.be/handle/2268/189192

[53] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.

[54] S. Glumac and Z. Kovačić, "Defect Analysis of a Non-Iterative Co-Simulation," *Mathematics*, vol. 11, no. 6, p. 1342, 2023.

[55] H. M. Bucker and G. F. Corliss, "A bibliography of automatic differentiation," *LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING*, vol. 50, p. 321, 2006.

[56] D. Fabozzi, A. S. Chieh, B. Haut, and T. Van Cutsem, "Accelerated and localized newton schemes for faster dynamic simulation of large power systems," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4936–4947, 2013.

[57] J. S. Chai, N. Zhu, A. Bose, and D. J. Tylavsky, "Parallel Newton type methods for power system stability analysis using local and shared memory multiprocessors," *IEEE Transactions on Power Systems*, vol. 6, no. 4, pp. 1539–1545, 1991.

[58] S. Karagiannopoulos, G. Valverde, P. Aristidou, and G. Hug, "Clustering Data-Driven Local Control Schemes in Active Distribution Grids," *IEEE Systems Journal*, vol. 15, no. 1, pp. 1467–1476, 2020.

[59] T. M. Inc., *MATLAB version: 9.10.0 (R2021a)*. Natick, Massachusetts, United States: The MathWorks Inc., 2021. [Online]. Available: https://www.mathworks.com

[60] M. Datta and T. Senjyu, "Fuzzy control of distributed PV inverters/energy storage systems/electric vehicles for frequency regulation in a large power system," *IEEE Transactions on Smart Grid*, vol. 4, no. 1, pp. 479–488, 2013.

[61] P. Kundur, N. J. Balu, and M. G. Lauby, *Power system stability and control*. McGraw-hill New York, 1994, vol. 7.

[62] "IEEE Recommended Practice for Excitation System Models for Power System Stability Studies," pp. 1–207, 2016.

[63] A. G. NEPLAN, "TURBINE-GOVERNOR MODELS: Standard Dynamic Turbine-Governor Systems in NEPLAN Power System Analysis Tool," Tech. Rep., 2015.

[64] H. F. Latorre, M. Ghandhari, and L. Soder, "Control of a VSC-HVDC operating in parallel with AC transmission lines," in *2006 IEEE/PES Transmission & Distribution Conference and Exposition: Latin America*. IEEE, 2006, pp. 1–5.

[65] D. Fabozzi and T. Van Cutsem, "Simplified time-domain simulation of detailed long-term dynamic models," in *2009 IEEE Power & Energy Society General Meeting*. IEEE, 2009, pp. 1–8.

[66] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[67] M. De Castro, D. Winkler, G. Laera, L. Vanfretti, S. A. Dorado-Rojas, T. Rabuzin, B. Mukherjee, and M. Navarro, "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]," *SoftwareX*, vol. 21, p. 101277, 2023.

APPENDICES

APPENDIX I

Reference equations and parameters

In this section, the equations and parameters used for the case studies of the thesis are attached.

I.1 Synchronous generator

The DAEs of the second-order synchronous machine, including the motion equations (fourth order in total), are formulated below [61].

- Differential equations:

$$\dot{\Delta\omega_r} = \frac{1}{2H}(T_m - T_e - K_D\Delta\omega_r) \tag{I.1}$$

$$\dot{\delta} = \omega_0\Delta\omega_r \tag{I.2}$$

$$\frac{1}{\omega_N}\dot{\psi_f} = v_f - R_f i_f \tag{I.3}$$

$$\frac{1}{\omega_N}\dot{\psi_{q1}} = -R_{q1} i_{q1} \tag{I.4}$$

- Algebraic equations:

$$0 = \psi_d - L_{dd} i_d - L_{df} i_f \tag{I.5}$$

$$0 = \psi_q - L_{qq} i_q - L_{qq1} i_{q1} \tag{I.6}$$

$$0 = \psi_f - L_{ff} i_f - L_{df} i_d \tag{I.7}$$

$$0 = \psi_{q1} - L_{qq1} i_q - L_{q1q1} i_{q1} \tag{I.8}$$

$$0 = v_d + R_a i_d + \psi_q \tag{I.9}$$

$$0 = v_q + R_a i_q - \psi_d \tag{I.10}$$

$$0 = v_d - \cos\theta_r^o v_x - \sin\theta_r^o v_y \tag{I.11}$$

$$0 = v_q - \sin\theta_r^o v_x + \cos\theta_r^o v_y \tag{I.12}$$

$$0 = i_d - \cos\theta_r^o i_x - \sin\theta_r^o i_y \tag{I.13}$$

$$0 = i_q - \sin\theta_r^o i_x + \cos\theta_r^o i_y \tag{I.14}$$

I.2 Controllers

The schematic of the AVR and the governor controlling the synchronous machine, and the POD controller is presented in Fig. I.1, Fig. I.2, and Fig. I.3, respectively. In addition, the values for the parameters of the exciter, the governor, and the POD controller are summarized in Table I.1, Table I.2, and Table I.3, respectively. Finally, the fuzzy AVR schematic is illustrated in Fig. I.4.

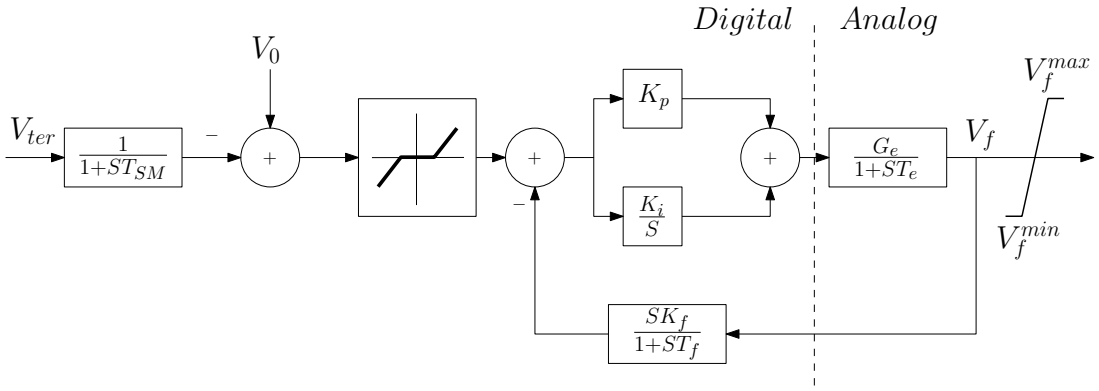

Figure I.1: Schematic of the AVR used to control the synchronous generator

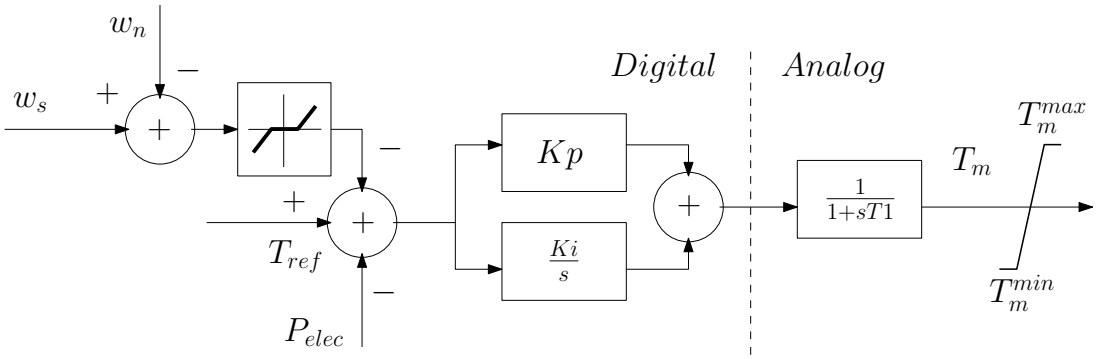

Figure I.2: Schematic of the governor used to control the synchronous generator

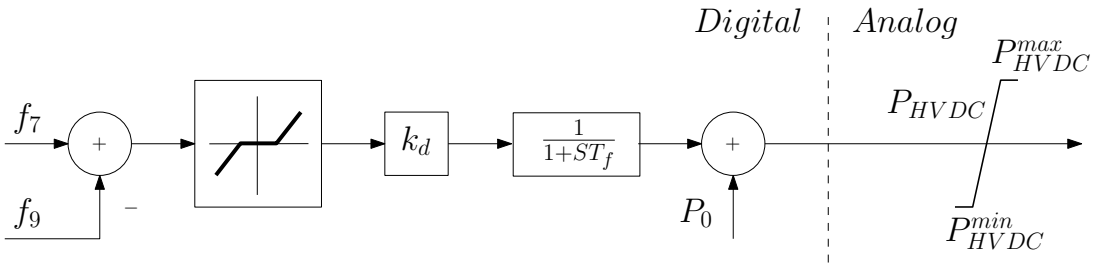

Figure I.3: Schematic of the POD controller used to control the HVDC line

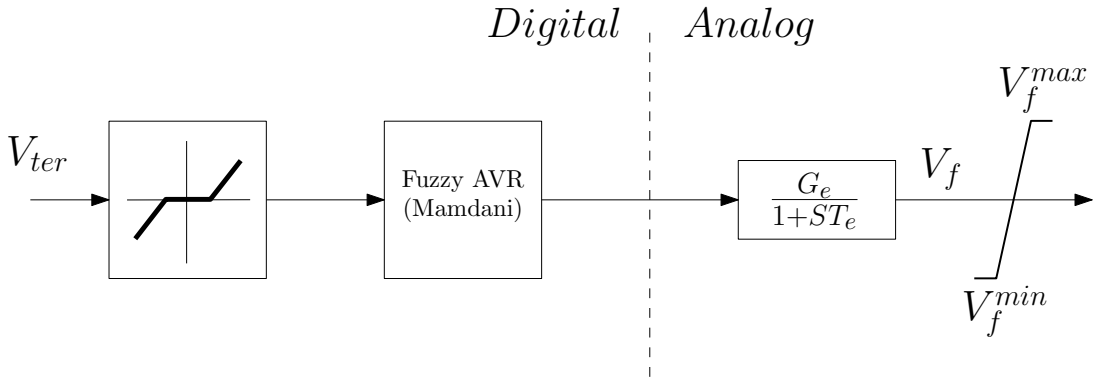

Figure I.4: Schematic of the fuzzy exciter used to control the synchronous generator

Table I.1: Parameter values for the AVR

Parameter	T_{SM}	K_p	K_i	G_e	T_e	K_f	T_f
Value	0.2	0.003	0.004	1	0.12	0.8	0.9

Table I.2: Parameter values for the governor

Parameter	K_p	K_i	$T1$
Value	12	0.2	0.3

Table I.3: Parameter values for the POD controller

Parameter	k_d	T_f
Value	8	0.1

APPENDIX II

Discrete controller handling GitHub repository

In this section a general guidance on how to use the GitHub repository of discrete controller handling [1] is provided.

On the main page of the repository, 6 folders can be found, each containing the code files of each published paper as shown in Fig. II.1.

Figure II.1: Main page of the discrete controller handling GitHub repository

Except for the `OSMSES 2024` folder, which has Modelica file simulations, MATLAB 2021 [59] is used for all the other simulation files.

Each case studies of papers are also organized in a separate folder. For example, 3 folders can be found for the `SEGAN 2025` paper as shown in Fig. II.2.

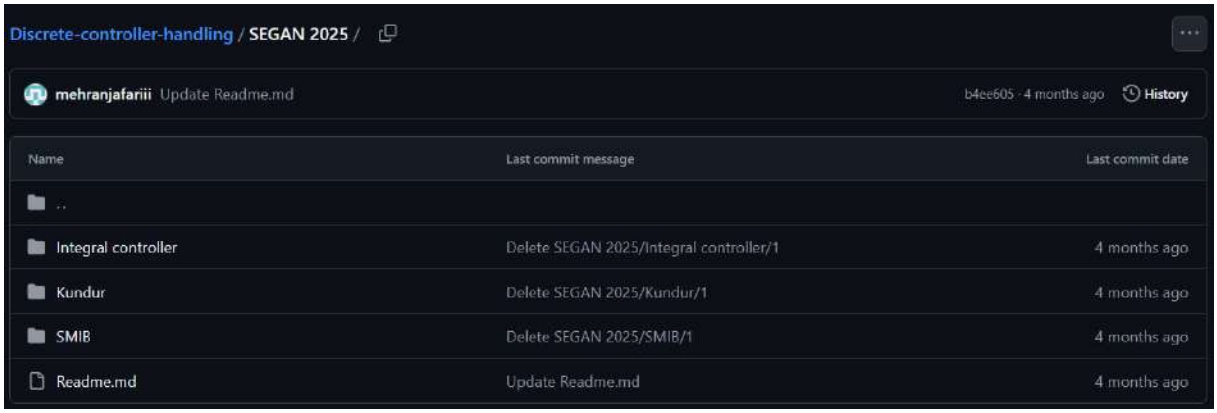

Figure II.2: Case studies of the SEGAN 2025 paper organized in three folders

Inside each case study folder, there are different files, each for a different method solving the same

[1] https://github.com/SPS-L/Discrete-controller-handling

test system. For example, Fig. II.3 shows the files for the integral controller case study of the `SEGAN 2025` paper.

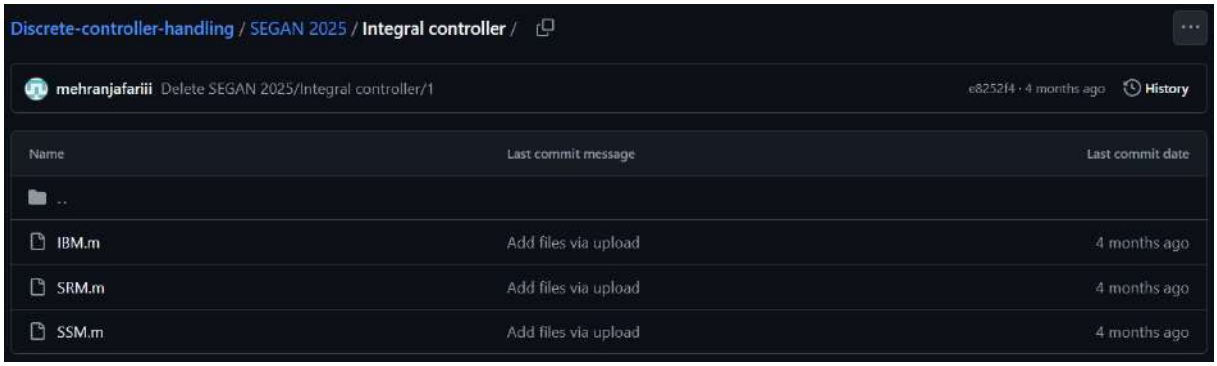

Figure II.3: Solving method's files for the integral controller test system of the SEGAN 2025 paper

If the case studies have equation-based controllers only, the user may simply run the code. However, if there is a fuzzy logic-based controller, first, the controller must be added to the MATLAB workspace. The fuzzy controllers can be found in the same case study folder as shown in Fig. II.4.

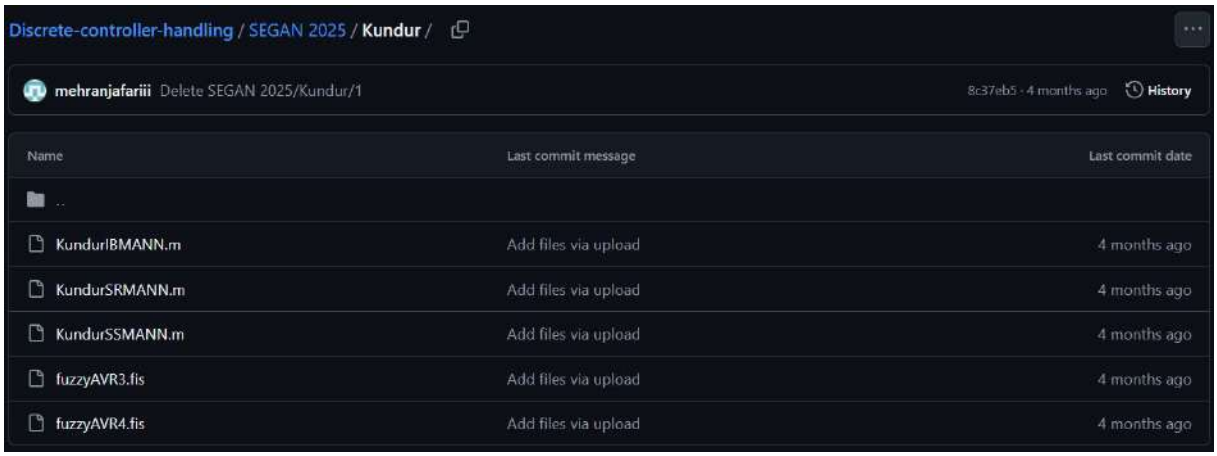

Figure II.4: Solving method's files and extra controller files of the Kundur test system of the SEGAN 2025 paper

The same logic stated above applies to other simulations, too. In addition, the user may refer to the `Readme` files of each published paper GitHub folder for the associated settings and more details.