**High-speed extended-term time-domain simulation for online cascading analysis of power system**

by

**Chuan Fu**

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Program of Study Committee:
James D. McCalley, Major Professor
Venkataramana Ajjarapu
Dionysios Aliprantis
Manimaran Govindarasu
L. Steve Hou

Iowa State University

Ames, Iowa

2011

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

A high-speed extended-term (HSET) time domain simulator (TDS), intended to become a part of an energy management system (EMS), has been newly developed for use in online extended-term dynamic cascading analysis of power systems. HSET-TDS includes the following attributes for providing situational awareness of high-consequence events: i) online analysis, including n-1 and n-k events, ii) ability to simulate both fast and slow dynamics for 1-3 hours in advance, iii) inclusion of rigorous protection-system modeling, iv) intelligence for corrective action ID, storage, and fast retrieval, and v) high-speed execution.

Very fast on-line computational capability is the most desired attribute of this simulator. Based on the process of solving algebraic differential equations describing the dynamics of power system, HSET-TDS seeks to develop computational efficiency at each of the following hierarchical levels, i) hardware, ii) strategies, iii) integration methods, iv) nonlinear solvers, and v) linear solver libraries.

This thesis first describes the Hammer-Hollingsworth 4 (HH4) implicit integration method. Like the trapezoidal rule, HH4 is symmetrically A-Stable but it possesses greater high-order precision ($h^4$) than the trapezoidal rule. Such precision enables larger integration steps and therefore improves simulation efficiency for variable step size implementations. This thesis provides the underlying theory on which we advocate use of HH4 over other numerical integration methods for power system time-domain simulation.

Second, motivated by the need to perform high speed extended-term time domain simulation (HSET-TDS) for on-line purposes, this thesis presents principles for designing numerical solvers of differential algebraic systems associated with power system time-

domain simulation, including DAE construction strategies (Direct Solution Method), integration methods(HH4), nonlinear solvers(Very Dishonest Newton), and linear solvers(SuperLU). We have implemented a design appropriate for HSET-TDS, and we compare it to various solvers, including the commercial grade PSS\E program, with respect to computational efficiency and accuracy, using as examples the New England 39 bus system, the expanded 8775 bus system, and PJM 13029 buses system.

Third, we have explored a stiffness-decoupling method, intended to be part of parallel design of time domain simulation software for super computers. The stiffness-decoupling method is able to combine the advantages of implicit methods (A-stability) and explicit method(less computation). With the new stiffness detection method proposed herein, the stiffness can be captured. The expanded 975 buses system is used to test simulation efficiency.

Finally, several parallel strategies for super computer deployment to simulate power system dynamics are proposed and compared. Design A partitions the task via scale with the stiffness decoupling method, waveform relaxation, and parallel linear solver. Design B partitions the task via the time axis using a highly precise integration method, the Kuntzmann-Butcher Method – order 8 (KB8). The strategy of partitioning events is designed to partition the whole simulation via the time axis through a simulated sequence of cascading events. For all strategies proposed, a strategy of partitioning cascading events is recommended, since the sub-tasks for each processor are totally independent, and therefore minimum communication time is needed.

# CHAPTER 1 INTRODUCTION

## 1.1 Introduction

A cascading event is recognized as chronological sequence of multiple lower-order dependent events [1]. An initial disturbance to a power transmission system, potentially capable of producing cascading failure, can propagate to ultimately produce a widespread blackout. Although the very severest cascading occurs only infrequently, when it does occur it may contribute significantly to economic and/or social catastrophe. A typical case of a large cascade is the blackout in the northern United States in August 2003 [2] It not only directly impacted residents and customers but also the regulatory environment and general perception of the entire industry. Cascading events need to be evaluated and managed in power system operation and planning, and constructive information regarding security of the power system, especially that related to high consequence events, must be supplied to assist operators in power-system control centers.

Extended-term time-domain simulation is significantly beneficial for analysis of power-system security, especially when cascading events can occur and persist for minutes or even hours. This is because a power system response to disturbances is decided not only by fast dynamics assisted by electronic technology, but also by the action of slow processes such as machine rotors and load dynamics. Simulation time ranging from minutes to hours can be helpful in investigating the effect of a series of power-system events and in determining a power system's ability to withstand large disturbances over extended periods of time. Due to the fact that such information is today typically not available in control centers with dynamic simulation tools, it is meaningful and even necessary to provide

operators with more information on extended-term simulation and to let control center personnel see the impact wrought by possible high consequence events.

There are two principal objectives for extended-term time-domain simulation of power systems: 1) online monitoring and tracking of high consequence events, and 2) providing preventive or corrective-action strategies for mitigating the impact of high-consequence events. For online application of extended-term simulation, one feasible idea is to simulate a system with well-selected contingencies (N-1 and N-k) as quickly as possible, permitting appropriate corrective or preventive actions to be taken to minimize damage to the system. [1,3,4] The simulation process associated with contingency selection and corrective operation can be illustrated in the three parts of Figure 1.1: 1) contingency selection, 2) time domain simulation and 3) intelligent and automatic operation part.

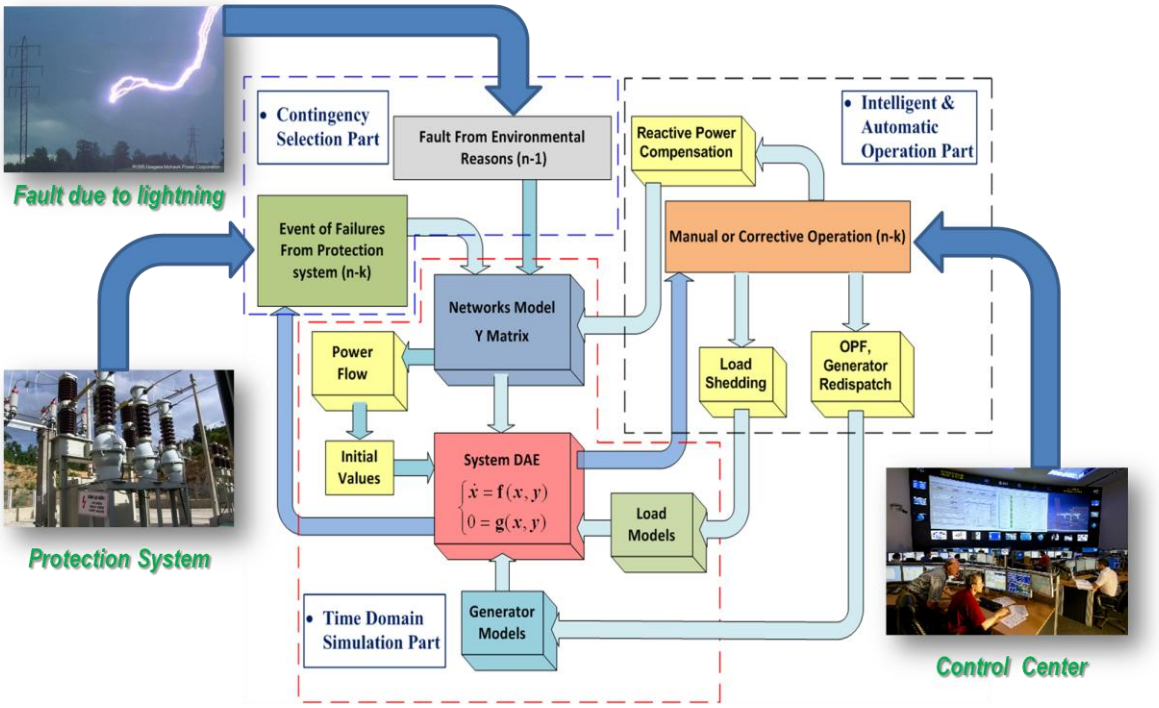



**Figure 1.1 The Relationship Between Contingency Selection, Time Domain Simulation and Corrective Operation**

- *Contingency selection*

For a power system in normal operation at a specific time, all state variables are in a balanced state, i.e, all state variables describing the system are not time-variable. System dynamics can be in unbalanced mode, perhaps produced by an event such as lightning-induced outage of transmission lines, and the corresponding Y impedance matrix of the system will be changed. The unbalanced mode can be reflected in a set of unbalanced dynamic equations describing the power system's dynamics, and solution of these equations can be stable or unstable. An event which produces mathematical equations describing an unbalanced power system is considered to be one contingency. The contingency selection part in Figure 1.1 supplies the (N-1) contingencies either from the network topology processing of the EMS (Energy Management System) or from possible contingencies obtained from historical statistical data. Cascading events are usually triggered by protection systems, and through analysis of such protection systems the potential high consequence event (N-k) can be supplied, as described in [1].

- *Time domain simulation*

When the impedance Y matrix is modified because of the potential events, each variable describing the system may be unbalanced and dynamic processes can occur on many dynamic elements such as generators, excitation, governors, and load. The essence of analyzing the dynamics of power systems is to solve a large set of differential algebraic equations (DAE). The initial values for the differential equations can be acquired via computation of power flow when the system is in a static state.

- *Intelligent and automatic operation*

After the occurrence of a high-consequence event, a possible positive result is that each variable in the DAE system describing the power system can achieve a new balance point at which each variable value lies within an acceptable bound. However, for most cases, the high consequence event may lead to instability or oscillation, and additionally some variables may stay out of acceptable limits. Three possible actions: load shedding, generation re-dispatch, and reactive power compensation, can be taken to prevent possibly damaging results. These actions can be directed by computations such as optimal power flow or by operator experience.

During this simulation process, high computational efficiency plays a decisive role. The faster the simulation, the more effective preventive action can be supplied. On the other hand, highly-efficient time-domain simulation is always welcomed for offline analysis of power-system security, especially when the power system is of large scale. Usually much longer time is required to simulate a large-scale system for even a very short period. High speed extended-term time-domain simulation for a large scale system is required for both industrial usage and academic research.

## 1.2 High Speed Extended Term (HSET) Time Domain Simulator (TDS)

A high-speed extended term (HSET) time domain simulator (TDS) represents a new functionality for control-center security assessment. This functionality is motivated by low-probability, high-consequence events to which power systems are continuously exposed. Such events, usually comprised of multi-element (so-called “N-k”) outages, often causing additional cascading events spanning minutes or even hours, are typically perceived to be

unlikely and therefore undeserving of preventive action and the associated costs typically involved in preventive action due to off-economic dispatch. Yet such events do occur and, in today's energy control centers, operational personnel have no available decision-support function to assist them in identifying effective corrective action, or even in becoming familiar with system performance under such events.

HSET-TDS, intended to be a part of an energy management system (EMS), has the following attributes [5]:

- *Probability-based contingency selection:* Contingencies are selected based on topological processing of node-breaker data based on user-specified probability order of magnitude. [1, 3, 4]
- *Extended-term:* Cascading sequences can play out over several hours, so HSET-TDS has capability to simulate for such a time frame.
- *Computational efficiency:* Time-domain simulation, involving the solution of numerical integration, is computationally intense. Therefore, it has been extremely challenging in today's control centers to implement associated functionality even for a short amount of simulated time, e.g., 10 seconds, for a limited number of contingencies. On-line simulation of minutes to hours for a very large number of contingencies requires computational efficiency several orders of magnitude greater than today's state-of-the-art.
- *Fast and slow-dynamics:* HSET-TDS must not only capture phenomena such as inertial instability affected by traditional fast dynamics (e.g., machine, excitation, speed-governing) but also phenomena such as voltage instability and cascading affected by slow dynamics (e.g., AGC, thermal changes in boilers, tap changing, and load variation).

- *Failure detection:* HSET-TDS must detect, within the simulation, unacceptable system performance such as out-of-step conditions, voltage deterioration, and thermal overload.
- *Corrective action identification:* Failure detection must be followed by, within the simulation, the identification of corrective actions such as redispatch, load shedding, network switching, or islanding.
- *Result storage:* HSET-TDS may be helpful in a responsive mode where it is run following initiation of a severe disturbance. Alternatively, we envision that it will play a more significant role in an anticipatory mode, continuously computing responses to many contingencies and storing preparatory corrective actions that would be accessed by operational personnel should one of the contingencies occur. The goal is to cover as much of the event-probability space as possible within a particulate computing time, e.g., 1 hour. Results can be archived and re-used when similar conditions are met in the future.

## 1.3 Motivation

In order to meet the requirement of on-line security assessment, we have been developing a new functionality that we refer to as a high-speed extended-term (HSET) time-domain simulator (TDS). HSET-TDS, intended to be a part of the energy management system (EMS), contains attributes, such as probability-based contingency selection, extended-term capability, computational efficiency, fast and slow dynamics, corrective-action identification, and result storage. An important attribute of HSET-TDS is very fast on-line computational capability to predict extended-term dynamic system response to disturbance. The work mainly focused on the acceleration of extended-term time-domain

simulation of power systems from the following five perspectives, 1) hardware, 2) DAE construction strategy, 3) integration methods, 4) nonlinear solvers, 5) linear solver libraries.

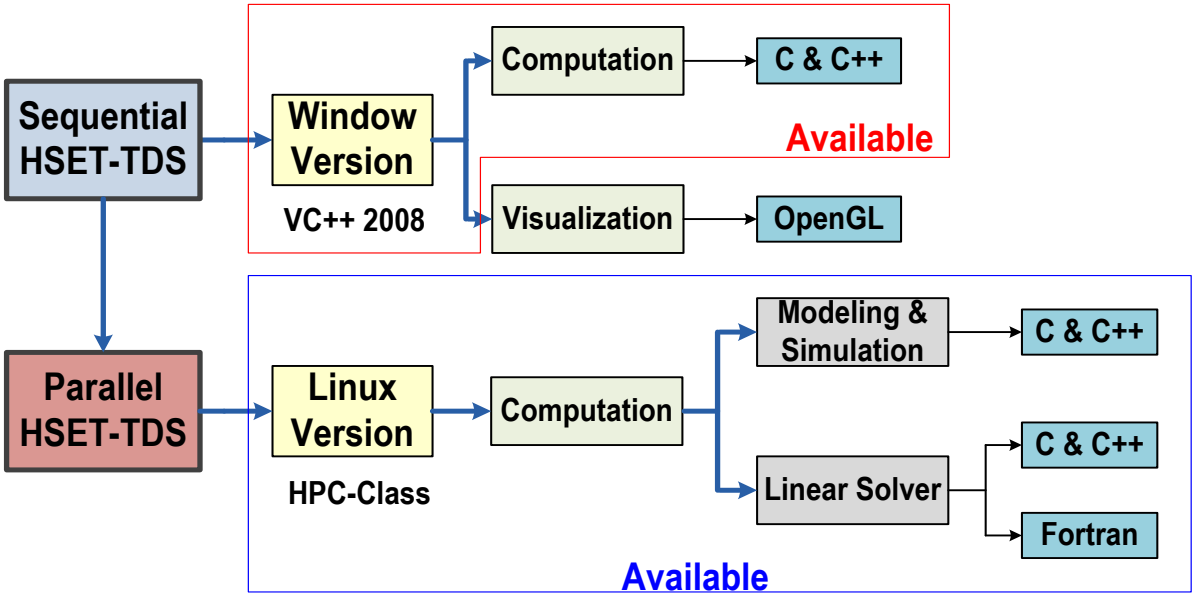



**Figure 1.2 Hardware Platforms of HSET-TDS**

So far as the hardware is concerned, HSET-TDS is being developed on two different platforms, sequential computing based on Windows and parallel computing based on Linux, as shown in Figure 1.2. The purpose of the sequential computing of HSET-TDS is to explore new numerical methods, nonlinear solvers, and linear solvers that are suitable for time-domain simulation of power system, and to compare the HSET-TDS with currently-available commercial software. Additionally, for small and medium-scale power systems, sequential computing is more meaningful than parallel computing. The parallel computing version of HSET-TDS is based on Linux operating systems, currently available on parallel-computing clusters of HPC-Class, and IBM Bluegene/L at Iowa State University. The goal of HSET-TDS parallel computing is to explore parallel computing algorithms suitable for extended-term time-domain simulation when a power system is large. Additionally, we wanted to

make use of a high-performance computer, IBM Bluegene/L, with many processors (1024), to experimentally determine whether a desirable simulation speedup can be acquired. The parallel computing version of HSET-TDS is also intended to meet the requirement of high online simulation cascading speed in large power systems.

HSET-TDS has been developed to include as many numerical methods as possible to determine the more suitable ones, and the structure of HSET-TDS related to these methods is shown in Figure 1.3. The motivation of exploring each category can be described as follows.

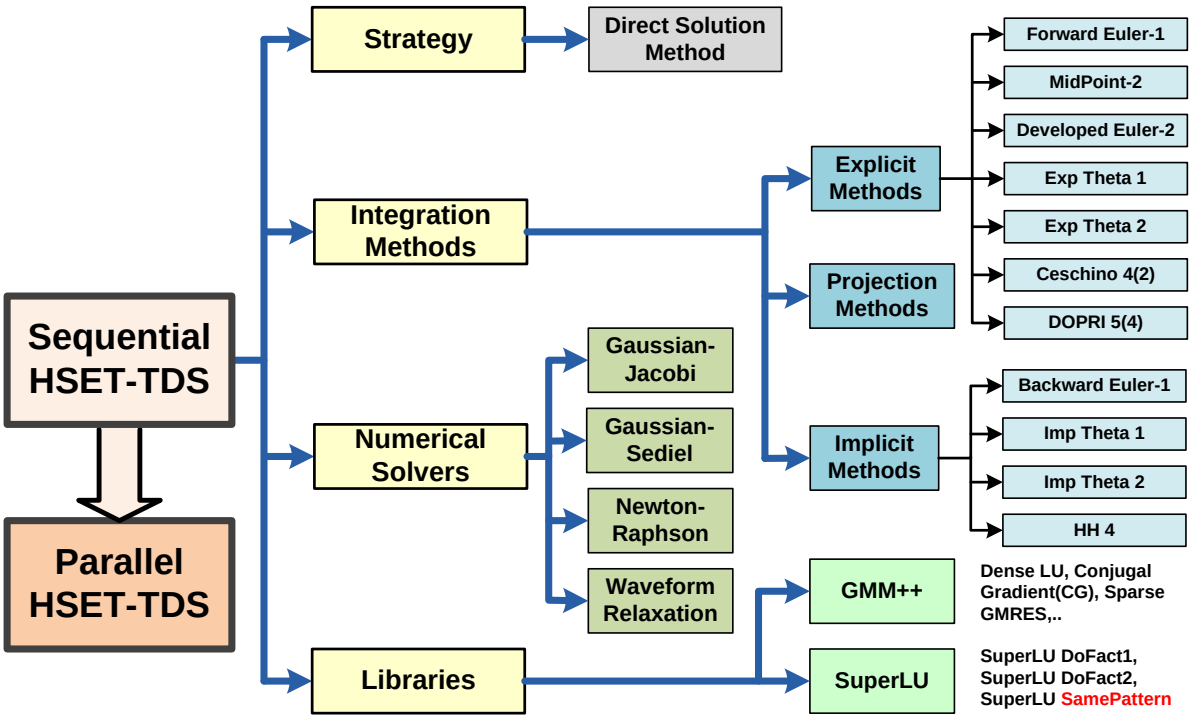



**Figure 1.3 Structure of Numerical Methods in HSET-TDS**

- *Strategies.*

The classification of strategies is introduced in Chapter 2, and it includes both the alternating-solution method and the direct-solution method. The motivation here is to find a suitable and efficient programming structure for HSET-TDS, and then to make the execution of integration methods, nonlinear solvers, and linear solvers more flexible.

- *Integration methods*

The integration methods in HSET-TDS include many types, including explicit methods, implicit methods, and projection methods, controlled by users and intended to be freely applied into the simulation. The motivation here is to explore new integration methods suitable for power-system analysis. A new integration method called Hammer-Hollingsworth 4 (introduced in Chapter 4), has been applied in HSET-TDS, and encouraging simulation results have been achieved.

- *Nonlinear solvers*

Nonlinear solvers are indispensible tools in the solution of differential algebraic equations. The motivation of this part is to explore a new parallel algorithm, one intended to separate the algebraic equations into independent parts and to thereby to easily realize the parallel computing algorithms.

- *Linear solver libraries*

The process of solving Ax=b is necessary in the solution of algebraic equations when Newton's method is used. Currently, there are many open-source sparse linear-solver libraries available in the world, including GMM++, SuperLU, UMFpack, and MUMPS. Some of these libraries are just for sequential computing, while some can be used for parallel computing. Additionally, the performances of these libraries are different because they encompass different methods and different programming approaches. Also, since the Jacobian matrix in power systems is highly sparse, the sparsity can impact the simulation performance tremendously. It is meaningful to include as many as linear solvers in HSET-TDS and to compare their performances.

Focusing on strategies, integration methods, and linear solver libraries for enhancing the efficiency of time-domain simulation, this thesis introduces a strategy of direct solution method, a new implicit integration method of Hammer-Hollingsworth 4(HH4), and serial and parallel SuperLU library of open-source linear solvers, all of which are used in HSET-TDS. For both small- and large-scale power system cases, all of the proposed numerical methods are compared with traditional methods adopted in many commercial software packages.

The thesis is organized as follows. Chapter 2 discusses the approaches necessary for constructing overall design of time-domain simulation. Chapter 3 verifies the newly-developed software HSET-TDS with reference to several dynamic models used in industrial software. Chapter 4 introduces integration methods including both several traditional integration methods as well as a newly-proposed implicit integration method, Hammer-Hollingsworth 4(HH4), and describes the stability properties of each. Chapter 5 introduces a stiffness-decoupling method based on stiffness-detection techniques. Chapter 6 introduces a linear solver of SuperLU used in HSET-TDS. Chapter 7 demonstrates the application of object-oriented programming in developing HSET-TDS. Chapter 8 introduces several parallel designs for extended time-domain simulation and comments on them. Finally, Chapter 9 concludes.

# CHAPTER 2 APPROACHES FOR HIGH SPEED EXECUTION FOR HSET-TDS

## 2.1 Problem Formulation

Time-domain simulation of power systems involves the solution of a large number of differential algebraic equations (DAEs)[6, 7], constructed based on the modeling of the power system electric elements and networks, including items such as generators, exciters, governors, and other electronic devices. The general form of DAEs can be described as follows.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{y}) & (1a) \\ 0 = \mathbf{g}(\mathbf{x}, \mathbf{y}) & (1b) \end{cases} \tag{2.1}$$

where vectors **x**, **y** are differential and algebraic variables respectively.

The ordinary differential equations (ODE) in (1*a*) describe the dynamics of machines and associated control systems. The algebraic equations in (1*b*) enforce Kirchhoff's Laws for the network and connected components. During the process of simulation the DAE system may change due to events such as faults (requiring modification of network topology during and after clearing the fault), or due to variation of injected power due to, for example, load and generator variation (including interruption or tripping).

## 2.2 Approaches Pyramid for Time Domain Simulation

An electric power system contains power networks and many different kinds of elements, such as generators, exciters, governors and other electronics devices, all of which can be described by a set of differential algebraic equations (DAEs)[6, 7]. Thus, the essence

of time-domain simulation is to solve a set of DAEs in the time domain. Many papers have been published on numerical methods used in commercial software for time-domain simulation of power systems, including PSS/E [8, 9], BPA [8], ETMSP [10], EXSTAB [11], and EUROSTAG [12, 13]. Design of DAE numerical solvers requires decisions to be made at several levels, including DAE construction strategy, integration method, nonlinear solver, and linear solver. We view these decisions hierarchically as illustrated in Figure 2.1 Hierarchical Pyramid for Time-Domain Simulation, with broader, structural design decisions represented at the top. There are five categories of methodologies during the process of solving a DAE system in power-system simulation: 1) hardware for execution-time domain simulation, 2) general methods for constructing DAE, 3) numerical-integration methods for discretization of differential equations, 4) numerical-iterative solvers for solving nonlinear algebraic equations, and 5) solver libraries for linear algebraic equations. In this paper, we define the following terminologies to describe each hierarchical category.

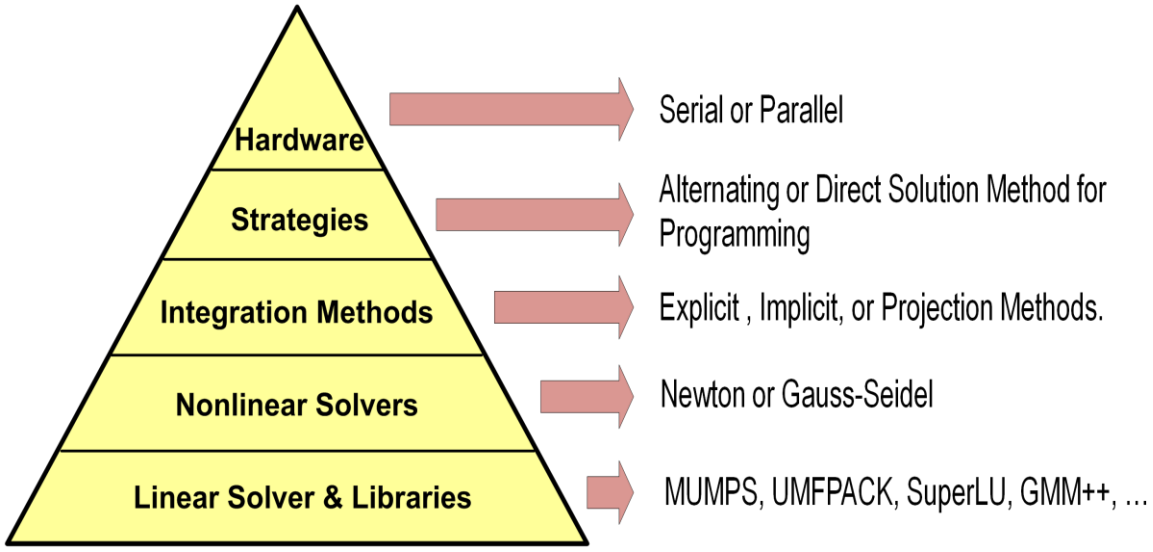



**Figure 2.1 Hierarchical Pyramid for Time-Domain Simulation**

**Hardware** – the hardware where the time-domain simulation is executed. It can be classified into sequential or parallel computing platforms .

**DAE construction strategy** – There are two basic strategies, depending on whether one solves the algebraic equations separately from or simultaneously with the differential equationsThe former approach is referred to as the alternating solution method (ASM), and the latter approach is referred to as the direct solution method (DSM). ASM is used most frequently in commercial grade applications [7], probably because it enables use of explicit integration methods, the integration method of choice for early designers.

**Integration method** –The integration method is used to discretize the ODE part of the DAE, resulting in algebraic equations that may or may not be solved with the network algebraic equations (1b) of (2.1) , depending on whether the integration method is implicit or explicit.

**Nonlinear solvers** –These are used to solve the nonlinear algebraic equations (2.1) of the problem. If an ASM strategy is used with explicit integration, then these equations will be simply those of (1b). If an ASM strategy is used with implicit integration, then there will two sets of nonlinear equations to solve – those of (1b) and those corresponding to the discretized ODEs. If a DSM strategy is used there is a single set of nonlinear equations to be solved simultaneously, comprised of (1b) combined with the discretized ODEs. The Gauss-Seidel method is used in many commercial applications software packages such as EXSTAB and EUROSTAG when a predictor-corrector scheme [11, 12] is adopted. Waveform Relaxation [14, 15, 16, 17], a convenient and effective approach for parallel computing, is another Gauss-Seidel-like nonlinear solver. A Newton method is frequently chosen for DSM strategies.

**Linear solvers** – Linear solvers are necessary to solve the linear algebraic equations $AX=B$ when a DSM strategy with a Newton nonlinear solver is used, and to solve the network equations if an ASM strategy is adopted. There are many available open-source linear-solver libraries such as GMM++ [18] , SuperLU [19] and UMFpack [20]. Some have reported on how to efficiently solve the linear equations to accelerate time-domain simulation by parallel computing, using the conjugate gradient method [21, 22] and block-bordered diagonal form [23] .

During the process of time-domain simulation, the selection of different choices in each hierarchical category of the pyramid will lead to different programming schemes, different computational efficiencies, and different computational precision. Also, a different choice in the top category has much influence on the bottom category, especially in computational quantity. The following sections will discuss each category respectively.

## 2.3 Hardware

Hardware is the basis for any calculation on computers. We classify the hardware into platforms for sequential computing and parallel computing according to the number of processors used in the computation of time-domain simulation,. For sequential computing, the HSET-TDS is generally developed on the PC with windows, while for parallel computing, the HSET-TDS is compiled and mounted on the Linux operating system adopted by high-performance computers in Iowa State University. In this section we mainly introduce the hardware for parallel computing.

**Table 2.1 HPC Local Recourses in ISU**

| HPC-class Cluster | BlueGene/L | Sun |
|---|---|---|
| Front-end 3.06 GHz dual Intel Xeon server with 2 GB of error-correcting memory. | Front-end node, service node, and storage nodes | |
| **16** compute nodes, 2.8 GHz dual Intel Xeons each with 2GB of ECC memory and 73GB of scratch disk. | **1024** compute nodes, dual-core PPC440 CPU, 700 Mhz, 512 MB RAM | 400 nodes, dual processor, AMD quad core, **3200** cores, 3.2 TB memory, |
| | 11 TB storage | 96 TB storage |
| | 5.7 TF peak compute power | 27.6TF peak compute power |
| Handles small-medium jobs | Handles large jobs | Handles large jobs |

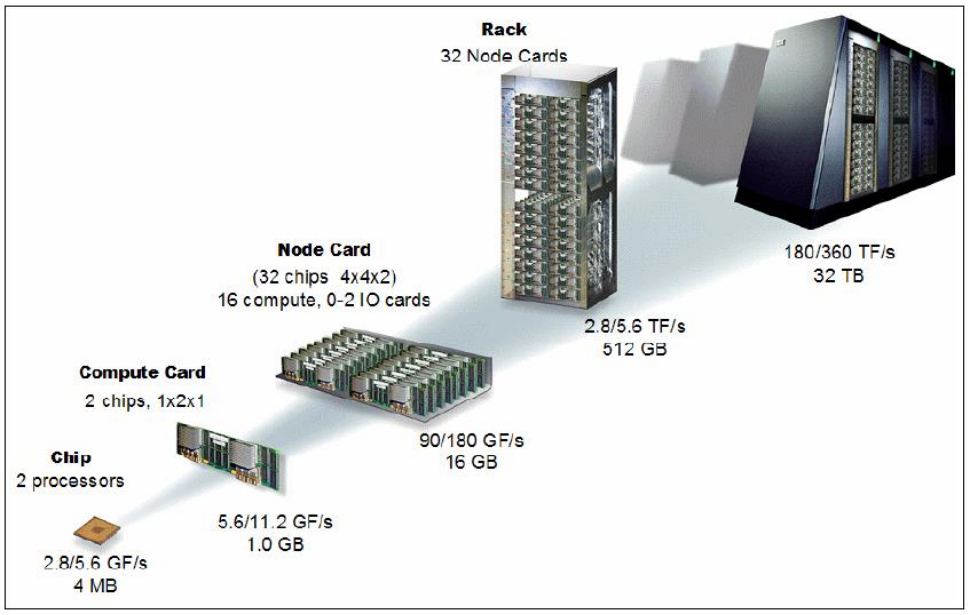



**Figure 2.2 BlueGene/L at a Glance**

Table 2.1 describes the high-performance computers available in Iowa State University. HSET-TDS has been mainly debugged on two platforms, an HPC-class cluster and BlueGene/L. The IBM Blue Gene/L represents today's state-of-the-art in computing. Compared to other computing solutions, it is physically small and very fast, has low power

consumption, and is relatively inexpensive. These attributes are achieved by providing massive parallelism via thousands of processing nodes connected together and organized into a grid, mesh, torus, or hypercube arrangement to allow each node to communicate with the other nodes. Iowa State University has recently purchased a Blue Gene/L consisting of 1024 chips, where each chip has two modified PowerPC® 440s running at 700 MHz. The structure of BlueGene/L is shown in Figure 2.2 (next page) The chips are connected by five networks with a latency of about 4 microseconds and a bandwidth of 350 Mb/sec. Each chip manages 64 compute nodes for a total of 65,536 compute nodes. The system stands within a rack with a power consumption of 28.14 kW. It has 512 GB of RAM and executes at 5.7 Tflops, in comparison to a 2.8 Ghz Pentium machine which typically has about 2 GB RAM and executes at 5.6 Gflops. A key feature of the Blue Gene/L is that its hardware is designed so that computational improvements are maximized when an algorithm is inherently parallelizable and implemented accordingly. We intend to achieve good parallelization results on Bluegene/L, and the preliminary experiments have been conducted on the HPC-Class machine.

## 2.4 Strategies

A power system includes networks and various electric elements, with the generator and the load being the main parts for dynamic analysis. Figure 2.3 illustrates, for a power-system DAE solution, the relationships between the main variables associated with generators, loads, and the network. Other devices that may also require modeling, particularly for extended-term simulation, are not shown in this high-level figure. The choice of DAE construction strategy, ASM or DSM, affects the relationship between solution of the

ODEs on the left-hand side of Figure 2.3 and the network and load equations on the right-hand side of Figure 2.3. We describe ASM in Section II-A and DSM in Section II-B together with its implementation in HSET-TDS.

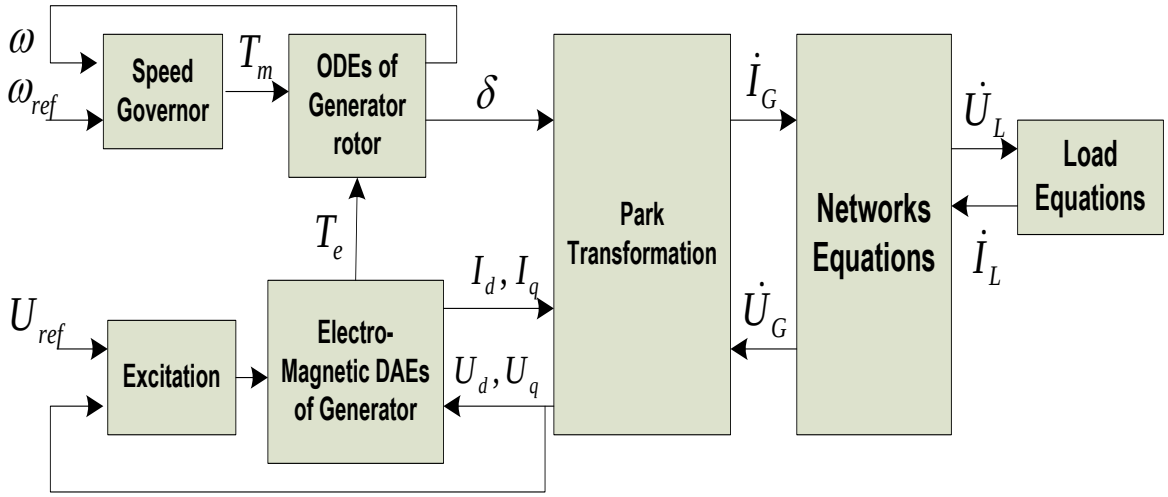



**Figure 2.3 DAE Construction of Power System**

### 2.4.1 Alternating Solution Method

References [6, 7] describe the alternating-solution method [6] (or partitioned-solution approach [7]) , the main idea of which is to alternatively solve for the injected currents of dynamic elements and the voltages of the nodes within the network by assuming the last step values of node voltages as the initial iterative values. ASM, therefore, divides the DAE system into two parts, the linear equations for the network and the sub-DAE system for the dynamic elements. The iterative process is illustrated in Figure 2.4, where it can be observed that there are two loops involved during the process of computing the next step value of the DAE system: the main iterative loop and the sub-iterative loop. The main loop iterates on network bus voltages using Gauss-Seidel, while the sub-loops iterate on the differential variables describing the dynamic devices. The advantage of this method is that the generators

can be represented by constant current sources, and admittances can be represented in the network admittance matrix. The resulting linear complex algebraic equations can be solved efficiently to obtain the bus voltages. However, since the Gauss-Seidel corrections of the main loop use sub-optimal directions, the main loop typically requires a larger number of iterations, with each iteration requiring a new set of sub-iterative loops (one for each generator). As a result, ASM-based simulation is computationally intensive compared to simulations where Newton's method is used.

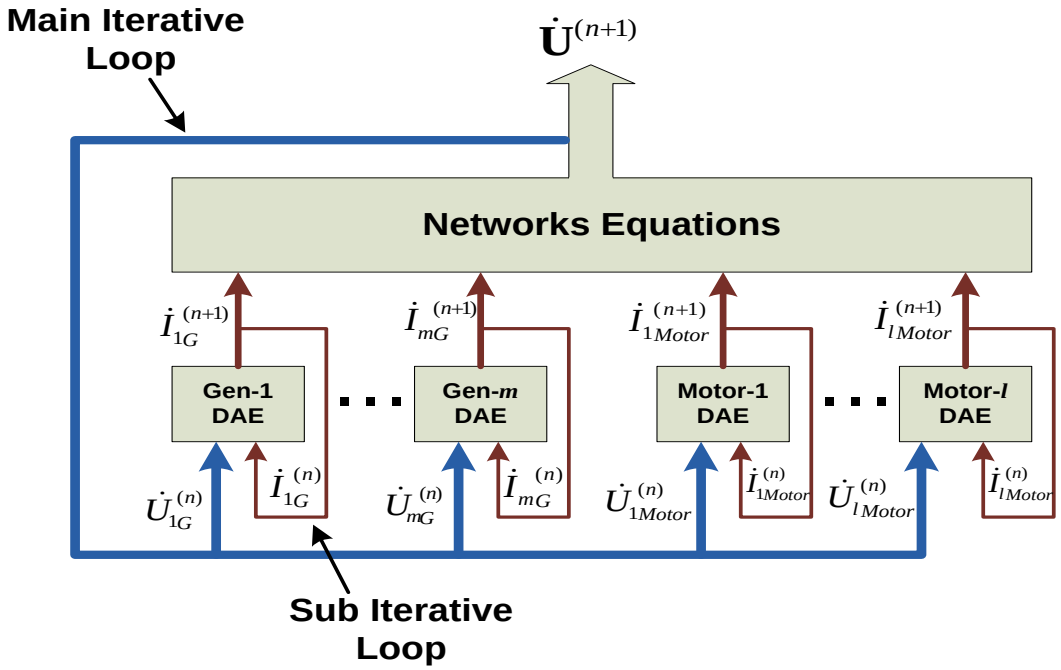



**Figure 2.4 The Iterative Process of the Alternating Solution Method**

### 2.4.2 Direct Solution Method

Unlike ASM, the DSM (or simultaneous-solution approach in [6]) combines the discretized ODEs and the algebraic equations describing the network and iterates as shown in Figure 2.5. One difficulty in realizing DSM is determining how to dynamically organize differential and algebraic variables, since different power systems have different topological

structures. The organizational structure shown in Figure 2.6 is adopted within HSET-TDS. Each variable corresponds to a particular set of differential equations or a particular set of algebraic equations. The functions of evaluating the equation and of taking the first derivative of the equation are programmed into software for the construction of the DAE system and corresponding Jacobian matrix. DSM has mainly two advantages compared with ASM. The first is that DSM allows gradient-based (Newton) methods such that each iteration's new solution is obtained along an optimal direction, and hence the number of iterations can be reduced. The second is that having code which constructs a single DAE system from the power system data makes experimentation with various numerical methods convenient. In contrast, ASM designs are constrained to use Gauss-Seidel iteration in the main loop. The sub-iterative loop may use any integration method, but most applications use the predictor-corrector method to achieve good efficiency [11, 12] .

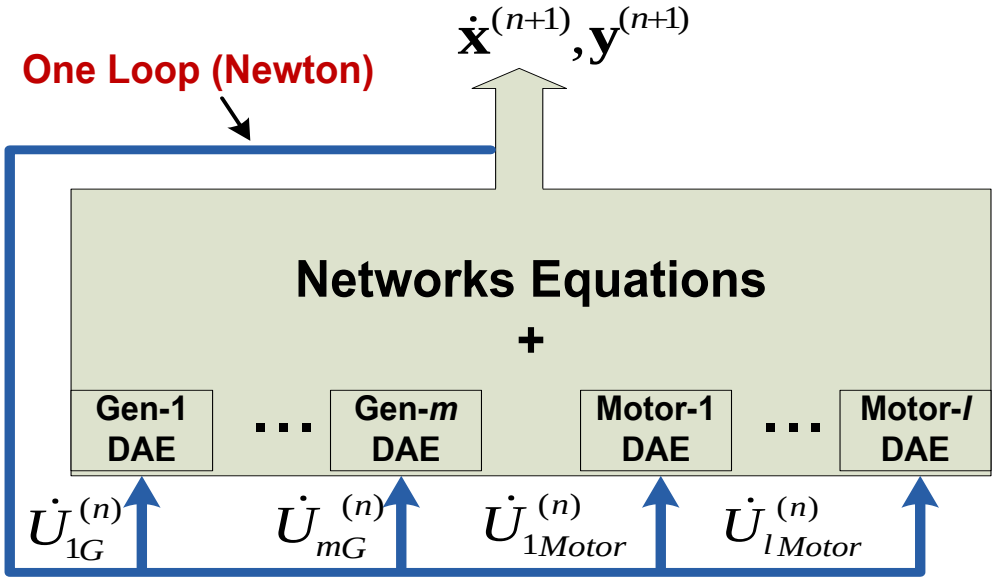



**Figure 2.5 The Iterative Process of Direct Solution Method**

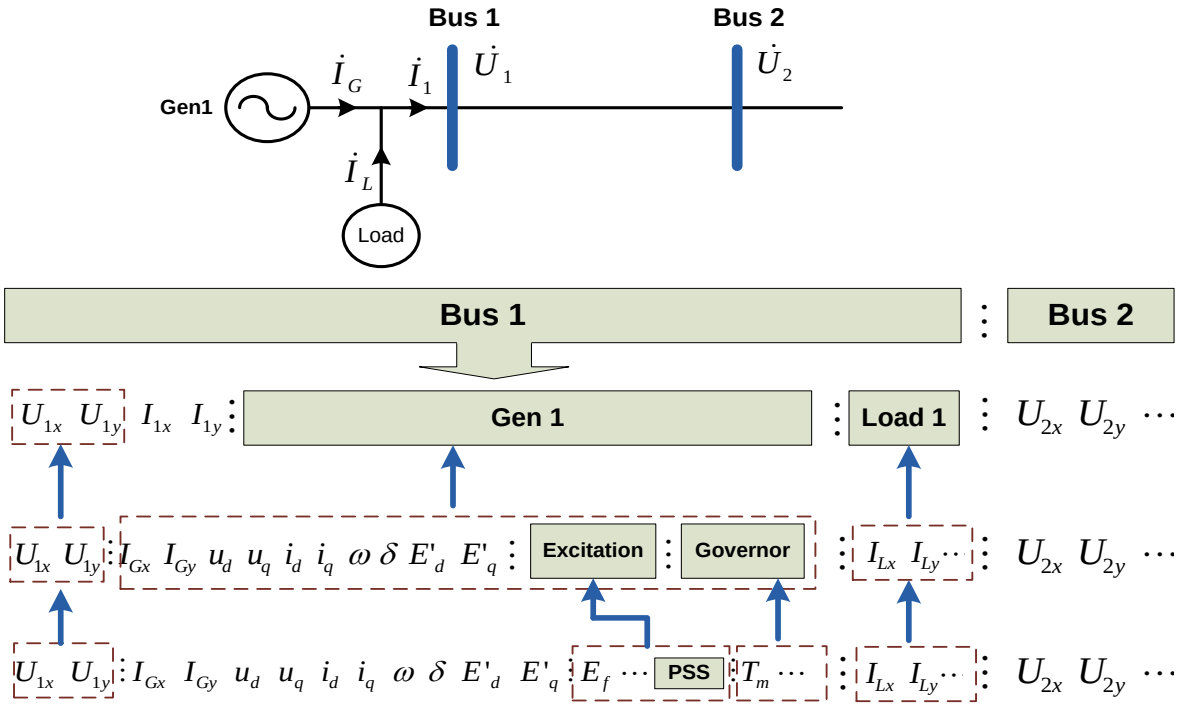



**Figure 2.6 Organization Structure of Differential and Algebraic Variables (The generator shown is of 4th order model)**

## 2.5 Integration Methods

The differential equation (1a) in the DAE system (1) requires use of numerical integration methods which discretize the differential equations. The discretization can lead to numerical error which can result in erroneous solution depending on the choice of integration method or the method's chosen parameters. There are two classes of integration methods: explicit (forward step) and implicit (backward step).

A suitable integration method for power system time-domain simulation must be capable of avoiding two problems, 1) stiffness [6, 24, 25, 26] and 2) hyper-stability [12] . Since there are many various dynamic elements with different time constants in electric power systems, the DAE system describing power systems can be stiff [6, 24, 25, 26].

According to [27], stiff equations are problems for which explicit methods do not work unless the integration step is taken very small. They are usually characterized by a wide range of time scales in terms of dynamic behavior, with eigenvalues large in magnitude present in the linearized system. Stiffness can lead to large error in simulation when using explicit integration methods if the integration step size is too large, a problem which can be solved by selecting a smaller integration step size at increased computational expense.

Reference [12] reported the phenomenon of hyper-stability of some numerical methods, which can cause an unstable system to simulate as a stable one. The hyper-stability problem is due to an attribute of some integration methods that results in overly-strong convergence when right-half-plane eigenvalues $\lambda$ of the linearized system, when discretized as $h\lambda$ ($h$ is time step), locate inside the stability domain of the integration method. The ability to deal with hyper-stability problem is part of our criteria in selecting a suitable integration method for power system time-domain simulation

There have been many publications about integration methods utilized in commercial software for time-domain simulation of power systems, including PSS/E, BPA , ETMSP, EXSTAB, and EUROSTAG. Table 2.2 summarizes the integration methods used by commercial software as reported in these publications. It is observed that the trapezoidal rule is used by many commercial time domain simulators. In addition, there has been significant investigation into other integration methods, such as multi-rated methods [28, 29] and the stiffness decoupled method[24, 25, 26]. In HSET-TDS, a new integrator named Hammer-Hollingsworth 4 is adopted, and the details is discussed in Chapter 4. Hammer-Hollingsworth 4 is not only A-stable sharing the same stability domain as Trapezoidal rule, but also is of the ability to compute the value of next point more precisely. The attribute of high precision

make it possible to enlarge the integration step, and therefore the whole integration times can be substantially decreased.

**Table 2.2 Integration Methods Adopted by Several Commercial Software**

| Commercial Software | Integration Method | Step Technique |
|---|---|---|
| **PSS/E** | Trapezoidal rule | Fixed step |
| **BPA** | Trapezoidal rule | Fixed step |
| **ETMSP** | Trapezoidal rule | Variable step |
| **EXTAB** | $\theta$ method ($\theta$=0.47) | Variable step |
| **EUROSTAG** | Mixed Adams-BDF ($2^{nd}$ order Adams) | Variable step |

## 2.6 Nonlinear Algebraic Equations Solver

The process of solving nonlinear algebraic equations, which is the third stage in Figure 2.1, is indispensible for computing the next step values with the nonlinear equations generated by the third stage of integration methods. Also, nonlinear algebraic equations solver supplies the linear algebraic equations to the fifth stage in Figure 2.1, and then these linear algebraic equations will be solved by linear equations libraries.There are two main iterative methods for solving nonlinear algebraic equations, i) Newton-Raphson method, and ii) Gaussian-Seidel method.

### 2.6.1 Gaussian-Seidel Method

The main idea of the Gauss-Seidel method is to solve the next iteration step value of one variable explicitly by using the same values of other variables in the last iteration step, or using a predicted value as an initial value to iterate and correct.

One application of Gauss-Seidel method in time domain simulation is the predictor-corrector scheme. The predictor-corrector scheme with Gauss-Seidel solution is adopted by many commercial time-domain simulation softwares because it is easy to program and because it conveniently accommodates the effects of non-linear elements (such as amplitude limiters). However, unlike the Newton-Raphson method, the Gauss-Seidel method is not gradient-based and therefore generally requires more iterations than the Newton-Raphson method.

### 2.6.2 Newton-Raphson Method

The Newton-Raphson (NR) method, an efficient method to solve nonlinear equations, has been widely used in many power system applications, especially power flow solutions. The main attribute of the NR method is that the solution process does not require many iterations compared with the Gauss-Seidel method because of the optimum choice in selecting the direction of variable change in each iteration. It is important to recognize that each NR iteration involves a solution of simultaneous linear algebraic equations, the part of the NR process that requires the most computational time. Figure 2.7 illustrates the relationship between the numerical integration methods, NR method and the linear solver libraries.

There are two problems associated with the efficiency of the NR method. First, the linear algebraic equations solved in each NR iteration are usually of large dimensions but are of highly sparsity requiring a sparse linear solver to avoid excessive computation time. Additionally, when the integration step becomes large, failure of the NR method may occur because of inappropriate starting points. Figure 2.8 shows three possible cases where NR can fail, and cases (b) and (c) may happen during the process of time domain simulation. Paper

[11] introduced a modified NR method where a deceleration factor is introduced. Use of a deceleration factor provides that failures due to cycling may be avoided; however, the cost is that the selected direction of variable change in each iteration is no longer optimal, and thus more iterations are needed.

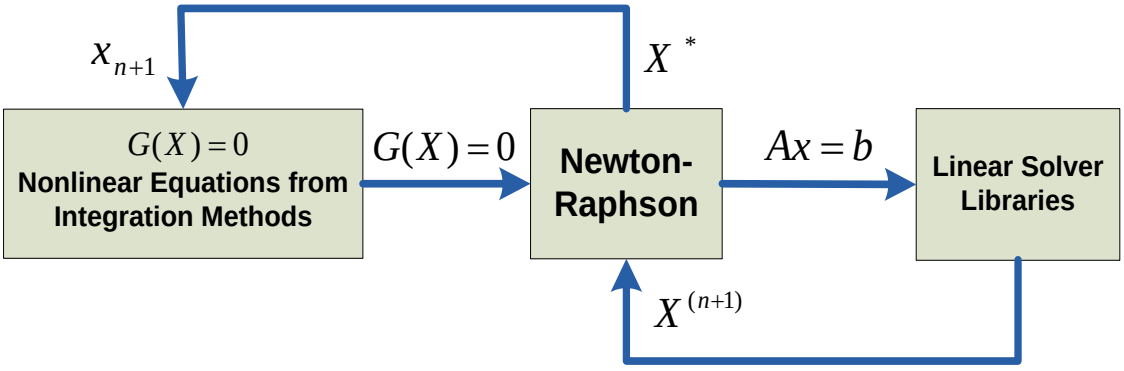



**Figure 2.7 Newton Method in Time-Domain Simulation**

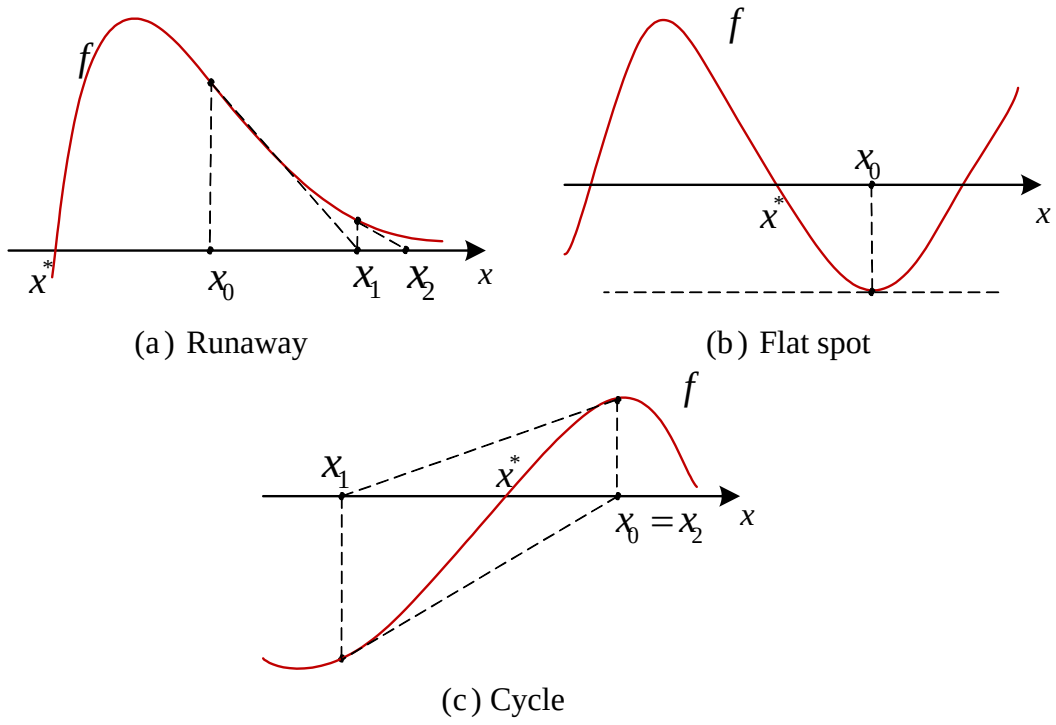



**Figure 2.8 Failure of Newton method**

A strategy used in HSET-TDS to address NR failures is step control, and of the number of NR iterations is the criterion used in determining whether to adjust the integration step. For the nonlinear algebraic equations $G(X)=0$, the modified NR method is described as follows.

$$\begin{cases} X^{(n+1)} = X^{(n)} - \alpha \cdot \Delta X \\ [J]\Delta X = [b] \end{cases} \quad \textbf{(2.1)}$$

where $[J]=\left[\frac{\partial G}{\partial X}\right]\Big|_{X=X^{(n)}}$ denotes Jacobian matrix; $\alpha$ is the deceleration factor; and $[b]=[G(X^{(n)})]$. Initially the deceleration factor is 1.0, and it is decreased by a small value when the max norm of $\Delta X$ exceeds a threshold. If the number of iterations exceed a set value, the NR method is considered failed, and then the integration step is decreased to obtain an improved starting point. The NR iterations will continue until $\Delta X$ falls below a tolerance. Another strategy in HSET-TDS to enhance efficiency of the NR method, which is called the Very DisHonest Newton (VDHN) method [12, 17, 30], is to keep the Jacobian matrix constant over many integration steps. The Jacobian matrix is not updated until the NR iterations fail. The advantage of VDHN is that the time dedicated to solving linear equations can be decreased since Jacobian factorization necessary in solving the linear equations is avoided for some iterations. In HSET-TDS, VDHN is implemented in coordination with SuperLU, a sparse linear solver discussed in the next section.

## 2.7 Linear Solver Libraries

Gaussian Elimination is the basic method to solve a set of linear equations, and there are many available open source libraries, such as GMM++[18] , SuperLU[19] , for solving

linear algebraic equations. Since much computation is cost during the process of solving linear equations in time-domain simulation, many papers have been report on how to efficiently solve the linear equations to accelerate time-domain simulation by parallel computing, such Conjugate Gradient Method[21, 22] , or how to partition the system via linear equations, such as Block Bordered Diagonal Form(BBDF)[23]. The Chapter 6 will discuss the open source library of SuperLU, and corresponding comparison with other linear solver in GMM++ will be elaborated.

# CHAPTER 3 GENERATOR MODELS AND VALIDATION OF HSET-TDS

HSET-TDS is designed for fast dynamic simulation of power systems, which involves the numerical computational solution of a large set of differential equations shown in (2.1). With the given DAE system, the following chapters will introduce a method for utilizing various numerical techniques to accelerate the solution of the DAE system describing power system dynamics. Before actually addressing the simulation results, it is important to determinewhether the basic DAE system is acceptable, and whether the simulation results can be matched with results produced by commercial software such as PSS/E when applied to the same system. The HSET-TDS has been developed to include the following ten different generator models[31, 32, 33] in order to satisfy the requirements of many different practical systems,.

1) 2nd order simplified Classic model, (just $X_d'$)
2) 2nd order Classic model ($X_d'$ and $X_q'$)
3) 3rd order model ($E_q'$)
4) 4th order model ($E_q'$and $E_d'$)
5) 6th order model (($E_q''$and $E_d''$)
6) 6th order model considering generator speed dynamics in stator equations.
7) 8th order model ( modeling from perspective of flux, no simplification, and considering stator dynamics)
8) GENROU model ($E_d'$, $E_q'$ , $\Psi_d'$, $\Psi_q'$)
9) GENROU model ($E_d'$, $E_q'$ , $\Psi_d'$, $\Psi_q'$, and considering speed dynamics in stator equations)

10) GENROU model ($E_d'$, $E_q'$, $\Psi_d'$, $\Psi_q'$, and considering stator dynamics)

In this chapter, however, only two of these ten generator models, the 6th order model and the GENROU model, will be introduced. Simulation results are presented and compared with those produced by PSS/E.

## 3.1 Generator 6th order model in HSET-TDS (GEN6)

The basic Park transformation generator equations can be expressed as follows.

$$\begin{bmatrix} \boldsymbol{u}_{dq0} \\ \boldsymbol{u}_{fDQ} \end{bmatrix} = \boldsymbol{p}\begin{bmatrix} \Psi_{dq0} \\ \Psi_{fDQ} \end{bmatrix} + \begin{bmatrix} \boldsymbol{S}_{dq0} \\ 0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{r}_{dq0} & \\ & \boldsymbol{r}_{fDQ} \end{bmatrix} \begin{bmatrix} -\boldsymbol{i}_{dq0} \\ \boldsymbol{i}_{fDQ} \end{bmatrix} \tag{3.1}$$

where $\boldsymbol{u}_{dq0} = \begin{bmatrix} u_d & u_q & u_0 \end{bmatrix}^T$ -- stator voltage after Park transformation

$\boldsymbol{u}_{fDQ} = \begin{bmatrix} u_f & u_D & u_Q \end{bmatrix}^T$ -- rotor voltage after Park transformation

$\boldsymbol{i}_{dq0} = \begin{bmatrix} \boldsymbol{i}_d & \boldsymbol{i}_q & \boldsymbol{i}_0 \end{bmatrix}^T$ -- stator current after Park transformation

$\boldsymbol{i}_{fDQ} = \begin{bmatrix} \boldsymbol{i}_f & \boldsymbol{i}_D & \boldsymbol{i}_Q \end{bmatrix}^T$ -- rotor current after Park transformation

$\Psi_{dq0} = \begin{bmatrix} \Psi_d & \Psi_q & \Psi_0 \end{bmatrix}^T$ -- stator flux

$\Psi_{fDQ} = \begin{bmatrix} \Psi_d & \Psi_q & \Psi_0 \end{bmatrix}^T$ -- rotor flux

$\boldsymbol{S}_{dq0} = \begin{bmatrix} -\omega\Psi_q & \omega\Psi_d & 0 \end{bmatrix}^T$

$\boldsymbol{r}_{dq0} = diag\begin{bmatrix} r_a & r_a & r_a \end{bmatrix}$

$\boldsymbol{r}_{fDQ} = diag\begin{bmatrix} r_f & r_D & r_Q \end{bmatrix}$

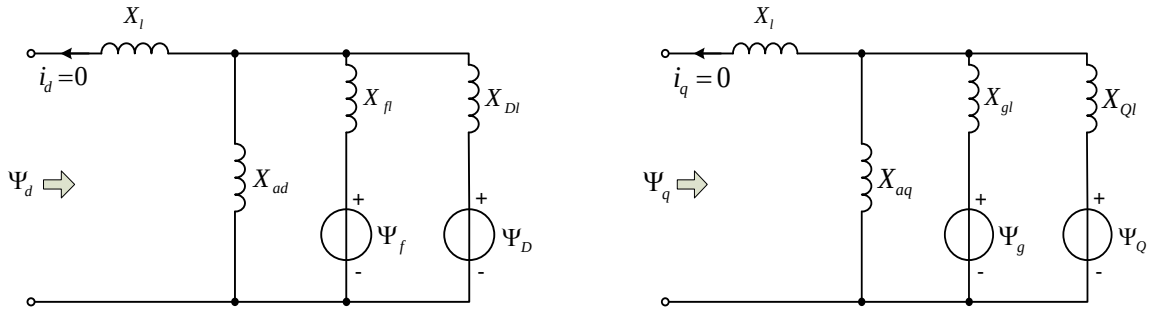
$$p = \frac{d}{\omega_B dt(s)}$$

The basic d-axis and q-axis flux equations can be described as follows.

$$\begin{cases} \Psi_d = -X_d i_d + X_{ad} i_f + X_{ad} i_D \\ \Psi_f = -X_{ad} i_d + X_f i_f + X_{ad} i_D \\ \Psi_d = -X_{ad} i_d + X_{ad} i_f + X_D i_D \end{cases} \tag{3.2}$$

$$\begin{cases} \Psi_q = -X_Q i_Q + X_{aq} i_g + X_{aq} i_Q \\ \Psi_g = -X_{aq} i_Q + X_g i_g + X_{aq} i_Q \\ \Psi_Q = -X_{aq} i_Q + X_{aq} i_g + X_Q i_Q \end{cases} \tag{3.3}$$

When a subtransient process is considered in generator equations, $E_q"$ and $E_d"$ can be acquired using the diagram shown in Figure 3.1.




(a)Expression of $\Psi_d$ for $E_q"$ (b) Expression of $\Psi_q$ for $E_d"$

**Figure 3.1 Modeling $\Psi_d$ and $\Psi_q$ for $E_q$" and $E_d$"**

We can define $E_q', E_d', E_q", E_d"$ according to [32] which can be shown as follows.

$$E_q' = \frac{X_f}{X_{ad}} \Psi_f \tag{3.4}$$

$$E_d' = -\frac{X_{aq}}{X_g} \Psi_g \tag{3.5}$$

$$E_q" = \frac{X_{ad}}{X_f X_D - X_{ad}^2} \left( X_{Dl} \Psi_f + X_{fl} \Psi_D \right) \tag{3.6}$$

$$E_d" = \frac{-X_{aq}}{X_g X_Q - X_{aq}^2} \left( X_{Ql} \Psi_g + X_{gl} \Psi_Q \right) \tag{3.7}$$

The time constant $T_{q0}', T_{d0}', T_{q0}'', T_{d0}''$ can be expressed as follows.

$$T_{q0}' = \frac{X_g}{r_g} \quad (3.8) \qquad T_{d0}' = \frac{X_f}{r_f} \quad (3.9)$$

$$T_{q0}'' = \frac{X_Q - \frac{X_{aq}^2}{X_g}}{r_Q} \quad (3.10) \qquad T_{d0}'' = \frac{X_D - \frac{X_{ad}^2}{X_f}}{r_D} \quad (3.11)$$

The transient and sub transient inductance can be described as follows.

$$X_q' = X_l + X_{aq} \quad (3.12) \qquad X_d' = X_l + X_{ad} \quad (3.13)$$

$$X_q' = X_l + X_{aq} // X_{gl} \quad (3.14) \qquad X_d' = X_l + X_{ad} // X_{fl} \quad (3.15)$$

$$X_q'' = X_l + X_{aq} // X_{Ql} \quad (3.16) \qquad X_d'' = X_l + X_{ad} // X_{fl} // X_{Dl} \quad (3.17)$$

From the d-axis and q-axis flux equations shown in (3.6) and (3.7) and the mechanical rotor equations, we can obtain the 6th order differential equations and 2 stator voltage equations to describe synchronous generator as follows:

$$\begin{cases} u_d = -\Psi_q - r_a i_d = E_d'' + X_q'' i_q - r_a i_d \\ u_q = \Psi_d - r_a i_q = E_q'' - X_d'' i_d - r_a i_q \end{cases} \quad (3.18)$$

$$T_{d0}' \, pE_q' = E_f - \frac{X_d - X_l}{X_d' - X_l} E_q' + \frac{X_d - X_d'}{X_d' - X_l} E_q'' - \frac{(X_d - X_d')(X_d'' - X_l)}{X_d' - X_l} i_d \quad (3.19)$$

$$T_{q0}' \, pE_d' = -\frac{X_q - X_l}{X_q' - X_l} E_d' + \frac{X_q - X_q'}{X_q' - X_l} E_d'' + \frac{(X_q - X_q')(X_q'' - X_l)}{X_q' - X_l} i_q \quad (3.20)$$

$$T_{d0}'' \, pE_q'' = \frac{X_d'' - X_l}{X_d' - X_l} T_{d0}'' \, pE_q' - E_q'' + E_q' - (X_d' - X_d'') i_d \quad (3.21)$$

$$T_{q0}'' \, pE_d'' = \frac{X_q'' - X_l}{X_q' - X_l} T_{q0}'' \, pE_d' - E_d'' + E_d' + (X_q' - X_q'') i_q \quad (3.22)$$

$$T_J \frac{d\omega}{dt} = T_m - \left[E_q"i_q + E_d"i_d - \left(X_d" - X_q"\right)i_d i_q\right] - D(\omega - 1) \tag{3.23}$$

$$\frac{d\delta}{\omega_B dt} = \omega - 1 \tag{3.24}$$

where $E_f$ and $T_m$ are the excitation voltage and the prime turbine mechanical torque . In HSET-TDS, the available model for excitation and governor are IEEE1 excitation models as in PSS/E (or EXC-1A as in ETMSP) and GOV1 model as in PSS/E (or GOV-8 as in ETMSP) respectively [10, 34]. The description of these four models are illustrated in Figure 3.2 through Figure 3.5. These excitation and governor models can be embedded within the set of all generator models in HSET-TDS with inputs $E_f$ and $T_m$. Also, these two models are just for research-grade application, so the dynamic data from practical power systems must be revised to match these two models. However, they are sufficient to serve as objects of comparison between HSET-TDS and commercial software such as PSS/E.

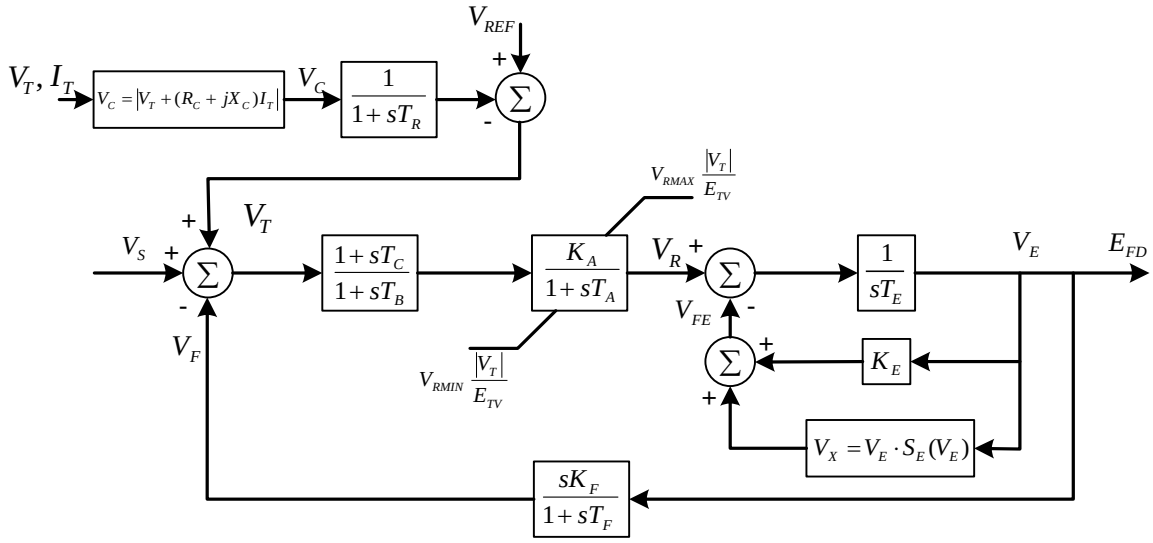



**Figure 3.2 EXC-1A in ETMSP**

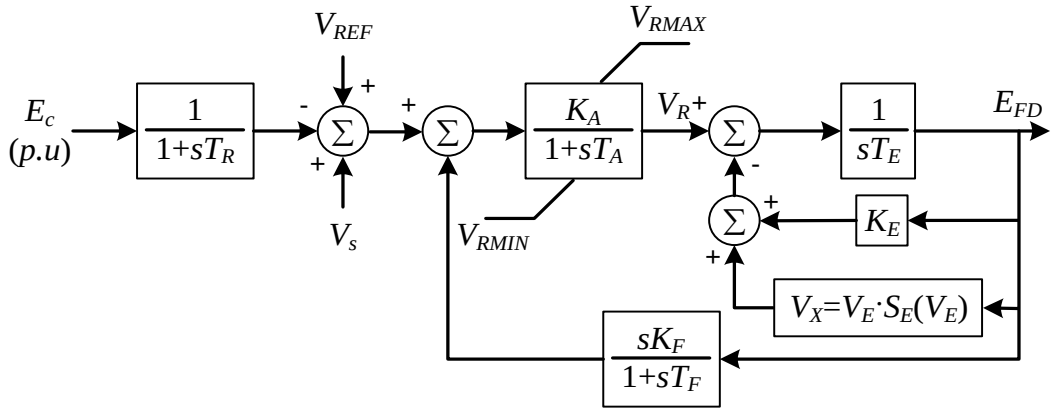



**Figure 3.3 IEEE1 Excitation Model in PSS/E**

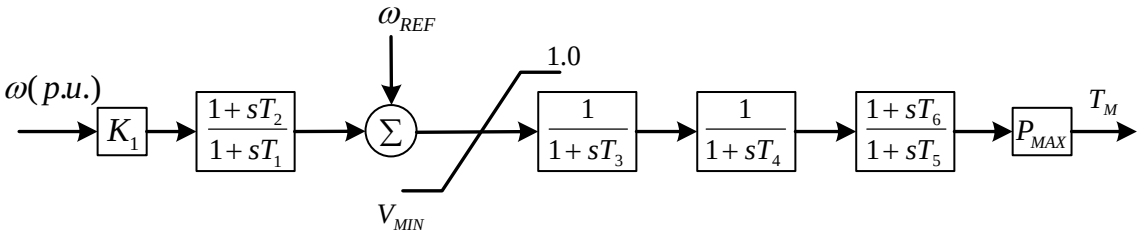



**Figure 3.4 GOV-8 in ETMSP**

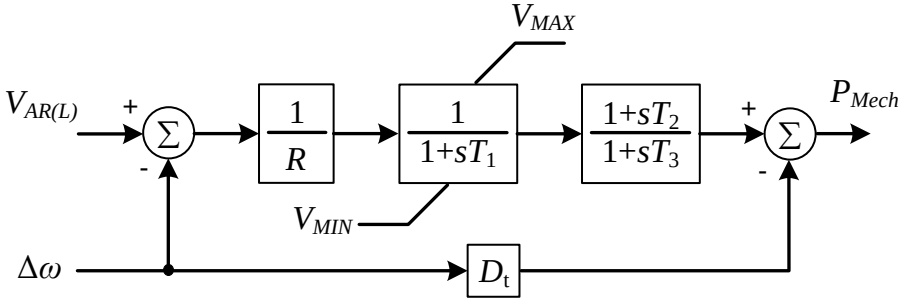



**Figure 3.5 TGOV1 model in PSS/E**

## 3.2 GENROU model in HSET-TDS

GENROU [34] is a standard generator model, widely used in practical dynamic analysis of power systems. According to May 2009 dynamics data from PJM, for a system

with approximately 10000 buses, more than 90% of the generators use GENROU model. The PSS/E manual supplies a diagram about showinghow to express $\Psi_d$" and $\Psi_q$" from the d-axis and the q-axis, shown as Figure 3.6 in [34].

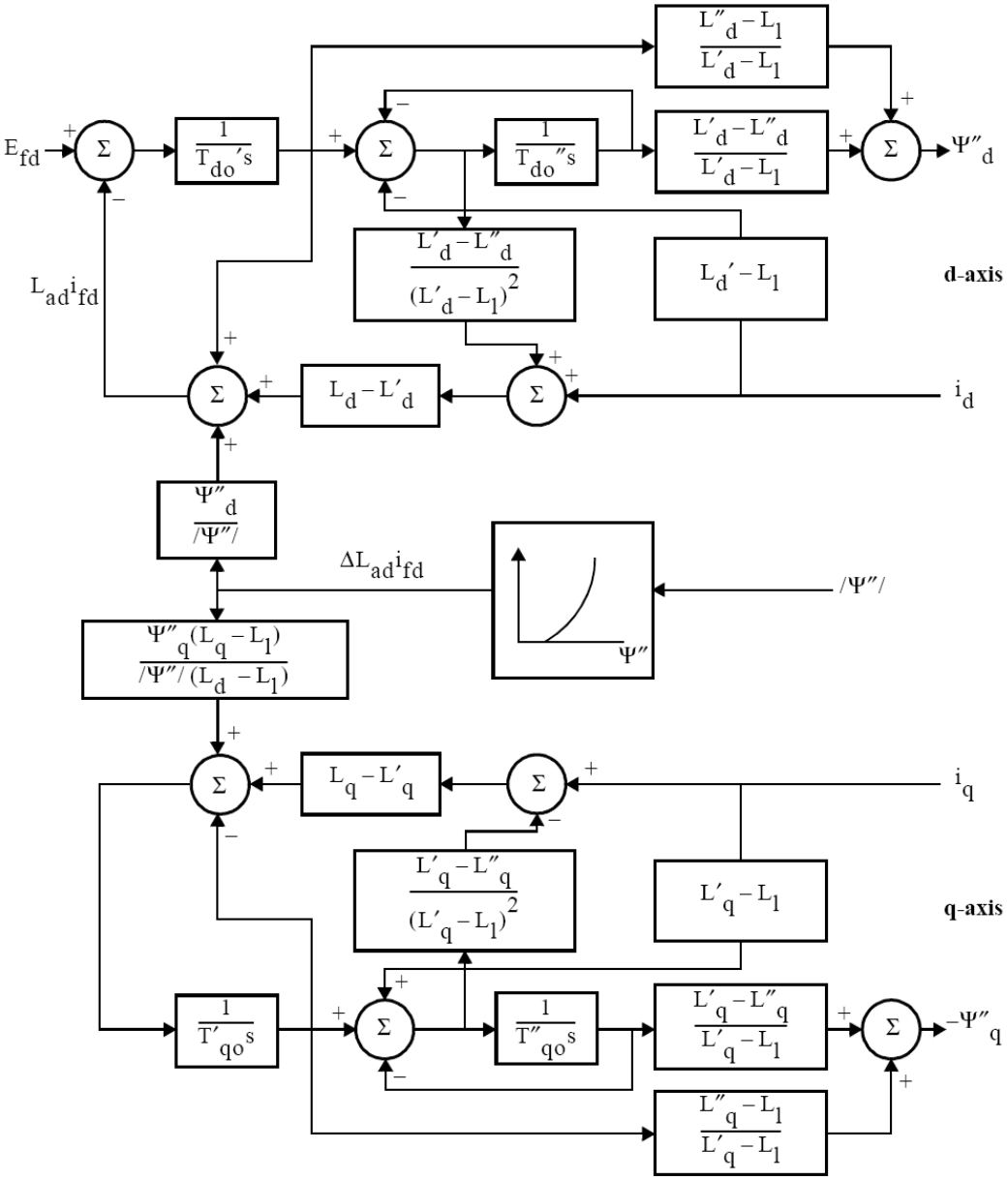



**Figure 3.6 Electromagnetic Model of Round Rotor Generator from PSS/E [34]**

[35] provides a similar GENROU diagram shown in Figure . It can be found that these two diagrams are basically identical with respect to the differential part but different with respect to saturation. In HSET-TDS, the GENROU model does not consider saturation currently, and the differential parts follow the diagram in Figure 3.7. The equations for $E_q'$, $E_d'$ , $E_q''$, $E_d''$ are described as follows.

$$T_{d0}'\frac{dE_q'}{dt} = E_f + \frac{(X_d - X_d')(X_d' - X_d'')}{(X_d' - X_l)^2}E_q'' - \frac{(X_d - X_d')(X_d'' - X_l)}{(X_d' - X_l)}i_d - \left[\frac{(X_d - X_d')(X_d - X_d')}{(X_d' - X_l)^2} + 1\right]E_q' \tag{3.24}$$

$$T_{q0}'\frac{dE_d'}{dt} = \frac{(X_q - X_q'')(X_q - X_q')}{(X_q' - X_l)^2}E_d'' - \frac{(X_d - X_d')(X_d'' - X_l)}{(X_q' - X_l)}i_q - \left[\frac{(X_q - X_q')(X_q - X_q')}{(X_q' - X_l)^2} + 1\right]E_d' \tag{3.25}$$

$$T_{d0}''\frac{dE_q''}{dt} = E_q' - (X_d' - X_l)i_d - E_q'' \tag{3.26}$$

$$T_{q0}''\frac{dE_d''}{dt} = E_d' + (X_q' - X_l)i_q - E_d'' \tag{3.27}$$

$$T_J\frac{d\omega}{dt} = T_m - \left(\Psi_d i_q - \Psi_q i_d\right) - D(\omega - 1) \tag{3.28}$$

$$\frac{d\delta}{\omega_B dt} = \omega - 1 \tag{3.29}$$

$$\Psi_d = \frac{X_d'' - X_l}{X_d' - X_l}E_q' + \frac{X_d' - X_d''}{X_d' - X_l}E_q'' - i_d X_d'' \tag{3.30}$$

$$\Psi_q = \frac{X_q'' - X_l}{X_q' - X_l}E_d' + \frac{X_q' - X_q''}{X_q' - X_l}E_d'' - i_q X_q'' \tag{3.31}$$

$$u_d = -r_a i_d - \Psi_q \tag{3.32}$$

$$u_q = -r_a i_q + \Psi_d \tag{3.33}$$

In a manner similar to that for the 6th order model, $E_f$ and $T_m$ in GENROU can be embedded within the IEEE1 or GOV1 model. Additionally, these two values can be set to

constant values depending on different situations. So far as the stator equations in the GENROU model are concerned, it can be shown that there are some differences when compared with stator expression (3.1). There are two assumptions about the stator equations:

i) Stator dynamics are not included, meaning that $\frac{d\Psi_d}{dt}$ and $\frac{d\Psi_q}{dt}$ are too small to be involved.

ii) During the disturbance process , generator rotor speed is close to its rated value, in which case $\omega \approx 1$ .

The first assumption is justified since $\left|\frac{d\Psi}{dt}\right|$ is much less than $|\omega\Psi|$ in the practical case, and therefore the stator equation become linear. The second assumption is justified since there are many control units for rendering rotor speed to be near rating values such that the system frequency can be guaranteed to be near its rated value. In HSET-TDS, there is an auxiliary GENROU model that includes the dynamics of stator or variable rotor speed in the stator equations. Comparison and analysis addressing this issue will be shown in a future report.

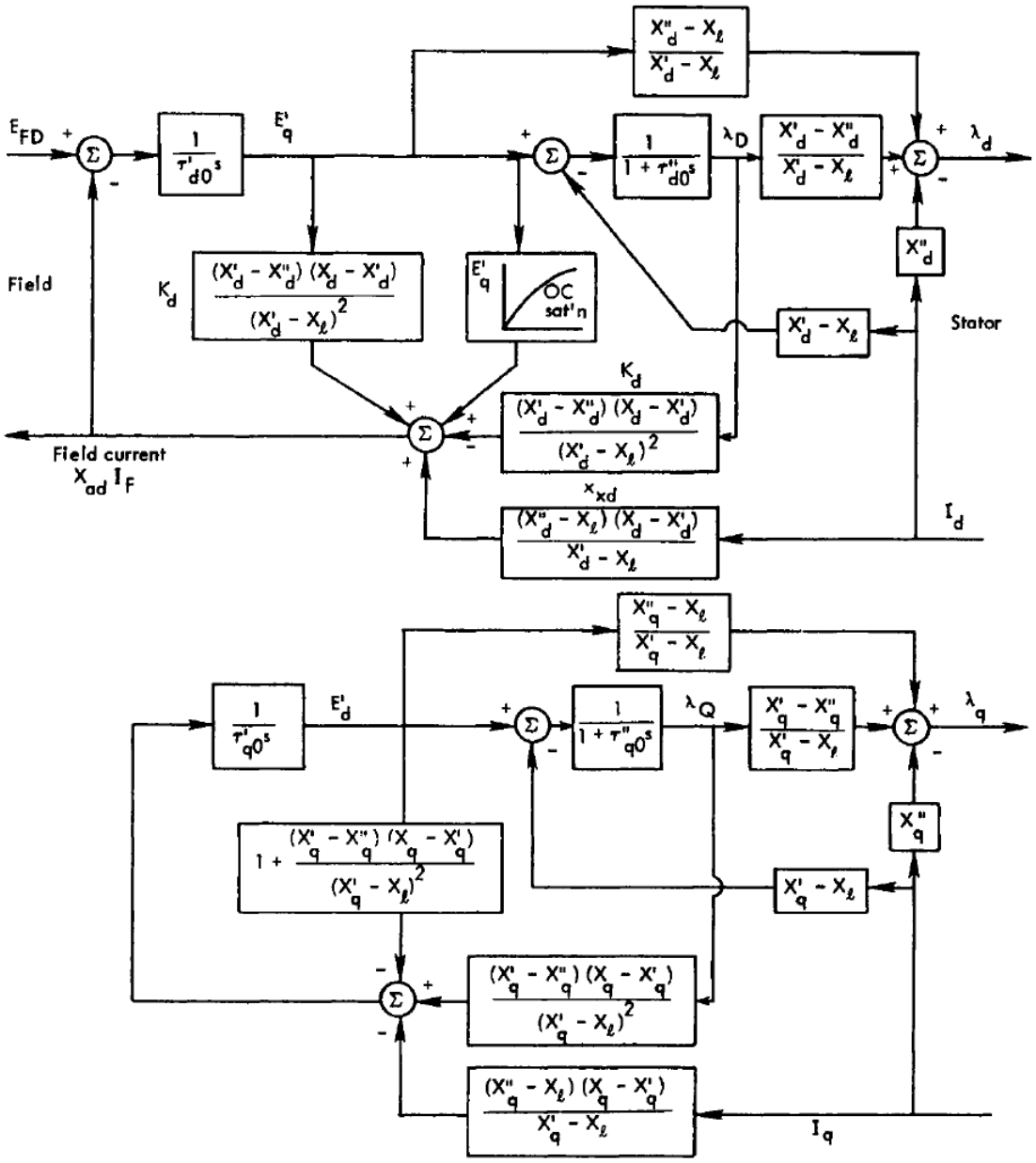



**Figure 3.7 Voltage behind Sub-transient Reactance Model [35]**

## 3.3 Validation of HSET-TDS by PSS/E

In this section three cases have chosen to allow comparison between HSET-TDS and PSS/E. The first such case is the New England 39-bus 10-generator system, and the second case is an expanded version of a 39-bus system. In HSET-TDS, initial values for the DAE

system are calculated by power flow, based on the Newton-Raphson method. The integration methods used is variable-step Trapezoidal rule, as will be discussed in next chapter. The nonlinear solver is based on the Newton method, and the SuperLU library is chosen as the linear solver.

### 3.3.1 Case 1: New England 39-bus, 10-gen system

IEEE New England 39 Bus 10 Gen system is a simplified system, as shown in Figure 3.8. In HSET-TDS, GEN6 and GENROU are chosen for simulation, while in PSS/E only the GENROU is adopted. $E_f$ and $T_m$ are assumed to be constant in each software package. The case event is selected to be a bus fault on bus 17 starting at 0.5s and lasting for 0.1s. The voltage at bus 37 and the speed of G8 will be monitored. The simulation lasts for 10 seconds. Figure 3.9 shows the simulation results from HSET-TDS and PSS/E.

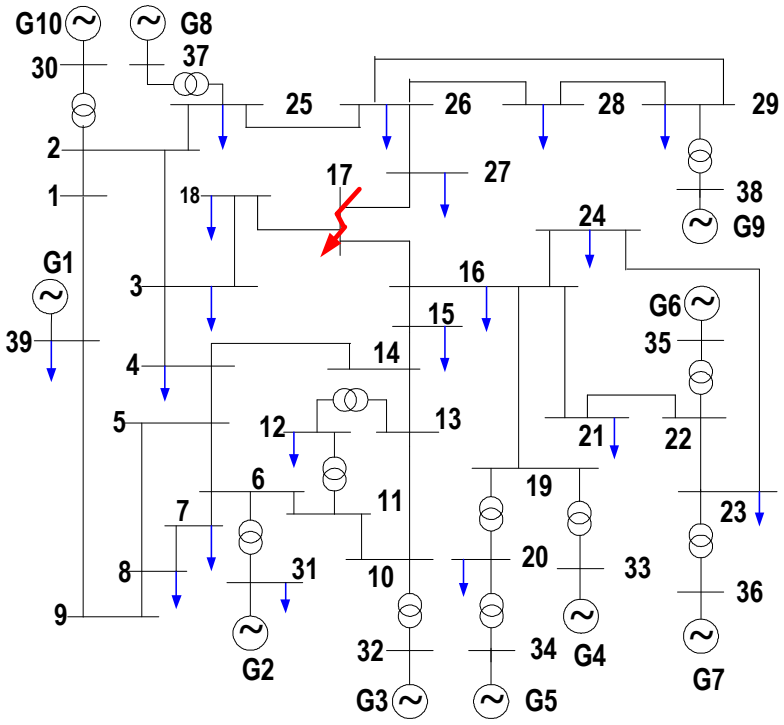
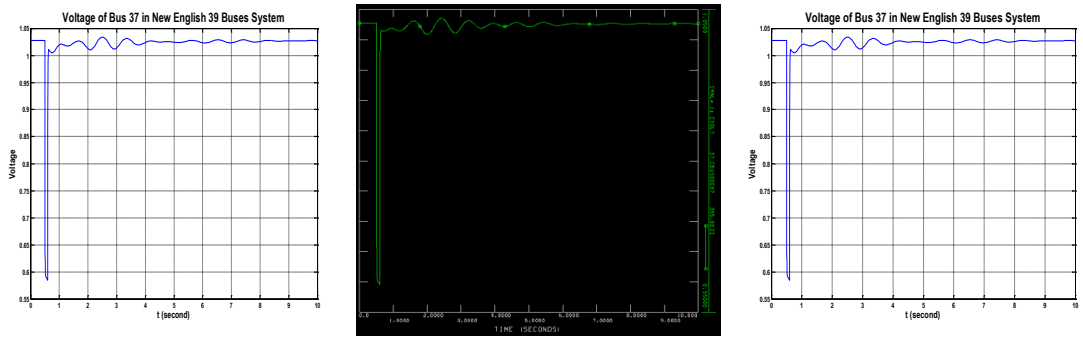
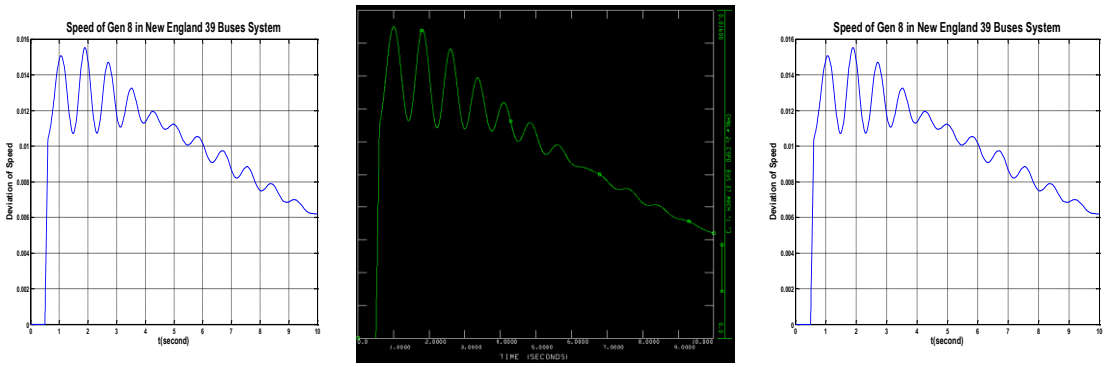



**Figure 3.8 New England 39 Bus 10 Gen System**




(i) HSET-TDS with Gen 6 (ii) PSS/E (iii)HSET-TDS with GENROU

(a) Voltage of Bus37




(i)HSET-TDS with Gen 6 (ii) PSS/E (iii)HSET-TDS with GENROU

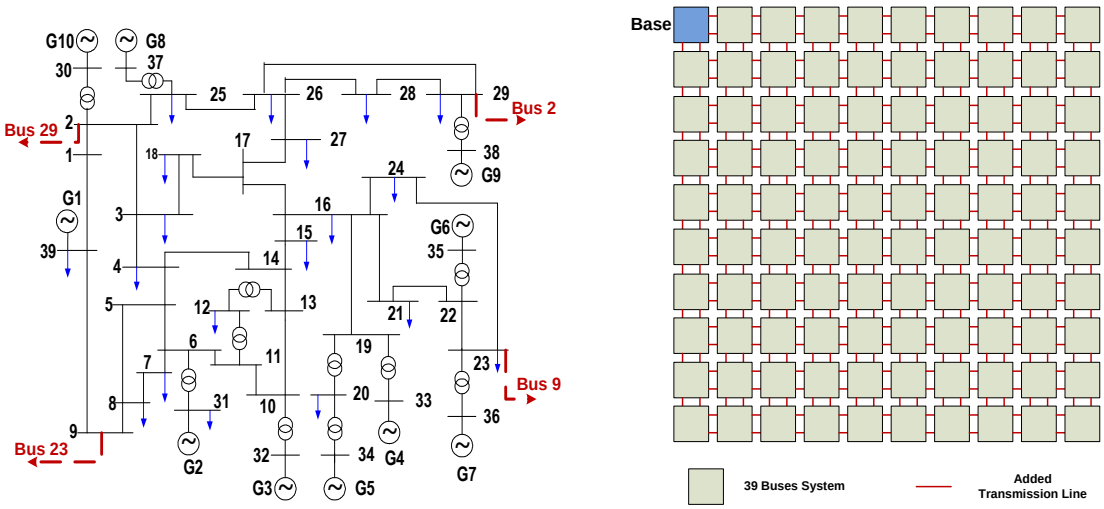
(b) Deviation of Speed on Gen8

**Figure 3.9 Simulation Results of New England 39 Buses System**

### 3.3.2 Case of Expanded 3900 buses system

In order to verify the capability of HSET-TDS for solving large systems, a large system was constructed from the basic 39-bus system. The idea of expanding New England 39-bus system was to replicate the system several times and then connect the buses between each replicated system. To guarantee the stability of this large system, the transmission lines connecting each of the 39 buses system have small impedance. This approach is similar to the way large systems tend to be connected today using ultra-high-voltage transmission lines.

In this case, the 39-bus system is going to be replicated 100 times forming the mesh shown in Figure 3.10. Buses 2, 9, 23, and 29 will be connected to adjacent systems. The resulting system contains 3900 buses, 1000 generators, and 4960 lines. In the DAE system, there are 6000 differential equations, 18000 algebraic equations, and 24000 variables when Gen6 is used.



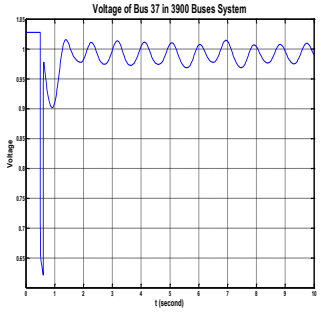

(a)Original 39 Buses System (b)Expanding Mesh

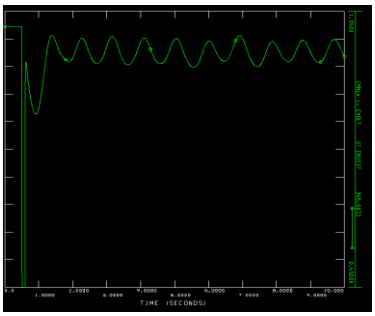
**Figure 3.10 Expanded System from New England 39 Buses System (10×10)**

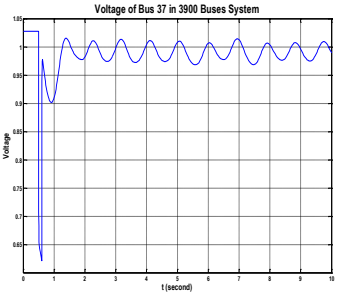
The selected event in the case is a bus fault on bus 17 starting from at 0.5s and lasting for 0.1s. The voltage of bus 37 and the speed of G8 will be monitored. The simulation lasts for 10 seconds. Figure 3.11 shows the simulation results produced by both HSET-TDS and PSS/E.

From the simulation results shown in Figure 3.9 and Figure 3.11, it can be seen that the results using the generator model of Gen6 and GENROU in HSET-TDS are identical. However, compared with the GEN6 model, since there are two more algebraic variables in GENROU ($\Psi_d$ and $\Psi_q$) there are more algebraic equations in the DAE system when the

GENROU model are used. Furthermore, it can be seen that the simulation results from HSET-TDS are nearly the same as those from PSS/E. This validation of HSET-TDS makes the simulation results in the following chapters more convincing and trustworthy.

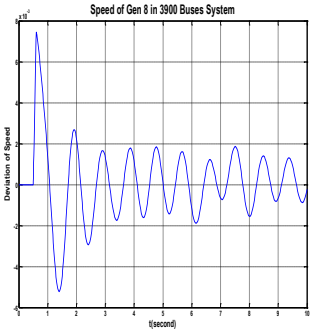


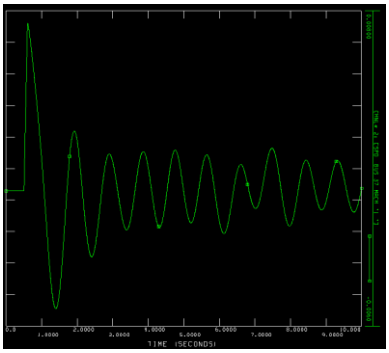



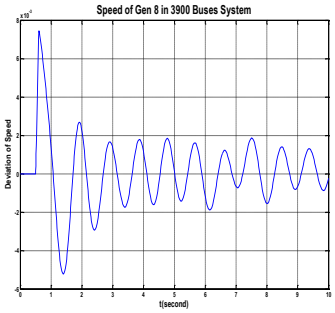



(i)HSET-TDS with Gen 6 (ii) PSS/E (iii)HSET-TDS with GENROU

(a) Voltage of Bus37






(i)HSET-TDS with Gen 6 (ii) PSS/E (iii)HSET-TDS with GENROU

(b) Deviation of Speed on Gen8

**Figure 3.11 Simulation Results for the Expanded 3900-Bus System**

# CHAPTER 4 HAMMER-HOLLINGSWORTH 4 (HH4)

In this Chapter, a new integration method, Hammer-Hollingsworth 4 (HH4), will be introduced. The motivation for exploring this method lies in the fact that the traditional integration method adopted by most of commercial software is the Trapezoidal rule, which is able to deal with not only the stiffness problem but also with the hyper-stability problem. However, since the Trapezoidal rule is of second-order precision, the integration step size must be small enough to guarantee required precision. We will try to find a new integration method that has the same stability attribute as the Trapezoidal rule but has the advantage of higher precision than trapezoidal rule.

## 4.1 Traditional Integration Methods for Power System Simulation

The analysis of stiffness [6, 24, 25, 26, 27] and hyper-stability [12] involves numerical stability analysis of integration methods, so here we will focus on stability analysis of several integration methods used in current commercial software: trapezoidal rule, theta-method, and Adams-BDF (backward differentiation formula) methods. We also describe the local truncation error and the numerical precision of these methods. Section III will provide similar analysis for the HH4 method.

### 4.1.1 Trapezoidal rule

The trapezoidal rule is a second-order implicit integration method which has been adopted in many commercial applications, as indicated in Table 2.2. For an ODE system

$$\begin{cases} \dot{x} = f(x) \\ x(t_0) = x_0 \end{cases} \tag{4.1}$$

the trapezoidal rule can be described as

$$x_{n+1} = x_n + \frac{h}{2}\left[f(x_n) + f(x_{n+1})\right] \tag{4.2}$$

where $h$ is the integration step size.

For the DAE system of (2.1), the trapezoidal rule can be described as

$$\begin{cases} \mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}\left[\mathbf{f}(\mathbf{x}_n, \mathbf{y}_n) + \mathbf{f}(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})\right] \\ 0 = \mathbf{g}(\mathbf{x}_{n+1}, \mathbf{y}_{n+1}) \end{cases} \tag{4.3}$$

We will discuss the trapezoidal rule from two aspects: numerical stability and numerical precision.

#### 4.1.1.1 Numerical stability of the trapezoidal rule

Assume that the general form of a differential equation is $\dot{x} = f(t, x)$. We linearize $f$ in its neighborhood as follows.

$$\dot{x} = f(t_n, x_n) + (t - t_n)\frac{\partial f}{\partial t}\bigg|_{(t_n, x_n)} + (x - x_n)\frac{\partial f}{\partial x}\bigg|_{(t_n, x_n)} + \cdots \tag{4.4}$$

The high order items can be omitted. Let $\lambda = \frac{\partial f}{\partial x}\Big|_{(t_n, x_n)}$, and we can get

$$\dot{x} = \lambda x + f(t_n, x_n) + (t - t_n)\frac{\partial f}{\partial t}\bigg|_{(t_n, x_n)} - \lambda x_n \tag{4.5}$$

If $\lambda \neq 0$, we can perform the transformation $\bar{x} = x + \frac{1}{\lambda}\left[ f(t_n, x_n) + (t - t_n + \frac{1}{\lambda})\frac{\partial f}{\partial t}\Big|_{(t_n, x_n)} - \lambda x_n \right]$,

Thus (4.1) can be transformed to

$$\dot{\bar{x}} = \lambda \bar{x} \tag{4.6}$$

Expression (4.6) is usually called a test equation (see the definition as follows). For a set of differential equations, $\lambda_i$ are the eigenvalues of the Jacobian matrix.

Next we apply to expression (4.6) a numerical method, for example, the forward Euler method, to produce

$$x_{n+1} = x_n + h\lambda x_n = (1+h\lambda)x_n = R(h\lambda)x_n = R(z)x_n \qquad \textbf{(4.7)}$$

where $z = h\lambda$. Assume that there is a disturbance $\delta_n$ on $x_n$, with a resulting disturbance on $x_{n+1}$ of $\delta_{n+1}$.

Then, $\delta_{n+1} = R(z)\delta_n$. If we want $|\delta_{n+1}| \le |\delta_n|$, we just require $|R(z)| \le 1$.

**Definition[27]:** The function $R(z)$ is called the ***stability function*** of the method. It can be interpreted as the numerical solution after one step for

$$\dot{x} = \lambda x, \qquad \text{with } x_0 = 1, \quad z = h\lambda,$$

which is the famous ***Dahlquist test equation***. The set

$$S = \{ z \in \mathbf{C}; \quad |R(z)| \le 1 \}$$

is called the ***stability domain*** of the method.

It can be seen that the stability function of Forward Euler method is $R(z) = 1+z$, a unit circle in the complex plane as shown in Figure 4.1. If we apply the trapezoidal rule (4.2) to the Dahlquist test equation, we can acquire the stability function of the trapezoidal rule,

$$R(z) = \frac{1+z/2}{1-z/2} \qquad \textbf{(4.8)}$$

The stability domain of the trapezoidal rule, the whole left part of the complex plane, is shown in Figure 4.1 The trapezoidal rule can deal with the stiffness problem, since it possesses the attribute of A-Stability. A method whose stability domain includes the left-half plane is called A-Stable [27]. If the stability domain of an integration method consists of only the left half of the complex plane, the method is symmetrically A-stable [12]. The advantages

of A-stability can be explained by comparing the trapezoidal rule with one of the explicit methods, e.g., the forward Euler method. For an appropriate integration step and a stable ODE system (i.e., where system eigenvalues at initial equilibrium all have negative real parts), all the values of $h\lambda_i$ are located inside the unit circle of the forward Euler stability domain, e.g., at point A in Figure 4.1. For a larger integration step, $h\lambda_i$ will increase and possibly fall outside the unit circuit, e.g., at point B in Figure 4.1, outside the stability domain of the forward Euler method. Thus, the forward Euler method can cause a stable ODE system to provide an unstable simulation result because of divergent numerical error. Of course, we can decrease the integration step so that all the values of $h\lambda_i$ are within the circle so that the forward Euler method provides the correct result; however, the cost is increased calculation time.

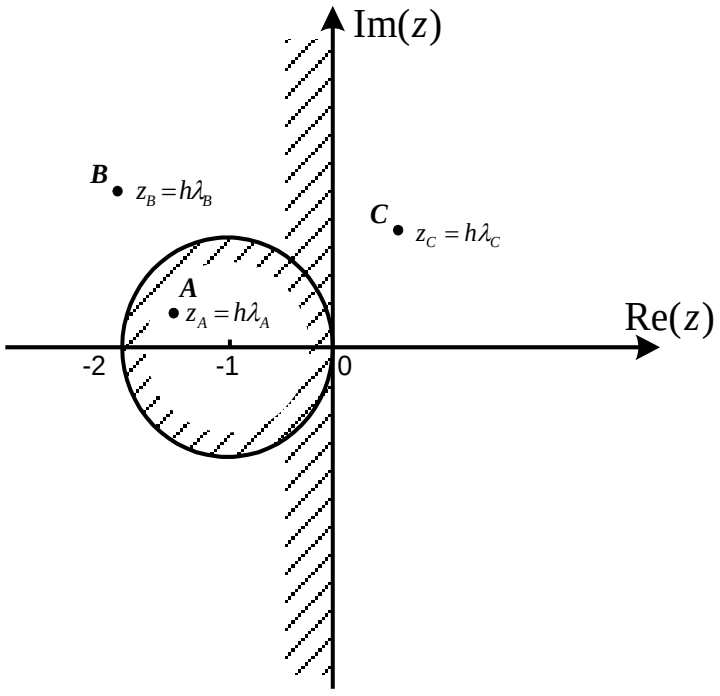



**Figure 4.1 Stability Domains of the Trapezoidal Rule and Forward Euler Method**

Alternatively, for a stable ODE system, the trapezoidal rule can not destabilize it, since its stability domain is only the left part of complex domain, so no value of *h* can cause *R*(*z*), as given in (4.8), to exceed 1 This is why the trapezoidal rule is classified as a symmetrically A-stable method. Clearly, the forward Euler method is not A-stable. In fact, none of the explicit methods, including all explicit Runge-Kutta methods, are A-stable. Reference [27] provides the stability function of any explicit Runge-Kutta method with order *p*, and it shows that if the Explicit Runge-Kutta method is of order *p*, its stability function can be expressed as

$$R(z) = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^p}{p!} + O(z^{p+1}) \qquad \textbf{(4.9)}$$

From (4.9), it can be seen that $|R(z)|$ is not less than 1 for any $z$ with negative real part. The stability domains of Runge-Kutta methods with order 1 to 4 are shown in Figure 4.2, and they do not cover all the left-half part of the complex plane. Therefore, the explicit Runge-Kutta methods may destabilize a stable ODE system if the integration step is not small enough.

This discussion provides the rationale behind the fact that so many commercial solvers employ the trapezoidal rule, i.e., that it is symmetrically A-stable. This attribute also implies that the method will not stabilize unstable ODE systems; if an eigenvalue has a positive real part and $z$ is in the right-half part of the complex plane, it is not possible to obtain a stable simulation via choice of integration step size. Point C in Figure 4.1 shows this scenario, and we can see that the trapezoidal rule is able to give an unstable result, since $|R(z)|>1$ and $|x_{n+1}|>|x_n|$.

The trapezoidal rule is numerically stable for dealing with stiffness problems; however this does not mean that the numerical error is always small or that the integration step can be arbitrarily large. A numerical error always exists for any nonzero value of integration step, and this error depends on the numerical precision of the trapezoidal rule. It is thus necessary to analyze the numerical precision of trapezoidal rule, as discussed in the next section.

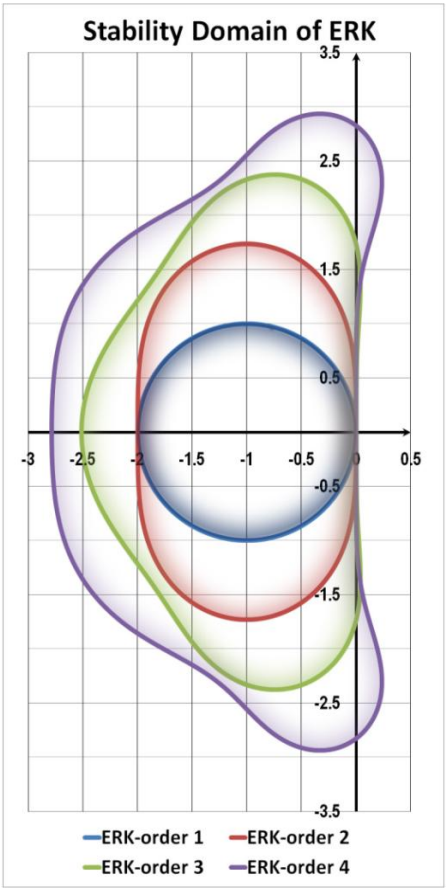



**Figure 4.2 Stability Domain of Explicit Runge-Kutta (ERK) Method (order 1 to 4) [27]**

#### 4.1.1.2 Numerical precision of trapezoidal rule

In the following discussion, we use two terms: local error and local truncation error. The term "local" distinguishes such an error from a "global" error. Local errors do not involve error existing in previous steps, so we can consider the previous step value to be an exact solution. In contrast, global errors include all deviations from the exact solution, the accumulated local error. However, a global error is difficult to evaluate, and it is not useful in controlling the integration step size. The local truncation error is a component of the local error, recognized in the local truncation error expression as the term with the smallest power of *h*.

The numerical precision of an integration method can be evaluated as the local error [36],the difference between the next step value produced by this method and the exact next step value. For the ODE system (4.1), the exact value of the next step can be expressed using Taylor series expansion as:

$$x_{n+1} = x(t_n + h) = x_n + hf(x_n) + \frac{h^2}{2!} f'(x_n) f(x_n) + \frac{h^3}{3!}\left[f''(x_n) f^2(x_n) + f^{2\prime}(x_n) f(x_n)\right] + O(h^4) \quad \textbf{(4.10)}$$

The next step value by the trapezoidal rule in expression (4.2) can be expressed as (4.11), where $f(x_{n+1})$ in expression (4.2) can be expanded by Taylor series expansion as:

$$x_{n+1} = x_n + hx'(t_n) + \frac{h^2}{2!} x''(t_n) + \frac{h^3}{4} x'''(t_n) + O(h^4) \quad \textbf{(4.11)}$$

Then the local error $|E(x_{n+1})|$ of the trapezoidal rule is the difference of right[hand parts of expressions (4.10) and (4.11), or

$$\left|E(x_{n+1})\right| = \left|\frac{h^3}{12}x'''(t_n) + O(h^4)\right| \qquad \textbf{(4.12)}$$

In (4.12), the term $|h^3x'''(t_n)/12|$ is the local truncation error. Because the lowest power of $h$ in the local error expression is 3, we conclude that the trapezoidal rule guarantees precision $h^2$ for simulation results, a significant weakness that motivates the search for an improved method, as described in Section 4.2.

### 4.1.2 Theta method

There are two forms for expressing the theta method. We express each form for the ODE system (2). The first form of the theta method [36] is:

$$x_{n+1} = x_n + hf\left[(1-\theta)x_n + \theta x_{n+1}\right] \qquad \textbf{(4.13)}$$

where $\theta$ is a parameter with $0 \le \theta \le 1$. The second form of the theta method [11, 37, 38] is:

$$x_{n+1} = x_n + h\left[(1-\theta)\,f(x_n) + \theta\, f(x_{n+1})\right]. \qquad \textbf{(4.14)}$$

where $\theta$ is a parameter with $0 \le \theta \le 1$. This second form is adopted in EXSTAB, so we will focus on the analysis of stability and precision for this form.

#### 4.1.2.1 Numerical stability of the Theta Method

Applying the theta method (4.14) into the Dahlquist test equation, we can acquire its stability function

$$R(z) = \frac{1+(1-\theta)z}{1-\theta z} \qquad \textbf{(4.15)}$$

Thus, the stability domain of the theta method (11) is

$$S_{theta-M} = \left\{ z \in \mathbf{C}; \quad \left|\frac{1+(1-\theta)z}{1-\theta z}\right| \le 1 \right\}$$

We classify the possible values of $\theta$ into four cases:

- If $\theta = 0.5$, the theta method becomes the trapezoidal rule, and the stability domain is the whole left part of the complex plane.
- If $\theta = 0$, the theta method becomes the Forward Euler method, and the stability domain is a unit circle in the left half of the complex plane.
- If $\theta = 1$, the theta method becomes the the Backward Euler method, and the stability domain is the whole complex plane except for the unit circle centered at (1,0).
- If $\theta \neq 0.5$, then the stability domain of the theta method (4.14) can be written as

$$S_{theta-M} = \left\{ z \in \mathbf{C}; \quad \left| z + \frac{1}{1-2\theta} \right| \leq \left| \frac{1}{1-2\theta} \right| \right\}$$

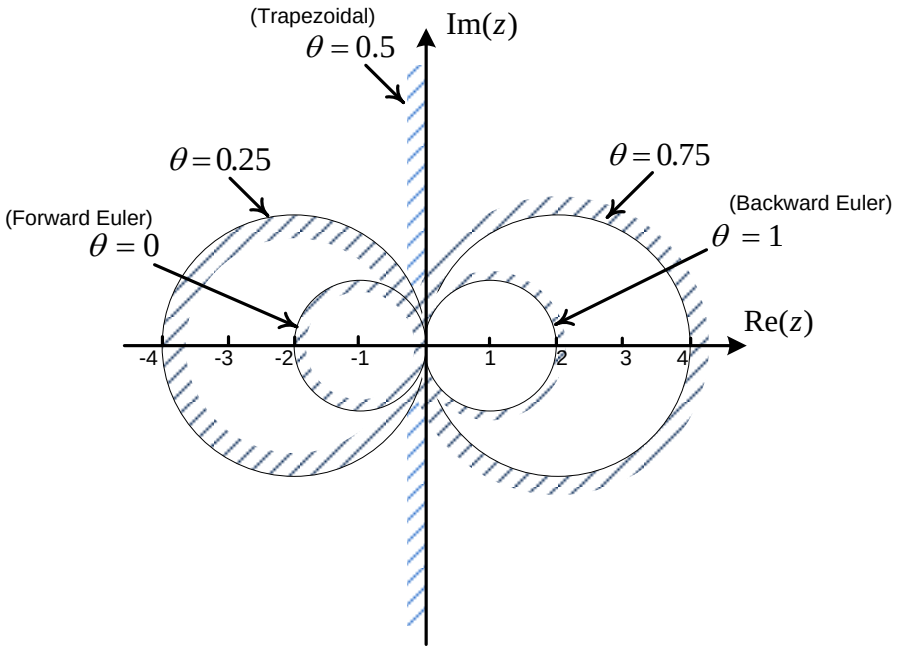



**Figure 4.3 Stability Domain of Theta Method**

Figure 4.3 shows the stability domain when $\theta$ is 0, 0.25, 0.5, 0.75, and 1, demonstrating that the theta method loses A-stability when $\theta$<0.5. Like the explicit methods

(e.g., Forward Euler) , the theta method with $\theta$<0.5 will exhibit numerical instability (stable ODE systems will be unstable in simulation) if the integration step is chosen too large.

When $\theta$>0.5, the stability domain contains the left-half complex plane along with some area of the right-half complex plane, so the theta method ($\theta$>0.5) is A-stable. However, the region of the stability domain lying in the right-half complex plane can cause hyper-stability[12], as described in the following.

Hyper-stability refers to the situation when the ODE system is unstable but the simulation exhibits a stable response. We use the following example to illustrate this phenomenon. Consider the following ODE system:

$$\begin{cases} \dot{x}_1 = 100x_1 - 400x_2 \\ \dot{x}_2 = 100x_1 + 100x_2, \quad x_1(0) = 2, x_2(0) = 2 \end{cases} \tag{4.16}$$

where (4.16) is a linear system with two eigenvalues $100\pm j200$. According to Lyapunov stability theory, system (4.16) is unstable. Figure 4.4 shows the simulation results of $x_1$ and $x_2$ and the corresponding points of $z$ produced by the theta method when $\theta$=1 (the Backward Euler). It can be found that, when $h=10^{-4}$, $z_1=\lambda_1 h=0.01+j0.02$, which is outside the stability domain, the simulation result is divergent as indicated in Figure 4.4b; this response is very close to the exact solution, which is simulated with $h=10^{-5}$ and shown in Figure 4.4a. However, when $h$=0.005, $z_1=\lambda_1 h=0.5+j1$, which is within the stability domain, the simulation results turn out to be convergent, as indicated in Figure 4.4c. Figure 4.4d shows the system simulated by the integration method proposed in this paper and to be discussed in Section 4.2.

The cause of hyper-stability can be investigated from the perspective of the stability function. For example, when the Backward Euler method is used to solve an unstable ODE

system, we require $R(z)=\left|\frac{1}{1-z}\right|<1$ if $z$ is in the right-half complex plane but outside the unit circle centered at (1,0). The method is attractive in that it imposes numerical error convergence such that $|\delta_{n+1}|\leq|\delta_n|$. However, the method also forces $|x_{n+1}|\leq|x_n|$, leading to incorrect simulation results when the ODE system has positive eigenvalues. Therefore, the stability domain of a suitable integration method should not include the right part of complex plane to ensure avoidance of hyper-stability.

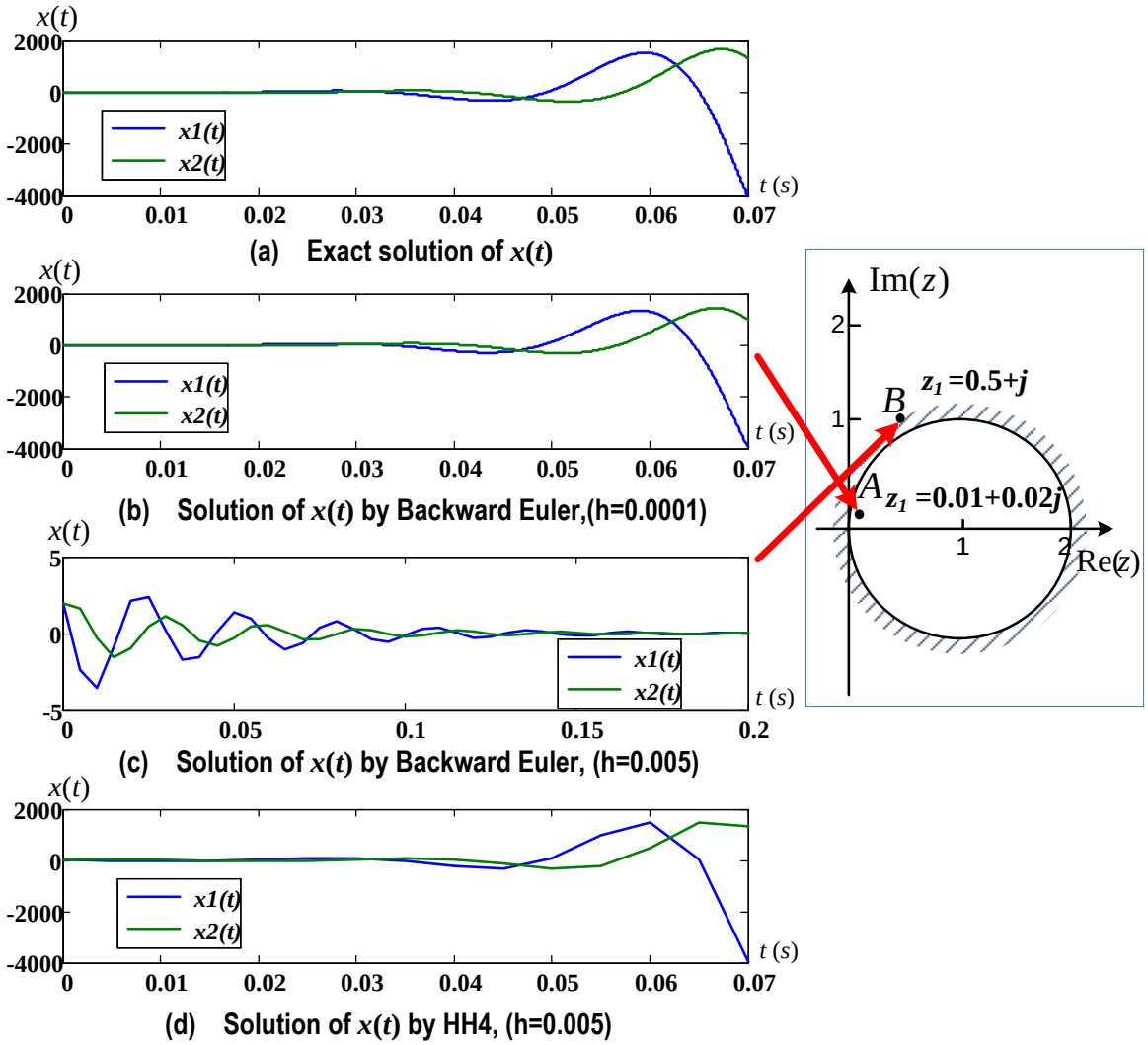



**Figure 4.4 An Example about Hyper-Stability Problem**

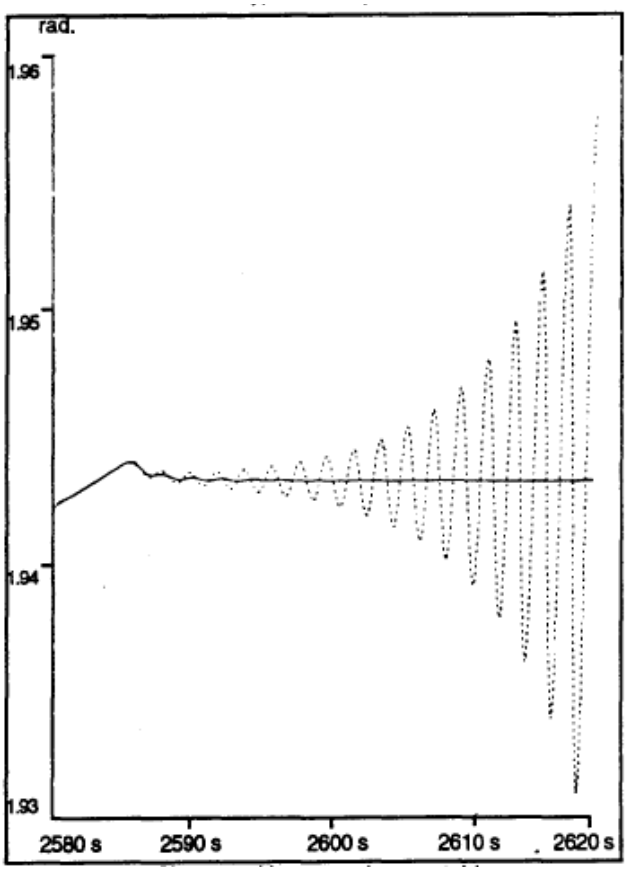



**Figure 4.5 An Practical Example about Hyper-Stability Problem [12]**

A practical example of the hyper-stability has been discussed in [12]. Assume that a single unit is connected to an infinite bus (presenting large external system) by step-up transformer and transmission line. The external voltage of the unit is set with an active power feedback control system via voltage regulator. Besides, the rotor current limit and an internal angle limiting loop is set up inside the generator. After the system reach the steady state, a voltage set-point decrease ramp (0.01 per unit per second) is applied to the generator, and the internal angle is adjusted to be a large value until it reaches an upper threshold which triggers the control of angle limiting loop. The simulation result of angle is expected to become unstable after the voltage set-point decrease ramp. The practical simulation results in Fig. 4.5 shows that the trapezoidal rule (with symmetrical stability domain) only needs the precision

of $0.5\times10^{-3}$ to capture the unstable simulation results, while BDF 2 needs the precision of $10^{-5}$ to acquire the correct simulation results.

There have been some reports describing how to decrease numerical oscillations observed in the theta method when $\theta$ is chosen slightly greater than 0.5. Reference [39] improved the trapezoidal rule with a damping term, and this modified trapezoidal rule is actually an implementation of the theta method. The damping term can make $\theta$ >0.5, and thus force the simulation to avoid numerical oscillation due to pure imaginary eigenvalues which can make $|R(z)|=1$ . Paper [11] provides a suggested value of theta, $\theta$=0.53, for use in (4.14), and this also decreases numerical oscillation. However, Figure 4.6 shows that there is still a region in the right-half plane which might lead to hyper-stability. Additionally, the precision of the theta method will be altered if $\theta\neq0.5$, as described in the next section.

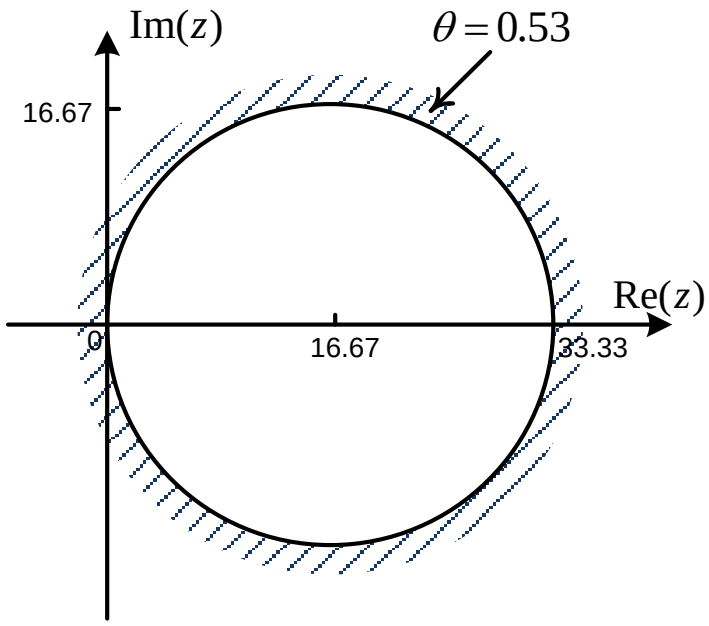



**Figure 4.6 Stability Domain of Theta Method ($\theta$=0.53)**

#### 4.1.2.2 Numerical precision of trapezoidal rule

In a manner similar to the analysis of the trapezoidal rule, $f(x_{n+1})$ in expression (4.14) can be expanded by Taylor series expansion, and the next step value produced by the theta method (4.14) can be written as

$$x_{n+1} = x_n + hx'(t_n) + \theta h^2 x''(t_n) + \frac{\theta}{2} h^3 x'''(t_n) + O(h^4) \quad \textbf{(4.17)}$$

Comparing (4.17) with the exact $x_{n+1}$ from (4.10), the local truncation error of the theta method is

$$\left|E(x_{n+1})\right| = \left| \left(\frac{1}{2} - \theta\right) h^2 x''(t_n) + \left(\frac{\theta}{2!} - \frac{1}{3!}\right) h^3 x'''(t_n) + \left(\frac{\theta}{3!} - \frac{1}{4!}\right) h^4 x^{(4)}(t_n) + O(h^5) \right| \quad \textbf{(4.18)}$$

Inspection of (4.18) indicates that the theta method will lose second-order precision if $\theta \neq 0.5$. Hence, the trapezoidal rule, which is the theta method with $\theta=0.5$, is more precise.

### 4.1.3 Adams method and BDF

The methods of explicit Adams, implicit Adams, and BDF are multi-step integration methods for solving differential equations. The general formula of implicit Adams methods [36] can be written as

$$x_{n+1} = x_n + h[\beta_k f(x_{n+1}) + \cdots + \beta_0 f(x_{n-k+1})] \quad \textbf{(4.19)}$$

where $\beta_k$ is the coefficient for $k$ (the number of previous steps used to evaluate the next step). $\beta_k$ can be obtained by comparing exact solutions of $x_{n+1}$ in (4.10) and $x_{n+1}$ in (4.19), where Taylor expansion must be applied to terms $\beta_k f(x_{n+1})$, $\cdots$, $\beta_0 f(x_{n-k+1})$. The implicit Adams method with $k$=1 is shown in expression (4.20) [36], the same as the the trapezoidal rule, the

integration method used in [12] (BDF is also used in [9] but only for evaluating error of algebraic variables).

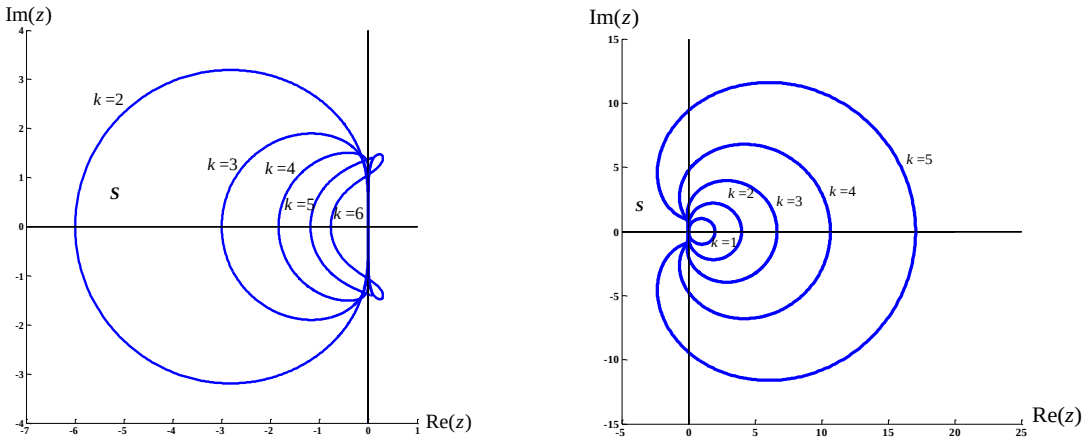
$$x_{n+1} = x_n + h[0.5f(x_n) + 0.5f(x_{n+1})] \quad \textbf{(4.20)}$$

The general expression for BDF (Backward Differentiation Formulas) [36] is:

$$\alpha_0 x_{n-k+1} + \alpha_1 x_{n-k+2} + \cdots + \alpha_k x_{n+1} = hf(x_{n+1}) \quad \textbf{(4.21)}$$

where $\alpha_k$ is the coefficient for step $k$. $\alpha_k$ can be obtained by comparing exact solutions of $x_{n+1}$ in (4.10) and $x_{n+1}$ in (4.19) where Taylor expansion needs to be applied to terms $f(x_{n+1})$ and $x_{n-k+1}$ ($k$>1).

Reference [27] supplies the stability domain of the implicit Adams method and the BDF, as shown in Figure 4.7, where we can observe in Figure 4.7a that the stability domains of the Adams method become smaller and smaller as the number $k$ is increased. The Adams method ($k$ > 1), therefore, will exhibit numerical instability if too large a time step is chosen. Figure 4.7b shows that the BDF method is exposed to hyper-stability for all $k \geq 1$, an observation which is also made in [12].




(a) Implicit Adam (b) BDF

**Figure 4.7 Stability Domina of Implicit Adam and BDF (*k* is order)**

## 4.2 Hammer-Hollingsworth 4 (HH4) Formula

The discussion in the last section confirms what others have apparently concluded, that the trapezoidal rule is an attractive integration method for time-domain simulation of power systems. The strength of the trapezoidal rule is that it avoids both numerical instability and hyper-stability. It can also achieve good simulation efficiency without sacrificing precision by using variable step integration [11, 12, 27, 36], which adjusts the integration step size based on estimated local error. When the estimated local error is less than a particular threshold, the integration step can be increased, increasing simulation efficiency; when the estimated local error is larger than this threshold, the integration step should be decreased, enabling control of precision. However, the integration step size for the trapezoidal rule is limited by its precision $h^2$; use of excessively large integration steps provide unacceptably inaccurate simulation results This motivates the following thought: if the local error can be reduced by adopting a more precise integration method (while maintaining the strengths of the trapezoidal rule), much larger integration steps may be possible, resulting in significant improvement in simulation efficiency.

In this section, an integration method based on quadratic functions, Hammer-Hollingsworth 4 [36, 40], is discussed. Hammer-Hollingsworth 4, which is a fourth order implicit Runge-Kutta method, was first introduced in [40]. Among all the different kinds of explicit and implicit Runge-Kutta methods, Hammer-Hollingsworth 4 possesses the following distinct attributes from other methods: 1) A-stability and the same stability domain as the trapezoidal rule, 2) higher precision than the trapezoidal rule. HH4 is one of the implicit integration methods used in HSET-TDS, and it is expected to exhibit better performance than the trapezoidal rule.

### 4.2.1 Formula of HH4

For the scalar ODE system of (4.1), reference [36] introduces an expression of the HH4 integration method which can be described as

$$\begin{cases} K_1 = f\left( x_n + h\left[ \frac{1}{4}K_1 + \left( \frac{1}{4} - \frac{\sqrt{3}}{6} \right) K_2 \right] \right) \\ K_2 = f\left( x_n + h\left[ \left( \frac{1}{4} + \frac{\sqrt{3}}{6} \right) K_1 + \frac{1}{4}K_2 \right] \right) \\ x_{n+1} = x_n + \frac{h}{2}\left[ K_1 + K_2 \right] \end{cases} \tag{4.22}$$

we modify this expression by letting

$$\begin{cases} \xi = x_n + h\left[ \frac{1}{4}K_1 + \left( \frac{1}{4} - \frac{\sqrt{3}}{6} \right) K_2 \right] \\ \eta = x_n + h\left[ \left( \frac{1}{4} + \frac{\sqrt{3}}{6} \right) K_1 + \frac{1}{4}K_2 \right] \end{cases} \tag{4.23}$$

Then we can acquire another expression of HH4, as:.

$$\begin{cases} (2\sqrt{3}-3)(\eta - x_n) + 3(\xi - x_n) = f(\xi) \\ 3(\eta - x_n) - (2\sqrt{3}+3)(\xi - x_n) = f(\eta) \\ x_{n+1} = x_n + \sqrt{3}(\eta - \xi) \end{cases} \tag{4.24}$$

There is a computational benefit of using this transformation: whereas (4.22) increases the number of non-zero elements in the Jacobian matrix, (4.24) maintains the original sparsity and therefore saves time during solution of the linear equations of the Newton iterations.

During the solution of HH4 two points $\eta$, $\xi$ are calculated by the first two equations in (22), and then the next step value $x_{n+1}$ is calculated using the third equation of (22) with values of points $\eta$, $\xi$ as illustrated in Figure 4.8

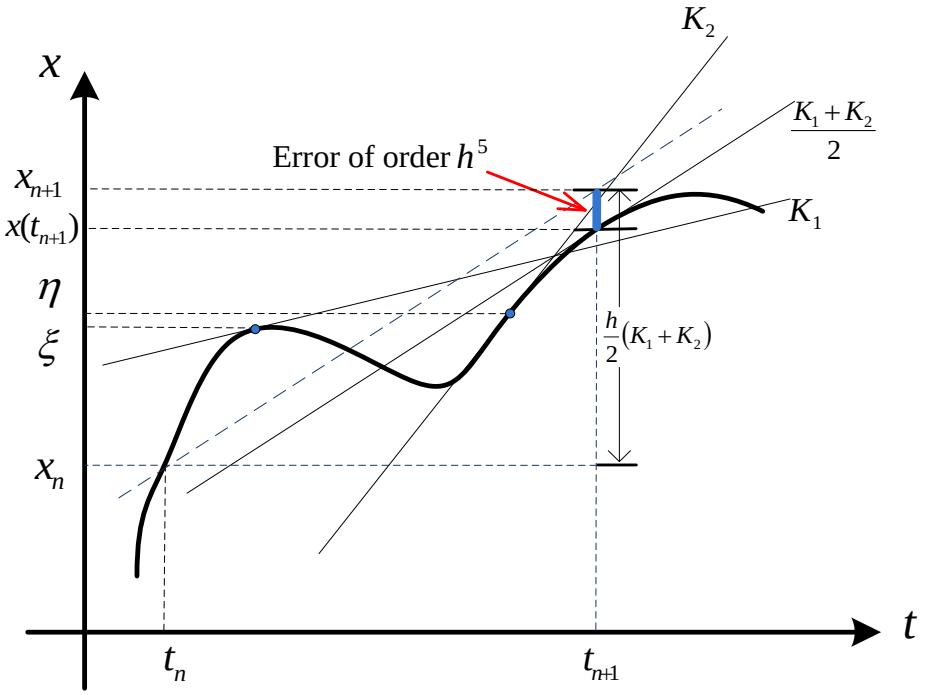



**Figure 4.8 Integration method of Hammer-Hollingsworth 4**

For the multidimensional DAE system (2.1), the solution of $\mathbf{x}_{n+1}$ using the HH4 integration method can be described as two stages, i) solution of vector $\boldsymbol{\eta}$, $\boldsymbol{\xi}$, and ii) solution of $\mathbf{x}_{n+1}$. For the first stage, we need to solve the following non-linear algebraic equations.

$$\begin{cases} (2\sqrt{3}-3)(\boldsymbol{\eta}-\boldsymbol{x}_n)+3(\boldsymbol{\xi}-\boldsymbol{x}_n)=\mathbf{f}\left(\boldsymbol{\xi},\boldsymbol{y}_{\xi}\right) \\ 0=\mathbf{g}\left(\boldsymbol{\xi},\boldsymbol{y}_{\xi}\right) \\ 3(\boldsymbol{\eta}-\boldsymbol{x}_n)-(2\sqrt{3}+3)(\boldsymbol{\xi}-\boldsymbol{x}_n)=\mathbf{f}\left(\boldsymbol{\eta},\boldsymbol{y}_{\eta}\right) \\ 0=\mathbf{g}\left(\boldsymbol{\eta},\boldsymbol{y}_{\eta}\right) \end{cases} \tag{4.25}$$

For the second stage we need to solve

$$\begin{cases} \boldsymbol{x}_{n+1}=\boldsymbol{x}_n+\sqrt{3}(\boldsymbol{\eta}-\boldsymbol{\xi}) \\ 0=\mathbf{g}(\boldsymbol{x}_{n+1},\boldsymbol{y}_{n+1}) \end{cases} \tag{4.26}$$

### 4.2.2 Numerical stability and precision of HH4

If we can apply the integration method of HH4 (4.22) to the Dahlquist test equation, we can acquire the stability function of HH4,

$$R(z)=\frac{1+z/2+z^2/12}{1-z/2+z^2/12} \tag{4.27}$$

The stability domain of HH4 is thus

$$S_{HH4}=\left\{ z\in \mathbf{C};\quad \left|\frac{1+z/2+z^2/12}{1-z/2+z^2/12}\right|\le 1 \right\} \tag{4.28}$$

as illustrated in Figure 4.9. This shows that the stability domain of HH4 is the left-half complex plane. The method of HH4, therefore, is symmetrically A-stable, and it avoids numerical instability and hyper-stability. Figure 4.4 exhibits the ability of HH4 to deal with hyper-stability. When $h$=0.005, the simulation results using backward Euler, Figure 4.4c, are incorrectly convergent, while the results using HH4, Figure 4.4d, are divergent, in agreement with Figure 4.4a.

Additionally, HH4 guarantees precision of $h^4$ with a local error of $O(h^5)$, which can be verified using the Taylor series expansion. The high precision of HH4 enhances simulation efficiency by allowing increased integration step size while maintaining precision. For example, if we require numerical precision of $10^{-4}$, the maximum integration step size for the trapezoidal rule must be less than 0.1 since its precision is second-order ($h^2=0.1^2=10^{-2}>10^{-4}$). In contrast, HH4 has fourth-order precision ($h^4$) and may therefore achieve numerical precision of $10^{-4}$ using an integration step as large as 0.1 ($h^4=0.1^4=10^{-4}$).

The "cost" of these good features is that the nonlinear algebraic equations that must be solved at each integration step have twice the number of dimensions as those that must be

solved using the trapezoidal rule, an attribute that can be observed in (4.25). However, when using the Newton method to solve these equations, the Jacobian is highly sparse, and therefore the "cost" can be minimized by utilizing a sparse linear-solver library.

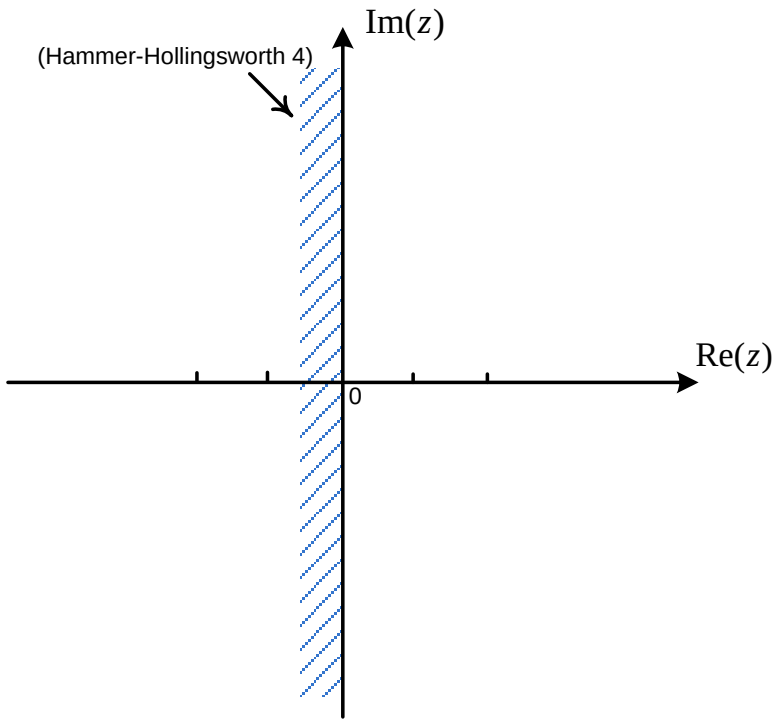



**Figure 4.9 Stability Domain of Hammer-Hollingsworth 4**

## 4.3 Time Step Control and Error Estimation

Step-control techniques and error estimation are used in many commercial time-domain simulation applications, including EXSTAB and EUROSTAG. We will describe implementation within our software application HSET-TDS for the HH4 integration method. Step-control techniques decrease or increase the integration step size based on an estimate of the local error incurred by the integration method or based on the number of iterations performed by of the nonlinear solver (generally a Newton method).

### 4.3.1 Time step control criteria

In HSET-TDS, integration step size is adjusted based on i) whether the norm of the vector of error estimation is larger or smaller than given threshold, or ii) whether the number of iterations within the nonlinear solver (a Newton method) exceeds a threshold chosen large enough that violation of this criterion implies the Newton iterations have diverged. The first criterion can be described as follows [11].

$$h_{\text{new}}=h_{old}[tolerance/\|e\|_{\infty}]^{1/p} \qquad \textbf{(4.29)}$$

where $\|e\|_{\infty}$ is the max norm of the estimated local truncation error, and $p$ is the order of the integration method. HSET-TDS adopts double thresholds , an upper threshold and a lower threshold, to control the integration step, If the error norm $\|e\|_{\infty}$ exceeds the upper threshold, the current integration step is decreased; if the error norm is smaller than the lower threshold, the next integration step is increased. This double-threshold approach, compared to a single-threshold approach, decreases the frequency at which the time step is changed and therefore improves simulation efficiency by reducing recalculations following a change of time step.

### 4.3.2 Estimation of truncation error

We consider the local truncation error to be the local error, which is justified since, for small $h$, higher-order terms in the error expression are negligible. For a $p$-order integration method, there are two methods for estimating its local truncation error: i) an extrapolation method, or ii) another integration method

#### 4.3.2.1 Extrapolation method

The main idea of the extrapolation method is to recalculate the value of the next point using two half-step calculations. The truncation error is then estimated as the difference

between the next-point value calculated by the original integration step and that estimated by the two half-steps. To describe this analytically, assume that $x(t_{n+1})$ is the exact solution, and $x_{n+1}^{[h]}$ is the numerical solution produced by an integration method of order $p$ with integration step of $h$, and $x_{n+1}^{[h/2]}$ is the numerical solution produced by the same integration method with an integration step size of $h/2$ . According to Gragg theory[36],

$$x_{n+1}^{[h]} = x(t_{n+1}) + \gamma(x_{n+1})h^p + O(h^{p+1}) \quad \textbf{(4.30)}$$

$$x_{n+1}^{[h/2]} = x(t_{n+1}) + \gamma(x_{n+1})\left(\frac{h}{2}\right)^p + O(h^{p+1}) \quad \textbf{(4.31)}$$

where $|\gamma(x_{n+1})\ h^p+O(h^{p+1})|$ is the local error for $x_{n+1}$ by the integration method, $|\gamma(x_{n+1})\ h^p|$is the local truncation error which must be evaluated, and $\gamma(x_{n+1})$ is a function associated with $x_{n+1}$. Since $\gamma(x_{n+1})$ does not depend on $h$, expression (4.30) can be subtracted from (4.31) to yield the following local truncation error estimation formula:

$$\left|\tilde{E}(x_{n+1})\right| = \left|\gamma(x_{n+1})h^p\right| \approx \left|\frac{2^p(x_{n+1}^{[h]} - x_{n+1}^{[h/2]})}{2^p - 1}\right| \quad \textbf{(4.32)}$$

**4.3.2.2 Error estimation by other integration methods**

Although the extrapolation method is able to estimate the main part of the truncation error, two extra steps of simulation are needed to calculate a more precise value of the next step point. An alternative approach for estimating local truncation error is to compare the simulation result with that from another auxiliary integration method having the same order or higher. Reference [11] has identified a specific integration method to use in error estimation for the theta method and the trapezoidal rule, as summarized in Table 4.1. We suggest use of Ceschino 4, a type of $4^{th}$ order explicit Runge-Kutta method, to estimate local

truncation error for HH4. The error estimation can be evaluated using the following expression.

$$\left|\tilde{E}(x_{n+1})\right| = c_e\left|x_{n+1} - \tilde{x}_{n+1}\right| \tag{4.33}$$

where $x_{n+1}$is the next step value calculated by one integration method, $\tilde{x}_{n+1}$ is the value calculated by another integration method, and $c_e$ is the error coefficient. Since the error term of HH4 is $h^5x^{(5)}(t_n)/720$ (see the following section), and that of the 4[th]-order explicit Runge-Kutta method is $h^5x^{(5)}(t_n)/120$ [27], it can be deduced that the error coefficient $c_e$ for HH4 and Ceschino 4 [36] can be set to be 1/5. In contrast, reference [11] indicates that the error coefficient is 1/6[1] when the theta method is used for integration and the explicit theta method is used for error estimation.

For the formula of HH4 in expression (4.22), letting $K_1' = \frac{dK_1}{dt}$ , $K_2' = \frac{dK_2}{dt}$, it can be shown that

$$\begin{cases} K_1 = x'(t_n) + \frac{1}{4}hK_1' + \left(\frac{1}{4} - \frac{\sqrt{3}}{6}\right)hK_2' \\ K_2 = x'(t_n) + \left(\frac{1}{4} + \frac{\sqrt{3}}{6}\right)hK_1' + \frac{1}{4}hK_2' \end{cases} \tag{4.34}$$

$$\text{and } K_1 + K_2 = 2x'(t_n) + \frac{1}{2}h\left(K_1' + K_2'\right) - \frac{1}{12}h^2\left(K_1'' + K_2''\right) \tag{4.35}$$

Letting $M=K_1+K_2$, then the next step value can be formulated as

$$x_{n+1} = x_n + \frac{h}{2}M = 2x'(t_n) + \frac{1}{2}hM - \frac{1}{12}h^2M'' \tag{4.36}$$

Replacing the derivatives of $M$ by expression (33), we obtain

[1] This coefficient is obtained theoretically when $\theta$=0.5 and $\omega$=1 for explicit theta method.

$$x_{n+1} = x_n + hx'(t_n) + \frac{h^2}{2!}x''(t_n) + \frac{h^3}{3!}x'''(t_n) + \frac{h^4}{4!}x^{(4)}(t_n) + \frac{h^5}{144}x^{(5)}(t_n) + O(h^6) \quad \textbf{(4.37)}$$

Comparing expression (8) with (35), the local error of HH4 can be expressed as follows.

$$\left|E(x_{n+1})\right| = \left|\left(\frac{1}{5!} - \frac{1}{144}\right)h^5 x^{(5)}(t_n) + O(h^6)\right| = \left|\frac{h^5}{720}x^{(5)}(t_n) + O(h^6)\right| \quad \textbf{(4.38)}$$

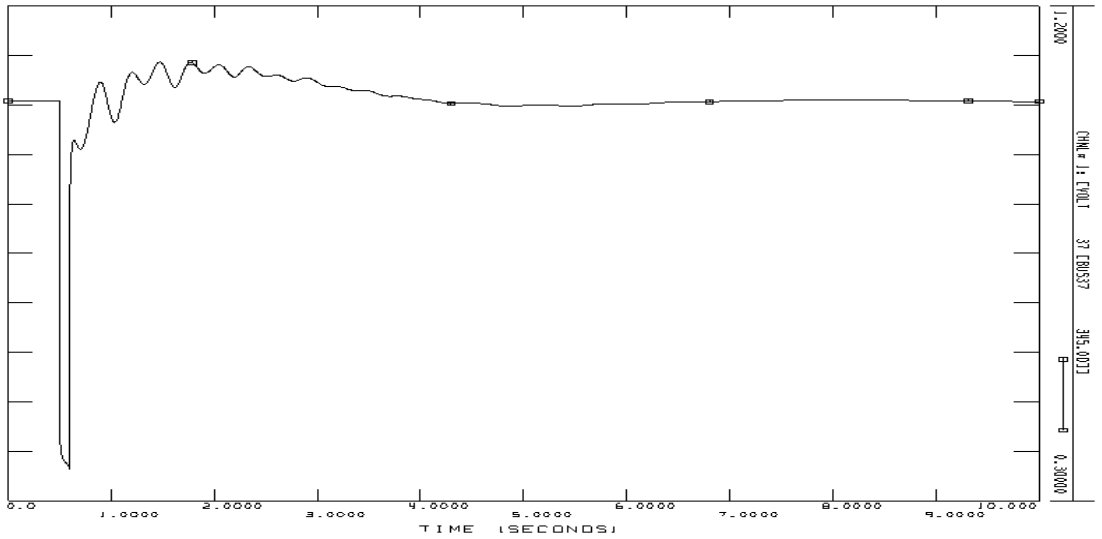
**Table 4.1 Integration methods for error estimation**

| *Integration Methods* | *Integrator for Error Estimation* |
|---|---|
| **Trapezoidal Rule** $x_{n+1} = x_n + \frac{h}{2}\left[f(x_n) + f(x_{n+1})\right]$ | **Explicit Adams 2** $\tilde{x}_{n+1} = x_n + h\left[\frac{3}{2}f(x_n) - \frac{1}{2}f(x_{n-1})\right]$ |
| **Theta Method** $x_{n+1} = x_n + h\left[(1-\theta)f(x_n) + \theta f(x_{n+1})\right]$ | **Explicit Theta Method [11]** $\tilde{x}_{n+1} = x_n + h\left(f(x_n) + \theta\left[f(x_n) - f(x_{n-1})\right]\right)$ |
| **Hammer-Hollingsworth 4** $\begin{cases}(2\sqrt{3}-3)(\eta - x_n) + 3(\xi - x_n) = f(\xi)\\ 3(\eta - x_n) - (2\sqrt{3}+3)(\xi - x_n) = f(\eta)\\ x_{n+1} = x_n + \sqrt{3}(\eta - \xi)\end{cases}$ | **Explicit Runge-Kutta method, Ceschino 4[36]** $\begin{cases}K_1 = f(x_n)\\ K_2 = f(x_n + hK_1/4)\\ K_3 = f(x_n + hK_2/2)\\ K_4 = f\left[x_n + h(K_1 - 2K_2 + 2K_3)\right]\\ \tilde{x}_{n+1} = x_n + h\left(K_1/6 + 4K_3/6 + K_4/6\right)\end{cases}$ |

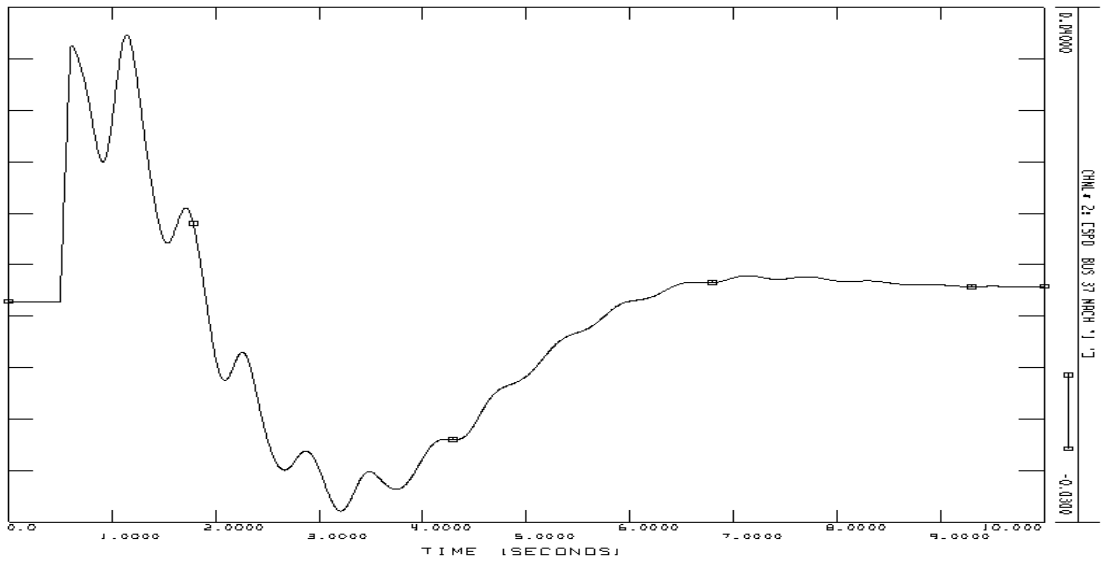
## 4.4 Simulation Results

In this section, the New England 39-bus system, an expanded New England 8775-bus system, and a modified PJM 13029-bus system are simulated within HSET-TDS to compare the integration methods of the trapezoidal rule and HH4 with fixed and variable steps. The simulation results are validated using PSS/E (version 30.3.2). The generator model, excitation model, and governor model used in the simulation cases is GENROU [34] (a six-order model), IEEET1 [34] as shown in Figure 3.3, and TGOV1 [34], as shown in Figure 3.5.

### 4.4.1 New England 39-bus system with 10 generators




**(a) Voltage of Bus 37**




**(b) Speed of Generator Connected to Bus 37**

**Figure 4.10 Simulation Results of New England 10 Gen 39 Bus System with fixed integration step in PSS/E**

The New England system with 10 generators and 39 buses is shown in Figure 3.8. HSET-TDS utilizes the direct-solution method to construct and solve the DAE system. The DAE system includes 110 ODEs and 190 algebraic equations. The initial values of the DAE

system are computed using the power-flow program in HSET-TDS, which is based on the Newton-Raphson method.

In order to compare the integration methods of the trapezoidal rule and HH4, a fault on bus 17 was simulated, starting at 0.5s and lasting for 0.1 seconds. The transient process was simulated for 10 seconds, and bus 37 voltage was monitored. The system was first simulated in PSS/E to provide a reference with which to validate results from HSET-TDS. Since the default integration step, 0.00833s, does not work in PSS/E, an integration step of 0.001s was selected. The simulation curve of bus 37 voltage is shown in Figure 4.10, which is from PSSPLT of PSS/E. Comparison between the integration methods both when the integration step size is fixed and when the integration step size is variable was made.

#### 4.4.1.1 Fixed Integration Step

Inntegration steps of 0.001s, 0.01s and 0.1s were selected for both the trapezoidal rule and HH4. The simulation results produced by the trapezoidal rule with h=0.001 is assumed to be the exact solution. The computation time is shown in Table 4.2, and the simulation curves (drawn by the 'plot' tool of MATLAB with the results from HSET-TDS), are shown in Figure 4.11,which are the same as the curves simulated from PSS/E shown in Figure 4.10. From the results shown in Figure 4.11, it can be seen that the simulation curves produced by the trapezoidal rule with h=0.01 are still acceptable, and the curves produced by the trapezoidal rule with h=0.1 deviates from the exact solution, while curves produced by HH4 with h=0.1 are well-matched with the exact solution. Additionally, the computational time used by HH4 with h=0.1 is much less than the time used by the trapezoidal rule with h=0.01.

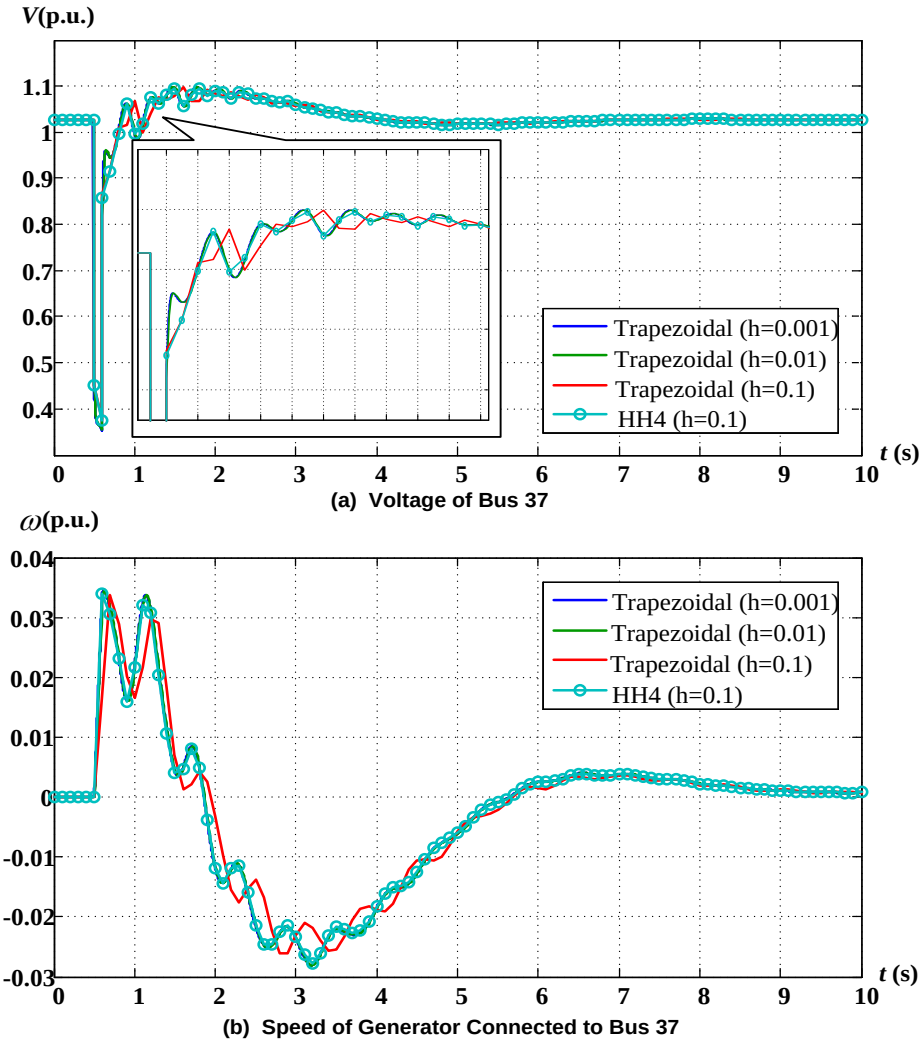



**Figure 4.11 Simulation Results of New England 39 Buses System with fixed integration step in HSET-TDS**

**Table 4.2 Simulation Results of New England 39 Buses System with Fixed Integration Step**

| Integration Methods | Integration Step (s) | Algebraic Solver | | Linear Solver Library | Simulation Time (s) | Correctness by Observation |
|---|---|---|---|---|---|---|
| | | Solver | Precision | | | |
| **Trapezoidal** | 0.001 | VDHN | $10^{-6}$ | SuperLU | 1.797 | (exact solution) |
| **Trapezoidal** | 0.01 | VDHN | $10^{-6}$ | SuperLU | 0.281 | Correct |
| **Trapezoidal** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 0.047 | Incorrect |
| **HH4** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 0.125 | Correct |
| **Simulation in PSS/E with integration step of 0.001** | | | | | around 1 s | |

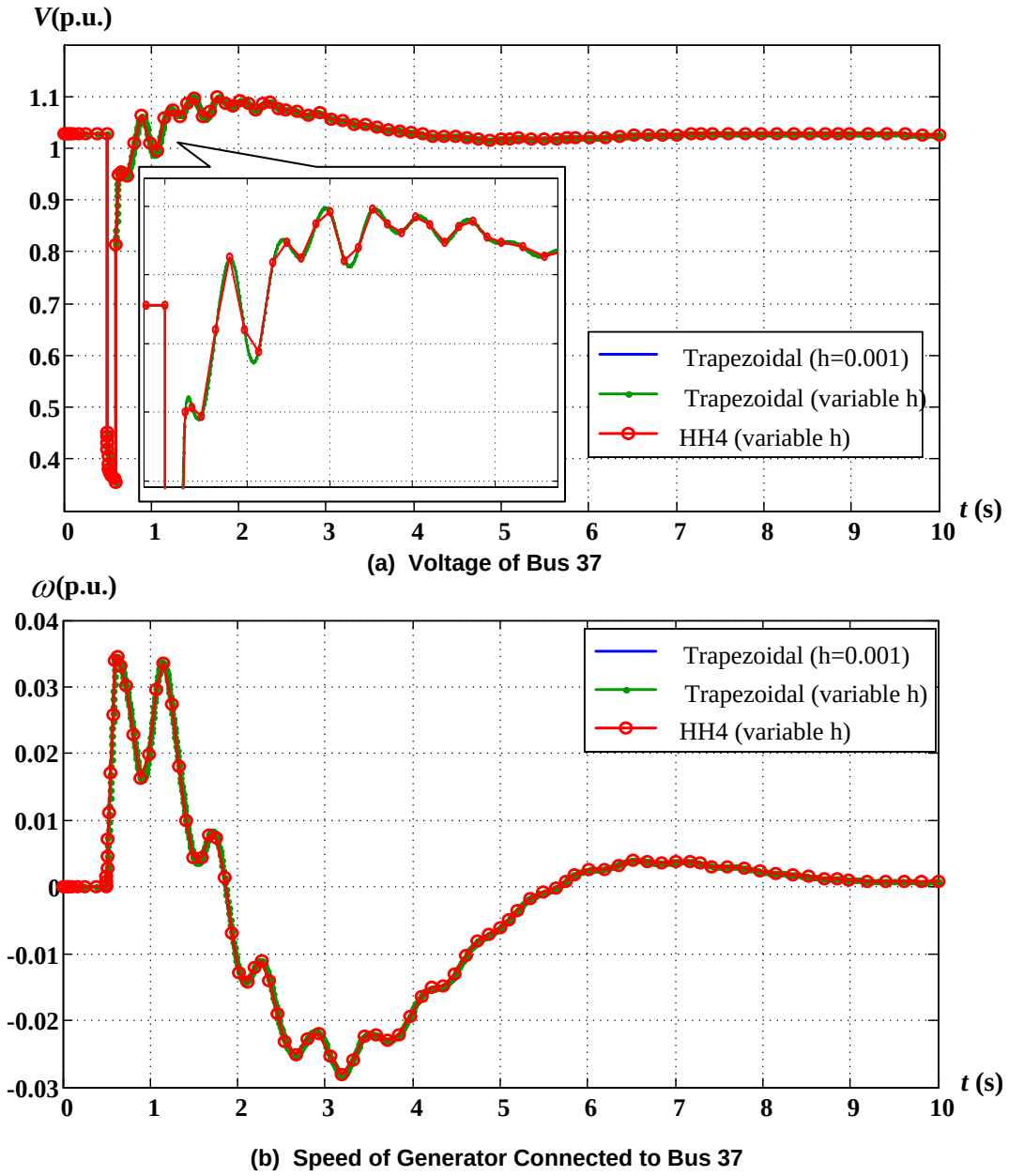



**Figure 4.12 Simulation Results of New England 10 Gen 39 Buses System with variable integration step in HSET-TDS**

#### 4.4.1.2 Variable Integration Step

HSET-TDS is developed with a variable-step technique, and the methods for error estimation are listed in Table 4.1. HSET-TDS adopts double thresholds, i.e., an upper and a

lower threshold, to control the integration step, as described in the previous section. The simulation results are shown in Table 4.3 and Figure 4.12. It is observed that the simulation by HH4 with variable step uses less integration steps and less simulation time than the simulation by trapezoidal rule with variable with no loss of precision.

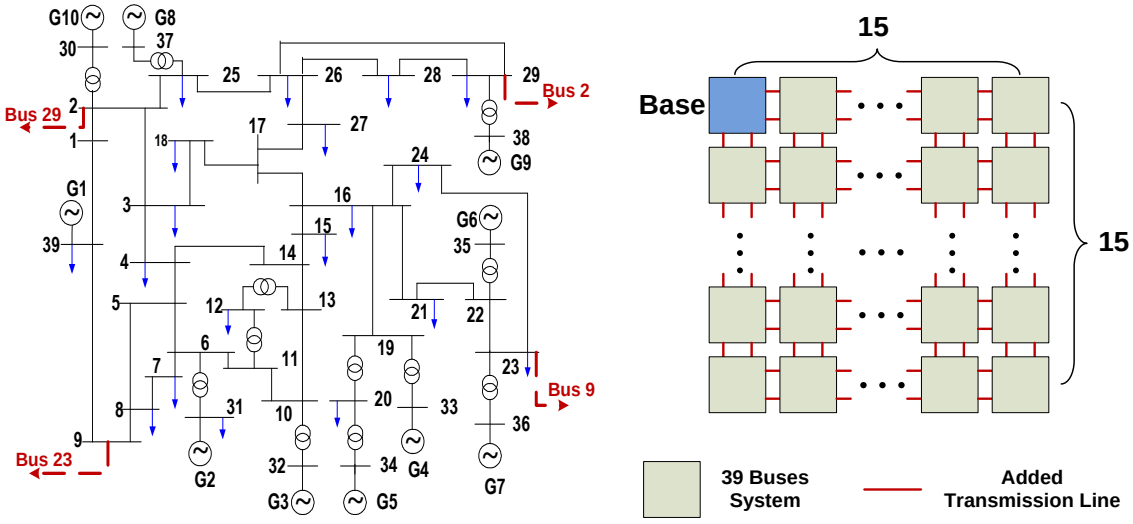
**Table 4.3 Simulation Results with Variable Integration Step**

| Integration Methods | | | | | Algebraic Solver | | Linear Solver Library | Simulation Time (s) |
|---|---|---|---|---|---|---|---|---|
| **Methods** | Upper Error Bound | Lower Error Bound | Maximum Step (s) | Num. of Integration Steps | Alg. Solver | Precision | | |
| **Trap.** | $5\times10^{-4}$ | $10^{-4}$ | 0.027742 | 751 | VDHN | $10^{-6}$ | SuperLU | 0.296 |
| **Trap.** | $10^{-4}$ | $5\times10^{-5}$ | 0.023554 | 1159 | VDHN | $10^{-6}$ | SuperLU | 0.421 |
| **Trap.** | $5\times10^{-5}$ | $10^{-5}$ | 0.020806 | 1542 | VDHN | $10^{-6}$ | SuperLU | 0.485 |
| **Trap.** | $10^{-5}$ | $5\times10^{-6}$ | 0.012266 | 2410 | VDHN | $10^{-6}$ | SuperLU | 0.735 |
| **Trap.** | $5\times10^{-6}$ | $10^{-6}$ | 0.011146 | 3346 | VDHN | $10^{-6}$ | SuperLU | 0.922 |
| **HH4** | $5\times10^{-4}$ | $10^{-4}$ | 0.221934 | 98 | VDHN | $10^{-6}$ | SuperLU | 0.188 |
| **HH4** | $10^{-4}$ | $5\times10^{-5}$ | 0.243508 | 125 | VDHN | $10^{-6}$ | SuperLU | 0.250 |
| **HH4** | $5\times10^{-5}$ | $10^{-5}$ | 0.166451 | 125 | VDHN | $10^{-6}$ | SuperLU | 0.203 |
| **HH4** | $10^{-5}$ | $5\times10^{-6}$ | 0.158328 | 170 | VDHN | $10^{-6}$ | SuperLU | 0.328 |
| **HH4** | $5\times10^{-6}$ | $10^{-6}$ | 0.126541 | 172 | VDHN | $10^{-6}$ | SuperLU | 0.281 |

### 4.4.2 Expanded New England 8775-Bus System with 2250 Generators

In order to test a large-scale system, we constructed a test system based on the New England 39-bus system. The construction scheme is as follows:

1) Assume there are 15×15 (225) subsystems, each of which is identical to the original New England 39-bus system;
2) Buses 2, 9, 23, 29 in each 39 bus subsystem are connected to adjacent subsystems as shown in Figure 4.13; the impedances of connecting branches are small in magnitude.
3) The dynamic elements (generators) within each subsystem are modeled as previously described for the original 39-bus system.

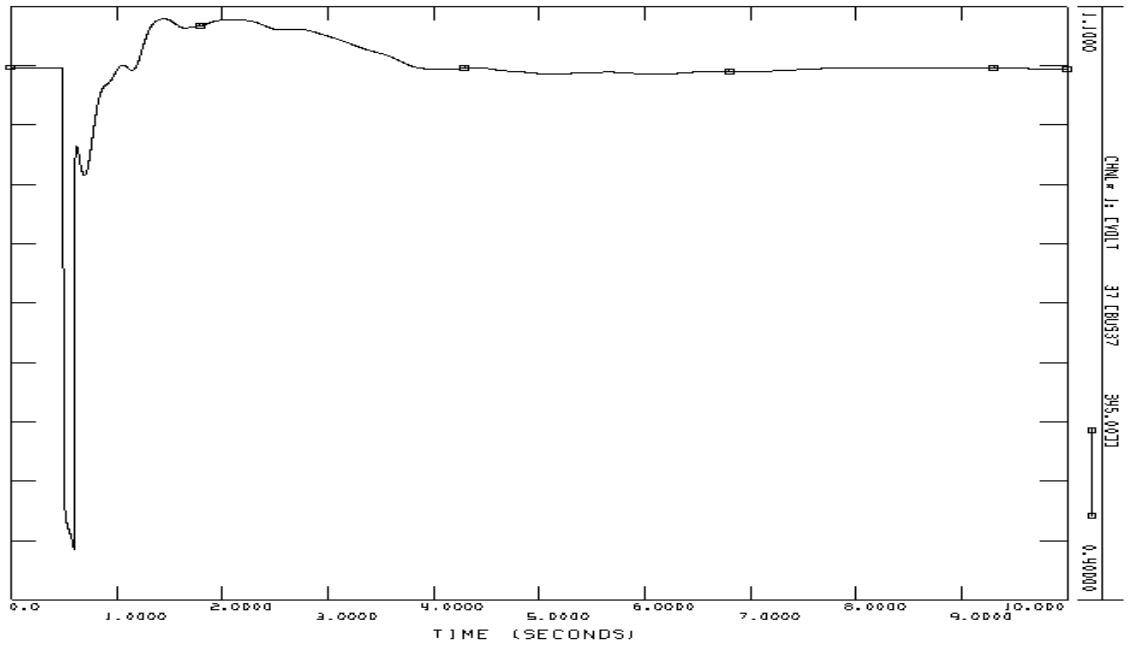



(a)Original 39 Buses System (b) Expanding Mesh

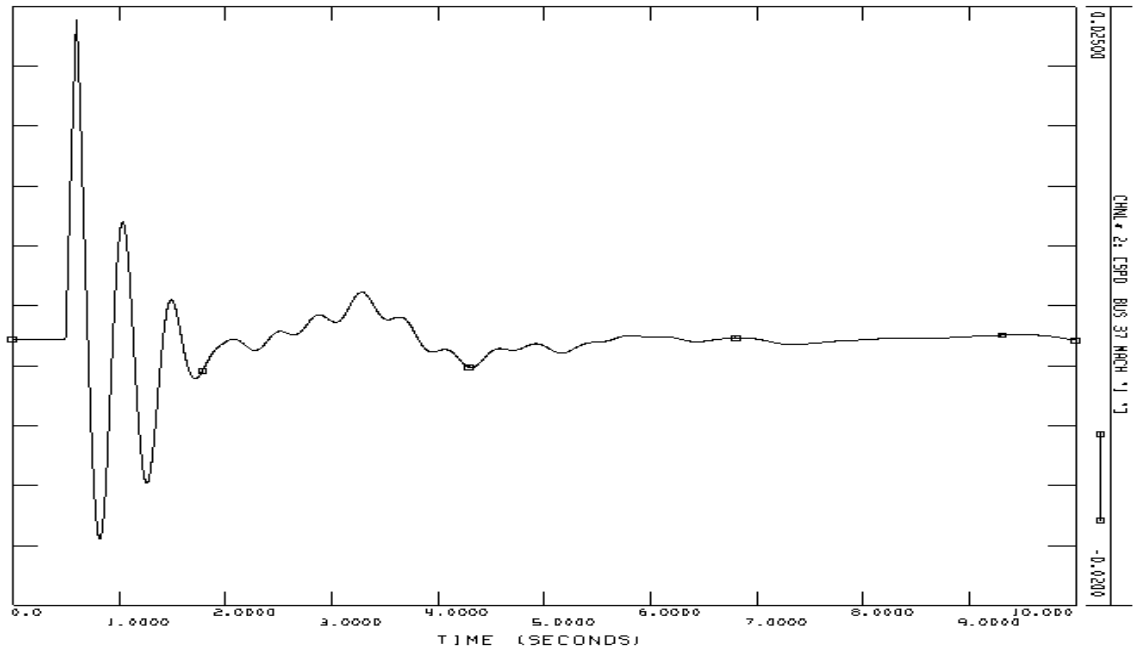
**Figure 4.13 Expanded System from New England 39 buses system (15×15)**

A fault on bus 17 of the base system is applied at 0.5s and removed at 0.6 seconds. The speed deviation of generator G8 in the base system is monitored. Results from a PSS/E simulation are shown in Figure 4.14.

The simulation results are summarized in Table 4.4, Table 4.5, Figure 4.15, and Figure 4.16, respectively. When the integration step is fixed, HH4 provides accurate simulation results for integration steps as large as 0.1seconds, causing the trapezoidal rule to fail. When the integration is variable, the maximum integration step for HH4 is much larger than the maximum integration step for the trapezoidal rule, and HH4 is significantly faster than the trapezoidal rule. HH4 provides this enhanced speed with no loss of precision, an observation made for the testing reported in Section 4.4.1.




**(a) Voltage of Bus 37**




**(b) Speed of Generator Connected to Bus 37**

**Figure 4.14 Simulation Results of 8775 Buses System with Fixed Integration Step in PSS/E**

**Table 4.4 Simulation Results of 8775 Buses System with Fixed Integration Step**

| Integration Methods | Integration Step (s) | Algebraic Solver | | Linear Solver Library | Simulation Time (s) | Correctness by Observation |
|---|---|---|---|---|---|---|
| | | Solver | Precision | | | |
| **Trapezoidal** | 0.001 | VDHN | $10^{-6}$ | SuperLU | 485.801 | (exact solution) |
| **Trapezoidal** | 0.01 | VDHN | $10^{-6}$ | SuperLU | 85.376 | Correct |
| **Trapezoidal** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 11.094 | Incorrect |
| **HH4** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 31.438 | Correct |
| **Simulation in PSS/E with integration step of 0.001** | | | | | around 92 s | |

**Table 4.5 Simulation Results with Variable Integration Step**

| Integration Methods | | | | | Algebraic Solver | | Linear Solver Library | Simulation Time (s) |
|---|---|---|---|---|---|---|---|---|
| **Methods** | Upper Error Bound | Lower Error Bound | Maximum Step (s) | Num. of Integration Steps | Alg. Solver | Precision | | |
| **Trap.** | $5\times10^{-4}$ | $10^{-4}$ | 0.046814 | 617 | VDHN | $10^{-6}$ | SuperLU | 78.86 |
| **Trap.** | $10^{-4}$ | $5\times10^{-5}$ | 0.032640 | 960 | VDHN | $10^{-6}$ | SuperLU | 122.782 |
| **Trap.** | $5\times10^{-5}$ | $10^{-5}$ | 0.024659 | 1301 | VDHN | $10^{-6}$ | SuperLU | 150.203 |
| **Trap.** | $10^{-5}$ | $5\times10^{-6}$ | 0.015703 | 1954 | VDHN | $10^{-6}$ | SuperLU | 228.952 |
| **Trap.** | $5\times10^{-6}$ | $10^{-6}$ | 0.010960 | 2717 | VDHN | $10^{-6}$ | SuperLU | 297.542 |
| **HH4** | $5\times10^{-4}$ | $10^{-4}$ | 0.280885 | 92 | VDHN | $10^{-6}$ | SuperLU | 41.663 |
| **HH4** | $10^{-4}$ | $5\times10^{-5}$ | 0.213538 | 112 | VDHN | $10^{-6}$ | SuperLU | 68.871 |
| **HH4** | $5\times10^{-5}$ | $10^{-5}$ | 0.177748 | 123 | VDHN | $10^{-6}$ | SuperLU | 65.903 |
| **HH4** | $10^{-5}$ | $5\times10^{-6}$ | 0.154095 | 155 | VDHN | $10^{-6}$ | SuperLU | 93.67 |
| **HH4** | $5\times10^{-6}$ | $10^{-6}$ | 0.133311 | 163 | VDHN | $10^{-6}$ | SuperLU | 74.639 |

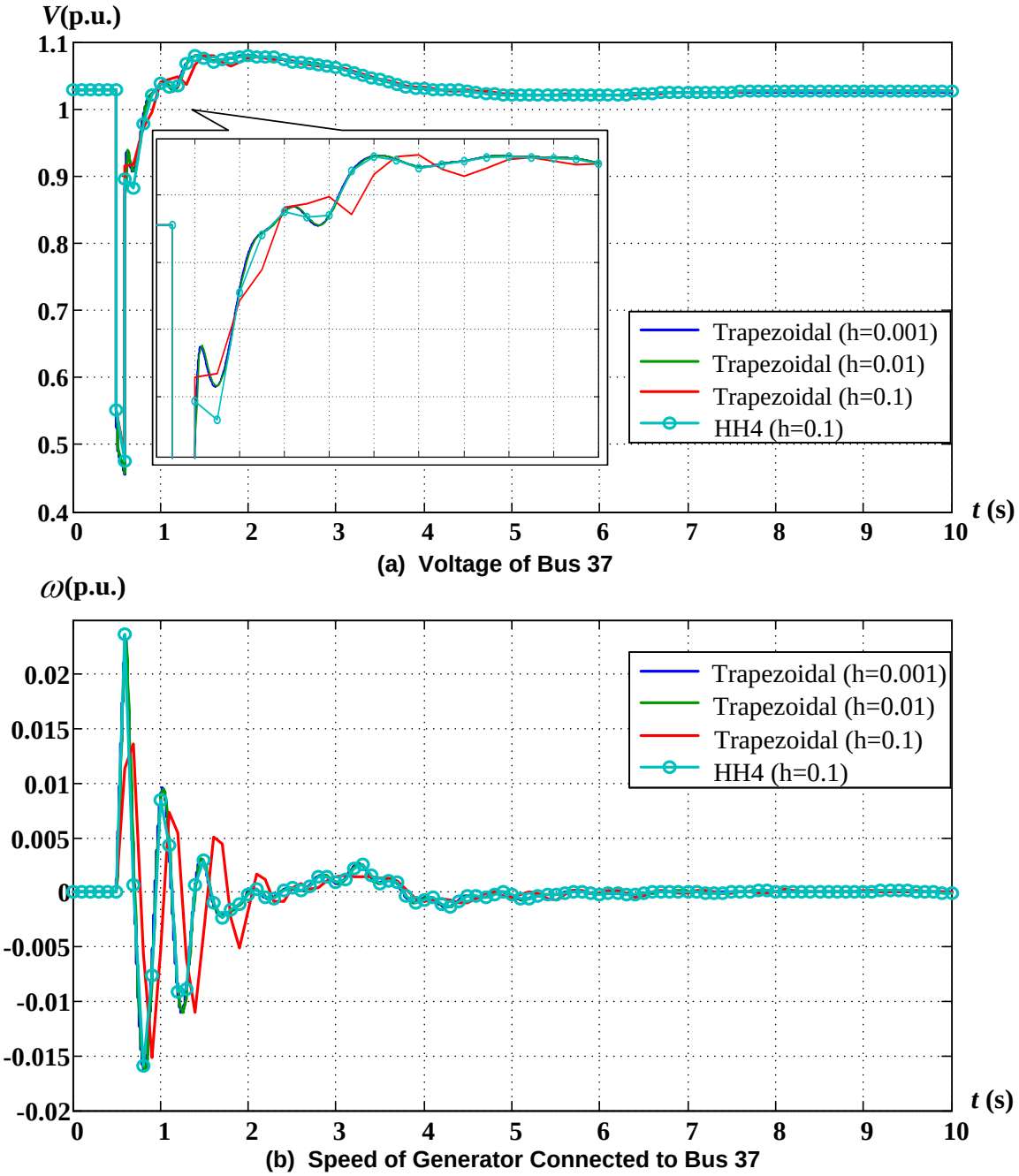



**Figure 4.15 Simulation Results of 8775 Buses System with fixed integration step in HSET-TDS**

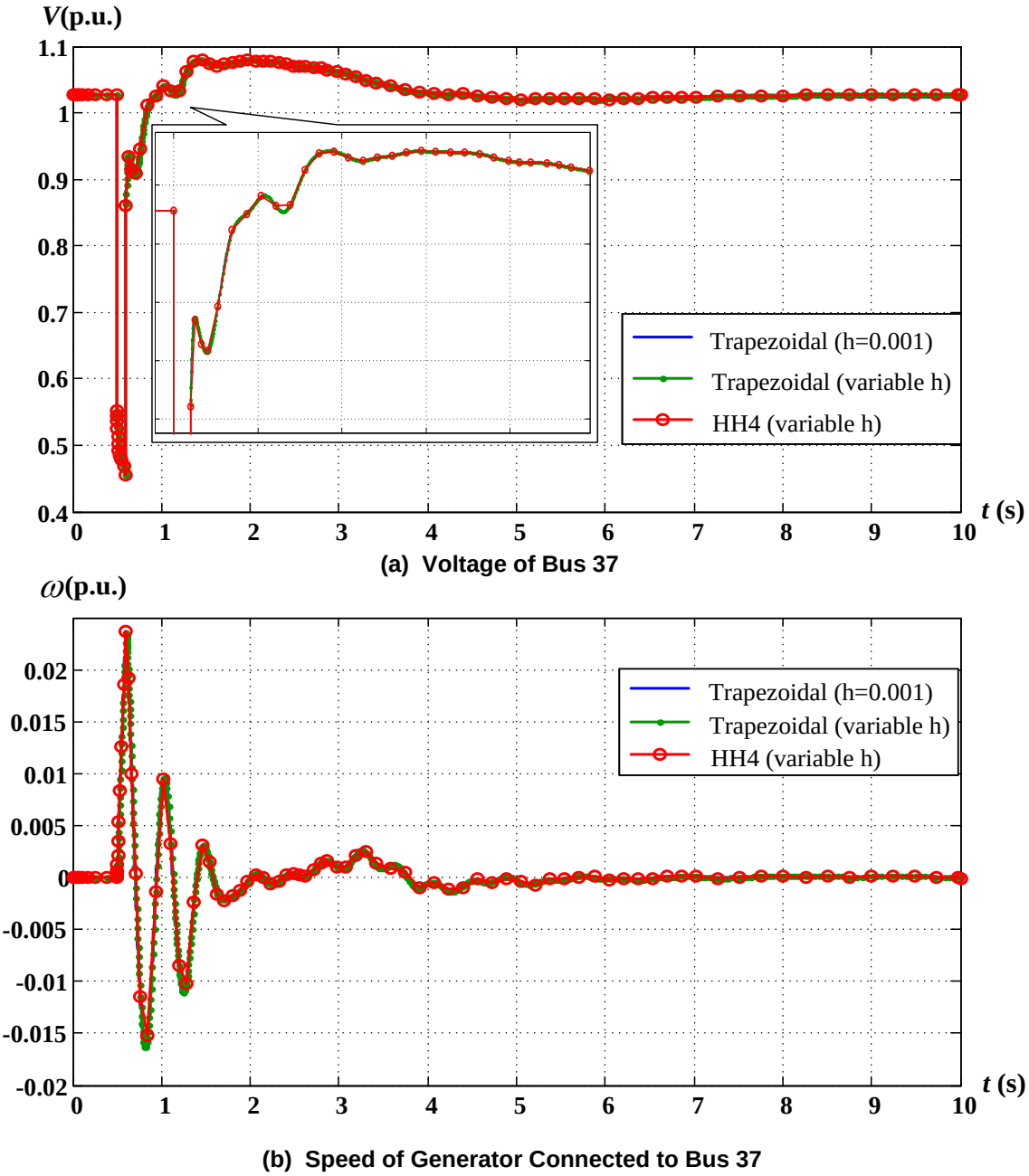



**Figure 4.16 Simulation Results of 8775 Buses System with variable integration step in HSET-TDS**

### 4.4.3 Modified PJM System with 13029 Buses and 431 Generator Buses

A model with 13029 buses, 431 generators, and 17,873 branches representing the PJM system was used to test HSET-TDS and to compare the integration methods of the trapezoidal rule and HH4. The dynamic models in the original dynamic file were replaced by the models of GENROU, IEEET1 and TGOV1, and the parameters of these models were adjusted correspondingly. A bus fault on bus 37 named "BRANCHBU" was applied at 0.5s and persisted until 0.6s. The generator connected to bus 26 named "BERGEN" was monitored. Simulation results from PSS/E are shown in Figure 4.17.

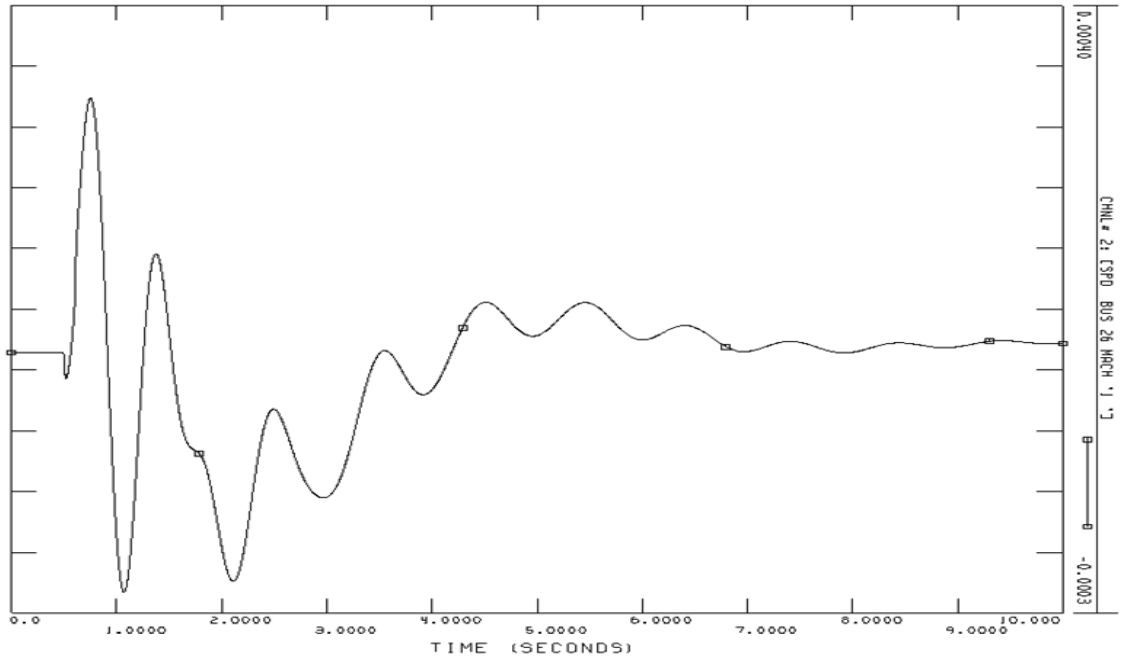



**Figure 4.17 Simulation Results of PJM 13029 Buses System with Fixed Integration Step in PSS/E**

Additionally, we simulated the modified PJM system in HSET-TDS by trapezoidal rule and HH4 methods, respectively, each with fixed and variable-step techniques. The

simulation results from HSET-TDS are shown in Table 4.6, Figure 4.18, Table 4.7, and Figure 4.19. It can be seen that HH4 is more efficient than the trapezoidal rule.

**Table 4.6 Simulation Results of PJM 13029 Buses System with Fixed Integration Step**

| Integration Methods | Integration Step (s) | Algebraic Solver | | Linear Solver Library | Simulation Time (s) | Correctness by Observation |
|---|---|---|---|---|---|---|
| | | Solver | Precision | | | |
| **Trapezoidal** | 0.001 | VDHN | $10^{-6}$ | SuperLU | 363.696 | (exact solution) |
| **Trapezoidal** | 0.01 | VDHN | $10^{-6}$ | SuperLU | 54.249 | Correct |
| **Trapezoidal** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 7.641 | Incorrect |
| **HH4** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 20.375 | Correct |
| **Simulation in PSS/E with integration step of 0.001** | | | | | around 62 s | |

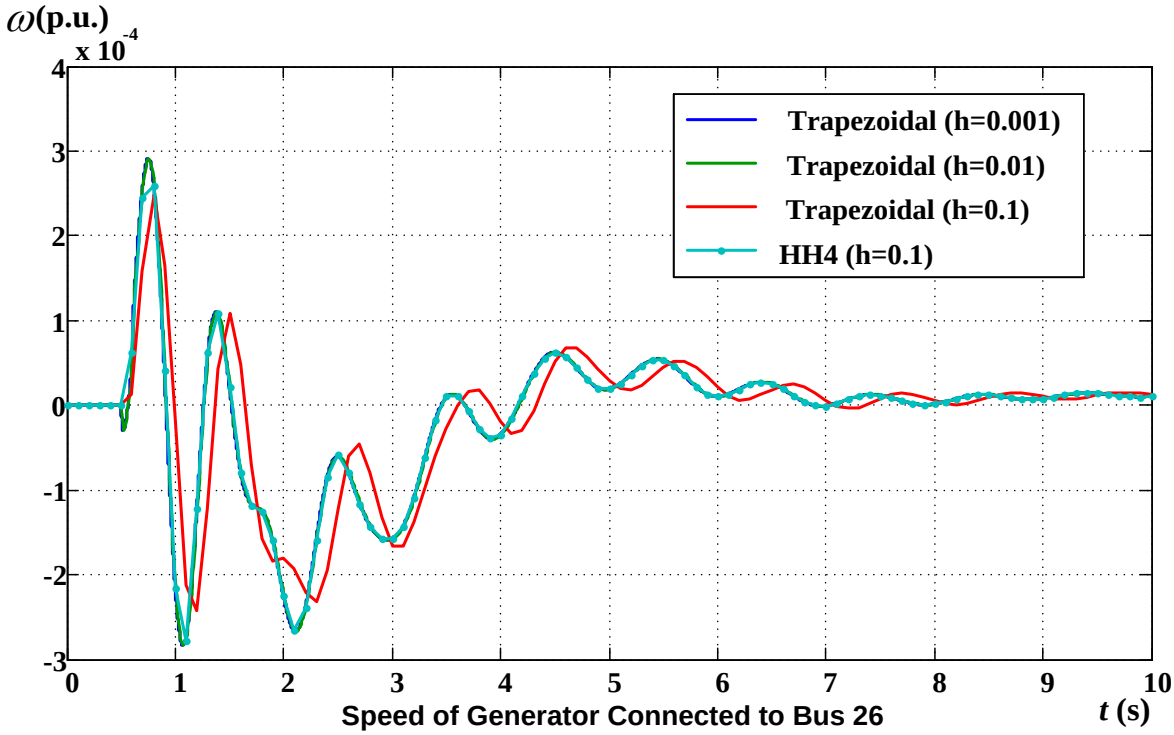



**Figure 4.18 Simulation Results of PJM 13029 Buses System with fixed integration step in HSET-TDS**

**Table 4.7 Simulation Results with Variable Integration Step**

| Integration Methods | | | | | Algebraic Solver | | Linear Solver Library | Simulation Time (s) |
|---|---|---|---|---|---|---|---|---|
| **Methods** | Upper Error Bound | Lower Error Bound | Maximum Step (s) | Num. of Integration Steps | Alg. Solver | Precision | | |
| **Trap.** | $5\times10^{-4}$ | $10^{-4}$ | 0.058452 | 510 | VDHN | $10^{-6}$ | SuperLU | 55.858 |
| **Trap.** | $10^{-4}$ | $5\times10^{-5}$ | 0.046033 | 687 | VDHN | $10^{-6}$ | SuperLU | 67.484 |
| **Trap.** | $5\times10^{-5}$ | $10^{-5}$ | 0.039319 | 817 | VDHN | $10^{-6}$ | SuperLU | 81.968 |
| **Trap.** | $10^{-5}$ | $5\times10^{-6}$ | 0.020983 | 1195 | VDHN | $10^{-6}$ | SuperLU | 101.25 |
| **Trap.** | $5\times10^{-6}$ | $10^{-6}$ | 0.013726 | 1530 | VDHN | $10^{-6}$ | SuperLU | 121.063 |
| **HH4** | $5\times10^{-4}$ | $10^{-4}$ | 0.394550 | 68 | VDHN | $10^{-6}$ | SuperLU | 22.437 |
| **HH4** | $10^{-4}$ | $5\times10^{-5}$ | 0.315996 | 79 | VDHN | $10^{-6}$ | SuperLU | 31.094 |
| **HH4** | $5\times10^{-5}$ | $10^{-5}$ | 0.249676 | 86 | VDHN | $10^{-6}$ | SuperLU | 27.937 |
| **HH4** | $10^{-5}$ | $5\times10^{-6}$ | 0.168772 | 103 | VDHN | $10^{-6}$ | SuperLU | 43.609 |
| **HH4** | $5\times10^{-6}$ | $10^{-6}$ | 0.187257 | 111 | VDHN | $10^{-6}$ | SuperLU | 37.297 |

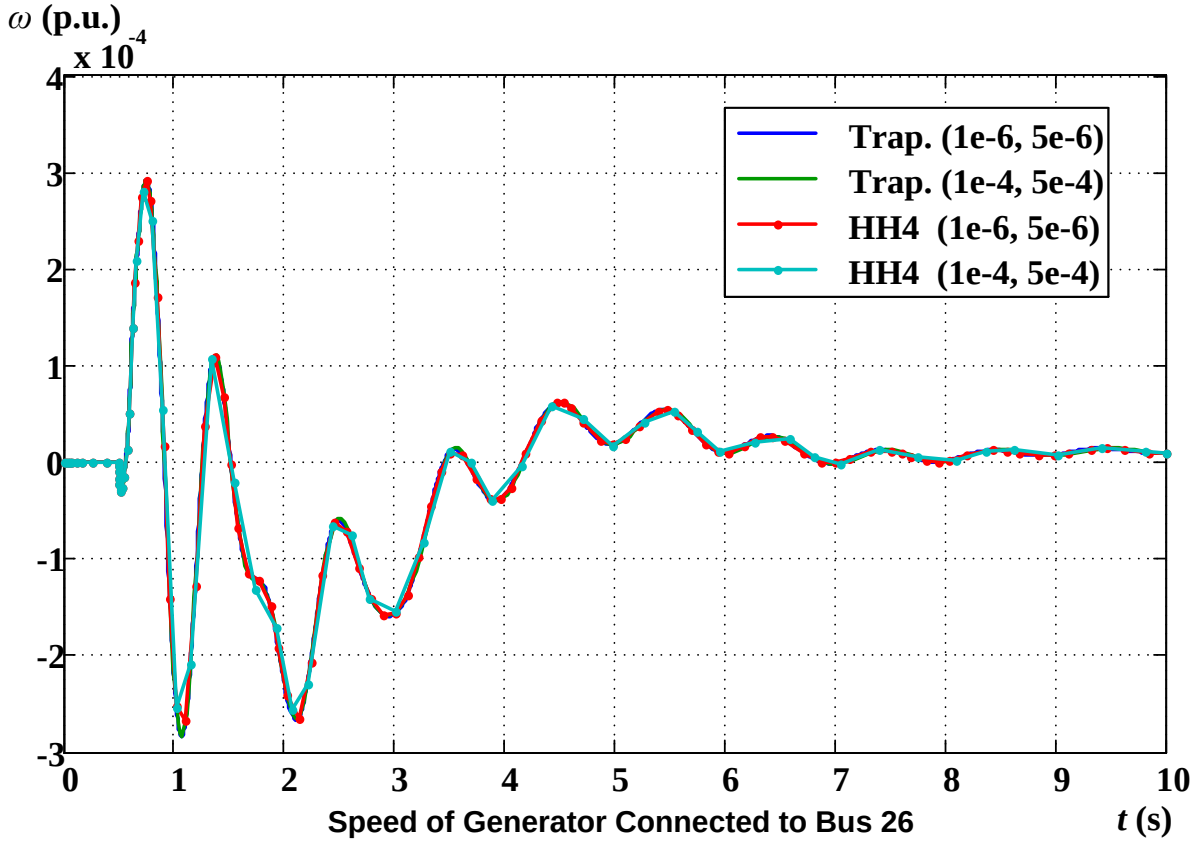



**Figure 4.19 Simulation Results of PJM 13029 Buses System with variable integration step in HSET-TDS**

### 4.4.4 Extended-Term Simulation of Cascading (1 hour) on a 39-Bus System

A case of cascading based on the 39-bus system shown in Figure 3.8 was simulated in HSET-TDS to further test the efficiency of HH4 while illustrating the type of application for which its strengths are most salient, i.e., long-term unfolding of cascading scenarios. As shown in Table 4.8, a sequence of fault/line outage events follows an initiating fault/line outage event, a scenario which could occur as lines load more heavily as a result of earlier outages. In addition, a corrective action (capacitor insertion) was included to prevent the system from becoming unstable. The voltage of bus 26 was monitored, and the simulation curve is shown in Figure 4.20. The responses without the corrective action of inserting shunt capacitors (events 7 and 10) are also shown in Fig. 21. Simulation times using variable-step technique are summarized in Table 4.9, and it can be seen that HH4 is more efficient.

**Table 4.8 Events list of Cascading on 39 Buses System**

| Event | Start Time(s) | Events List |
|---|---|---|
| 1 | 30.5 | Fault line 25-26 (end of 25) |
| 2 | 30.58 | Line 25-26 is outaged |
| 3 | 1000.0 | Fault line 26-29 (end of 26) |
| 4 | 1000.04 | Line 26-29 is outaged |
| 5 | 1010.0 | Fault line 17-18 (end of 17) |
| 6 | 1010.05 | Line 17-18 is outaged |
| 7 | 1020.0 | Shunt Capacitor is inserted at bus 26 |
| 8 | 2500.0 | Load on bus 26 is increased by 25% |
| 9 | 3000.0 | Load on bus 26 is increased by 25% |
| 10 | 3015.0 | Shunt Capacitor is inserted at bus 26 |

**Table 4.9 Simulation Results for 3600 sec. with Variable Integration Step**

| Integration Methods | Algebraic Solver | Linear Solver Library | Sim. Time (s) |
|---|---|---|---|
| **Trapezoidal** | VDHN | SuperLU | 19.437 |
| **HH4** | VDHN | SuperLU | 6.172 |
| **Simulation in PSS/E with integration step of 0.001** | | | around 123s |

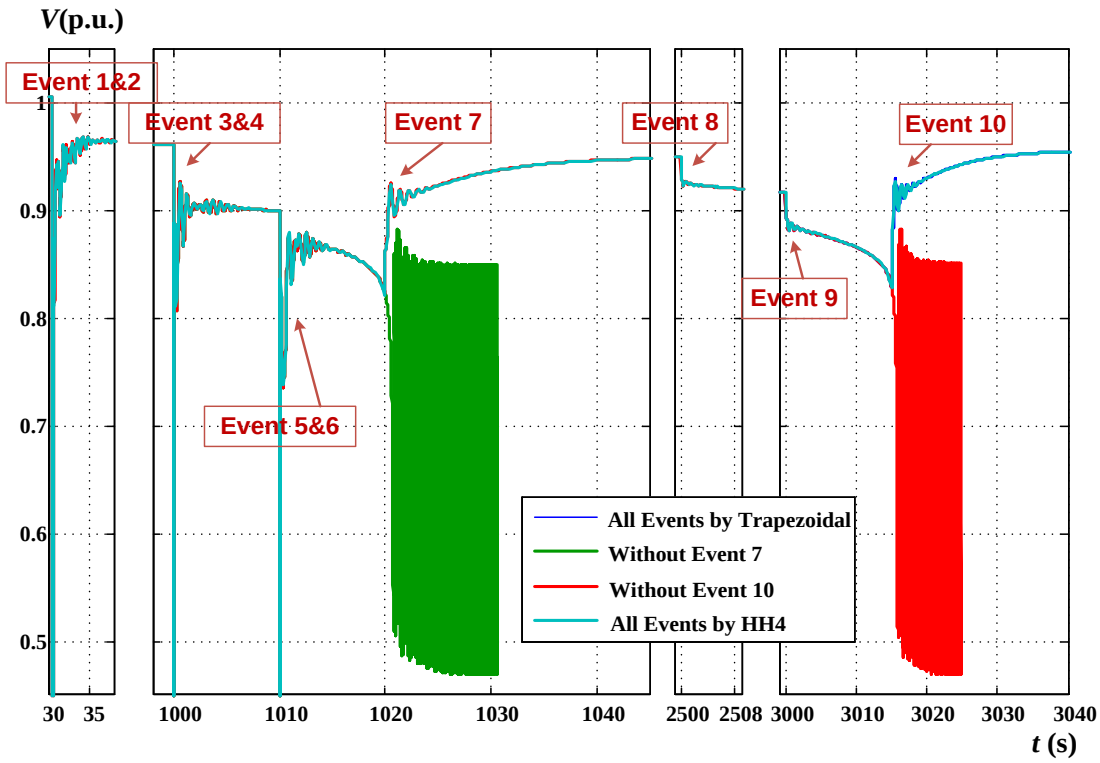



**Figure 4.20 Main Parts of Simulation Results of Cascading for 1 hour in HSET-TDS**

# CHAPTER 5 STIFFNESS DETECTION AND DECOUPLING METHOD

Step-by-step numerical integration is a basic way to solve differential algebraic equations (DAE). Usually numerical integration methods can be classified into two categories: explicit methods and implicit methods. In explicit methods, the next step calculation uses only the solution information known, and the computation of each iteration point is efficient. However, it is reported that the explicit method can encounter numerical stability difficulties when stiff problems are solved [6, 24, 25, 27]. Stiff equations are problems for which explicit methods don't work [27]. Large errors may occur when a fixed integration step technique is used or a too small integration step is used when avariable-step technique is used. In the implicit method, the calculation uses the unknown solution information of next-step(s), and non-linear equations must be solved at each step. Implicit methods are slow but stable. Implicit methods are commonly used for performing dynamic powersystem simulation. It is desirable to make use of the advantages of both explicit and implicit methods by integrating the two types..

In this chapter, a decoupling method will be introduced that can combine the explicit method and implicit methods. The main idea of this method is to decouple the differential portion of the DAE system into stiff and non-stiff parts. Explicit methods will be adopted to solve the non-stiff part, while implicit methods will be used to solve the stiff part. The critical technique for the decoupling method is detecting the stiffness in the DAE system and correctly partitioning the DAE into stiff and non-stiff parts. The following sections will

introduce the techniques, and in particular describe the practical technique used in HSET-TDS.

## 5.1 Automatic Stiffness Detection

Stiff equations are problems for which explicit methods don't work. There are two methods to detect the stiffness of ordinary differential equations[27].

### 5.1.1 Estimation of eigenvalues $\lambda_i$ the Jacobian matrix J

Assume that $R(z)$ is the ***stability function*** corresponding to an explicit method , where $z = h\lambda$ and $h$ is the step value. The ***stability domain*** of a method is the set $S = \{z \in C; |R(z) \le 1|\}$ . If the eigenvalue $\lambda_i$ satisfies the condition that $R(h\lambda_i) < 1$, then $h\lambda_i$ lies in the stability domain. The differential equations whose eigenvalues do not satisfy the condition of $R(h\lambda_i) < 1$ are stiff.

An example based on the test function $\dot{x} = \lambda x$ can demonstrate the stiffness phenomenon which is associated with eigenvalues and step size. The explicit method will be the Forward Euler method (FEM), whose stability function is $R_{FEM} = (1 + h\lambda)$ .We assume that step size $h = 0.001$ , and $\lambda$ is selected as $\lambda_1 = -100$ , $\lambda_2 = -1000 + 1000i$ , $\lambda_3 = -1500 + 1000i$ ,

When $\lambda_1 = -100$ , it can be shown that $|R(h\lambda_i)| < 1$ , and the problem is non-stiff. From the figure, it can be found that the simulation results are matched with the exact solution. When $\lambda_2 = -1000 + 1000i$ , it can be found that $|R(h\lambda_i)| = 1$ where the critical phenomenon appears. The simulation results show that there is much oscillation but the result is not totally divergent. When $\lambda_3 = -1500 + 1000i$ , it can be shown that $|R(h\lambda_i)| > 1$ , and

the problem is stiff. The simulation results in Figure 5.1 become divergent.

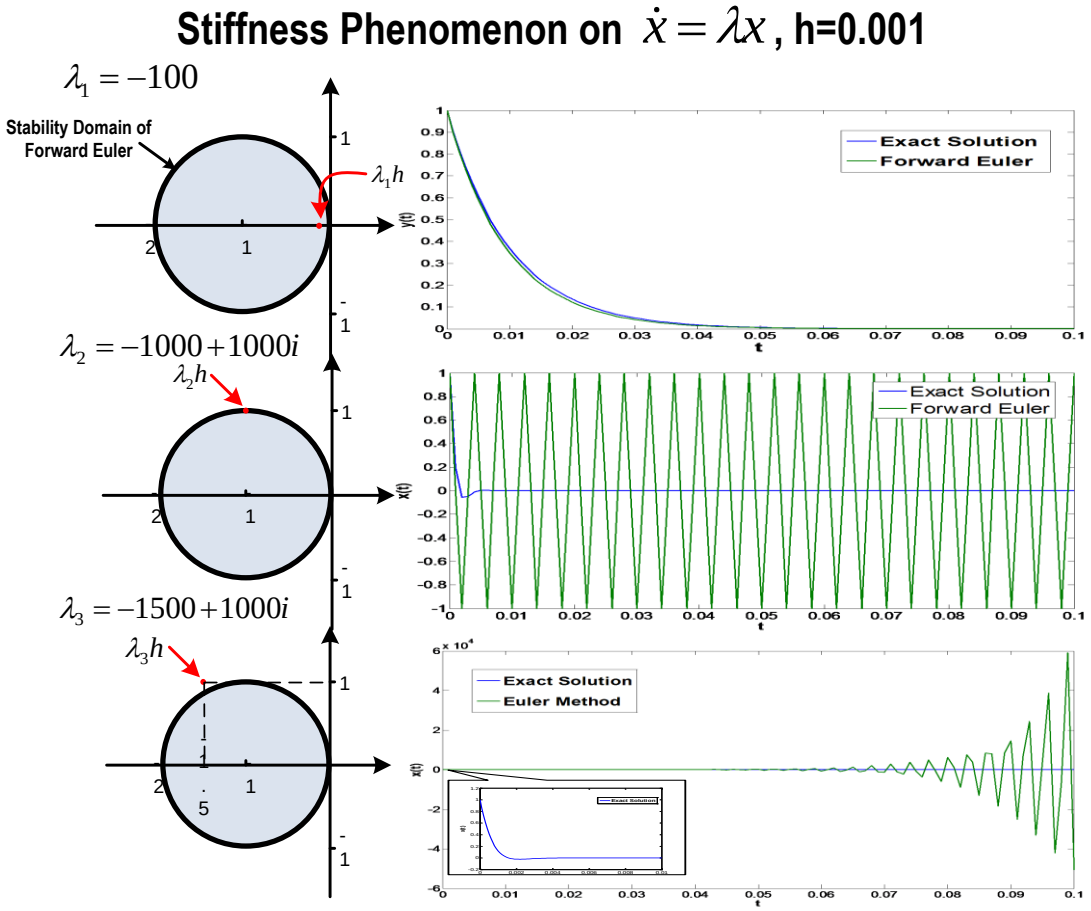



**Figure 5.1 Stiffness Phenomenon Associated with Eigenvalues**

Therefore, during the process of one-step integration, if all the eigenvalues of the Jabobian matrix from the DAE system can be acquired, stiffness can easily be determined from the stability function of an explicit method. The differential equations whose eigenvalues are outside the stability domain (for a given explicit method) will be differentiated by implicit methods, while explicit methods will be utilized to discretize differential equations whose eigenvalues are within the stability domain. This method is not efficient since we need to calculate the eigenvalues of the Jacobian matrix using numerical

iterative methods such as QR decomposition or the Arnoldi method, both of which are time-exhausting. Furthermore, since the eigenvalues are time-variable, the eigenvalues of the Jacobian matrix need to be updated after each integration step. This issue makes stiffness detection by calculation of eigenvalues excessively time-consuming.

### 5.1.2 Based on error estimation

The basic idea underlying this method is to check the variation of global-truncation error (GTE) of each step by explicit methods. The global-truncation error can be efficiently estimated by the Richardson extrapolation method. When $h\lambda_i$ lies within the stability domain, the global-truncation error is convergent and the values have decreasing tendency, while when $h\lambda_i$ lies outside the stability domain, global-truncation error is divergent and the values have increasing tendency.

*1) Illustration on a linear system*

We continue to consider the example linear system discussed in the previous section. For convenience, we skip the processes of Richardson extrapolation or error estimation techniques introduced in chapter 4 and use the exact solution to directly calculate global truncation error. The exact solution for the test function is $x(t) = e^{\lambda t}$ .The absolute values of GTE are shown in Figure 5.2.

From Figure 5.2, it can be found that, when the differential equation is nonstiff for the forward Euler method, the GTE is finally convergent. When the differential equation is in critical condition for the forward Euler method, the GTE becomes a constant value. When the differential equation is stiff, the GTE becomes divergent.

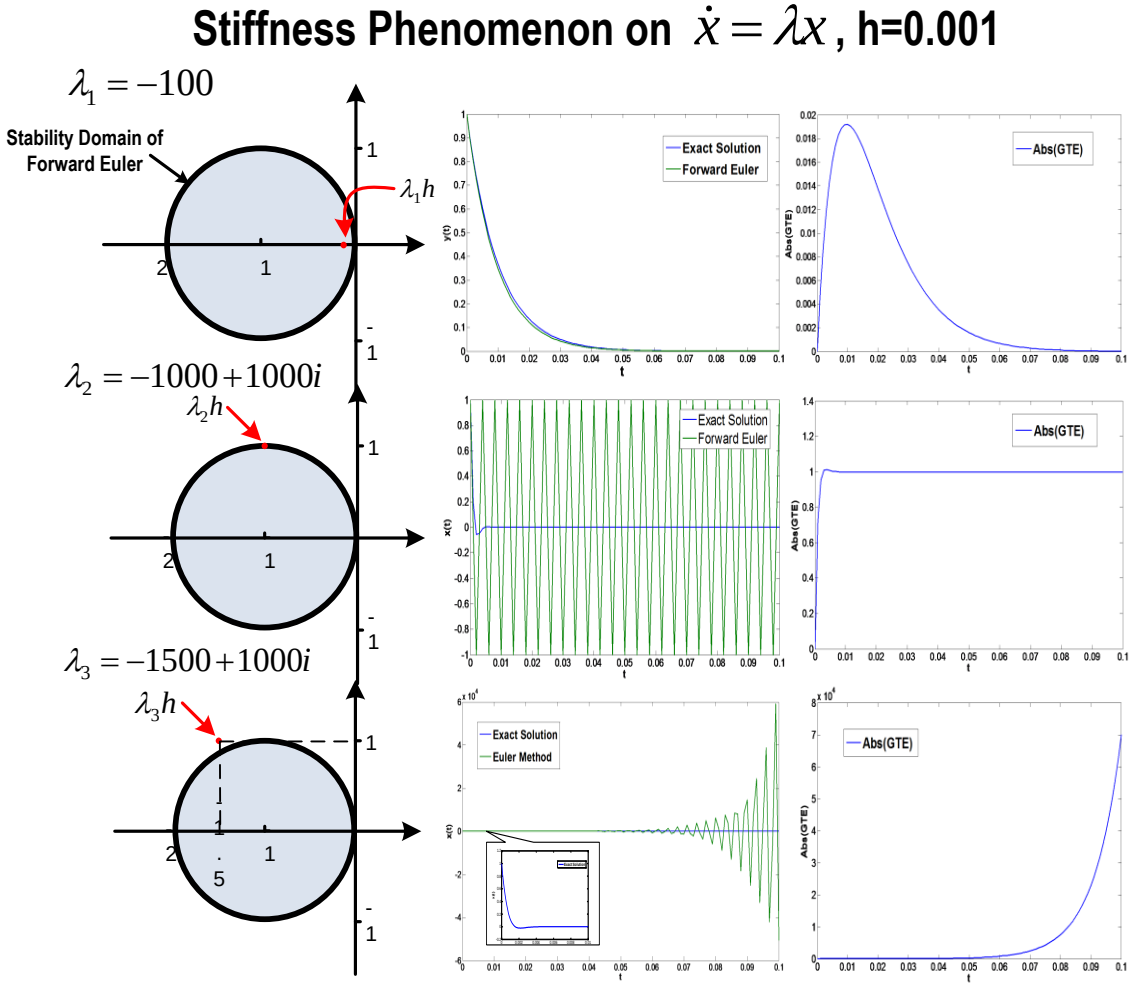



**Figure 5.2 Stiffness Phenomenon Associated with GTE on Test Function**

*2) Illustration on a nonlinear system*

The next example shows whether the automatic stiffness detection based on error estimation works for a nonlinear system when the eigenvalues will change from time to time and the system varies from stiff to non-stiff for the forward Euler method.

Analyze the ODE $\begin{cases} \dot{x} = x^2 \\ x(0) = x_0 \end{cases}$ The exact solution is $x(t) = -\frac{1}{t + 1/x_0}$ We assume that step size $h = 0.001$ . We still use the forward Euler method to calculate the system behavior. In order to make the eigenvalues of the ODE system variable starting from different points, we set the initial values for three cases: $x_0 = 400$, $x_0 = 1000$ and $x_0 = 1100$. The system was simulated for 0.2 second. We recorded the following curves, i) the trajectory of eigenvalues, ii) the simulation solution by the forward Euler method with step size *h* and *h*/2, and iii) the global truncation error by exact solution and extrapolation. The simulation results are shown in Figure 5.3 and the analysis is described as follows.

When $x_0 = 400$ , *z* starts from the inside of the stability domain. It can be shown that both simulation results are able to match the exact solution. Also, the GTE from both the exact solution and extrapolation becomes smaller and smaller following the change of *z*.

When $x_0 = 1000$ , *z* starts from the boundary of the stability domain. We found that the simulation result with *h*/2 is able to match the exact solution while the simulation result with *h* becomes incorrect and not totally divergent. Also, the GTE from both the exact solution and extrapolation becomes smaller and smaller following the change of *z*, but the error at the beginning is larger than that when $x_0 = 400$.

When $x_0 = 1100$ , *z* starts from the outside of the stability domain. It can be shown that both simulation results cannot match with the exact solution. Also, the GTE from both the exact solution and extrapolation becomes larger and larger following the change of *z*.

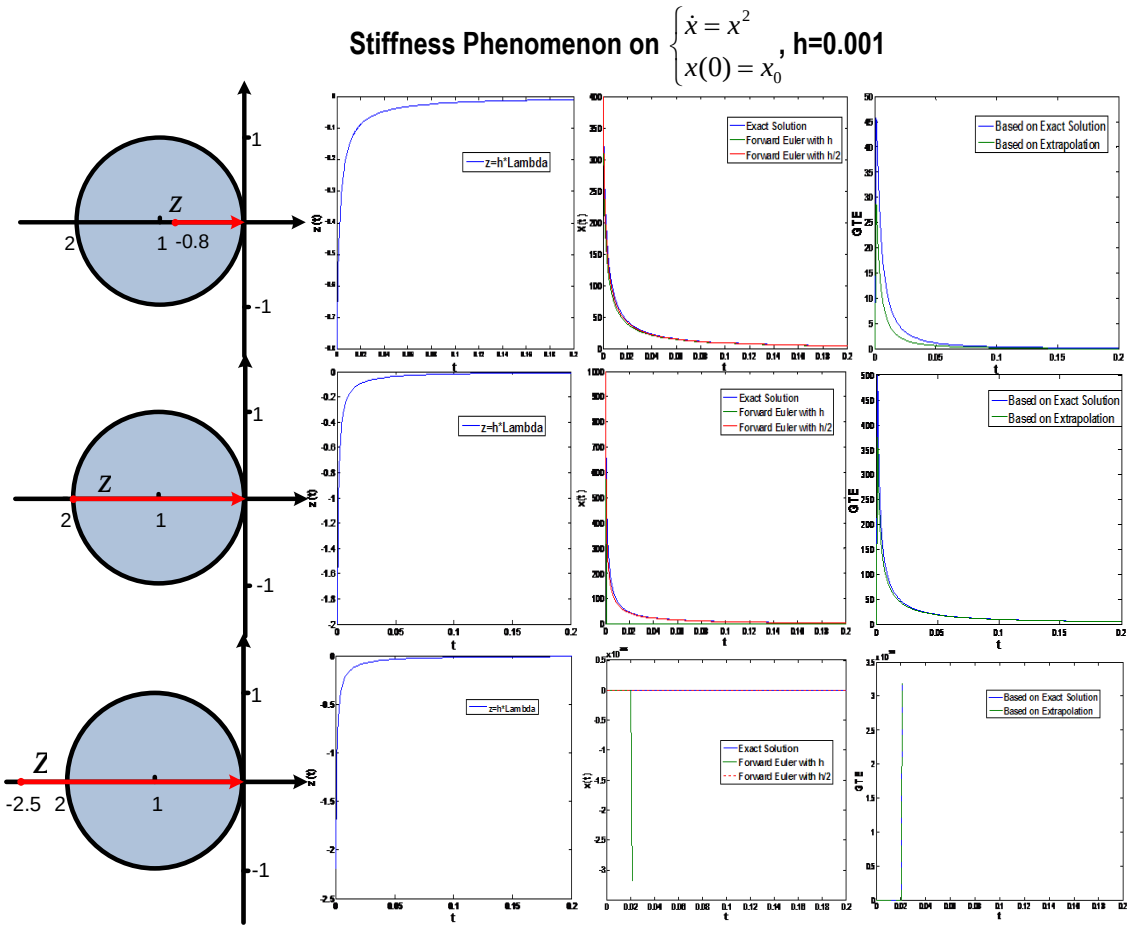



**Figure 5.3 Stiffness Phenomenon Associated with GTE on a Non-linear System**

The automatic stiffness detection based on error estimation can efficiently detect the stiffness of an ODE system. The computational cost is just basic global-error estimation. The potential problem associated with this stiffness detection method is detecting the increasing or decreasing tendency of the GTE. It's possible that the result of detection is conservative because there are probably many cycles of oscillation until the GTE decreases. There are two approaches for dealing with this problem.

Reference [27] introduces one idea, which is to check the error successively (say 15 times). Then final conclusion as to stiffness or non-stiffness can then be drawn.

Another idea is to give a threshold for global-truncation error. If the GTE is over the threshold, the corresponding equation is considered to be stiff. The stiffness threshold for an explicit method can be fixed before the simulation. It's a good idea to set the threshold with reference to the critical case. Also, considering that the absolute GTE is unfair for each equation of the ODE system, both relative and absolute GTEs can be used to check for stiffness.

## 5.2 Stiffness Decoupling Method

### 5.2.1 Recursive Projection Method

The main idea behind achieving computational gain is to take advantage of both the explicit and the implicit methods and simultaneously achieve as much parallelism as possible. This is made possible through the division of the ODE part of the DAE into stiff and non-stiff parts through a partition algorithm called the recursive projection method (RPM) [41, 42], an invariant subspace method. Consider the ODE system described in (5.1).

$$\begin{cases} \dot{x} = \mathbf{f}(x) \\ x(0) = x_0 \end{cases} \tag{5.1}$$

where $\boldsymbol{x}$ is an $n$-dimensional vector, and $\mathbf{f}(\boldsymbol{x})$ is an $n$-dimensional vector function described in expression (5.2)

$$\begin{cases} \dot{\boldsymbol{x}}_s = \boldsymbol{f}_s(\boldsymbol{x}_s, \boldsymbol{x}_{ns}) \\ \dot{\boldsymbol{x}}_{ns} = \boldsymbol{f}_{ns}(\boldsymbol{x}_s, \boldsymbol{x}_{ns}) \end{cases} \tag{5.2}$$

where $\boldsymbol{x}_s$ and $\boldsymbol{x}_{ns}$ are stiff and non-stiff variables, and $\boldsymbol{f}_s$ and $\boldsymbol{f}_{ns}$ are stiff and non-stiff equations. For the non-stiff part of ODEs, explicit methods can be used to efficiently compute the next-point values with numerical stability guaranteed, while implicit methods deal with the stability problems for the stiff parts using iterative computing.

The solution space $\mathbf{R}^n$ of ODEs (2) can be written as a direct sum of the span of the stiff eigenspace (say, invariant subspace **P**) and its orthogonal complement (say, invariant subspace **Q),** which is the non-stiff eigenspace. The original *n*-dimensional space can be split into two subsystems:

$$\mathbf{f}^P(\boldsymbol{p},\boldsymbol{q}) = \boldsymbol{Z}_1^T\mathbf{f}(\boldsymbol{Z}_1\boldsymbol{p}+\boldsymbol{Z}_2\boldsymbol{q}) \tag{5.3}$$

$$\mathbf{f}^Q(\boldsymbol{p},\boldsymbol{q}) = \boldsymbol{Z}_2^T\mathbf{f}(\boldsymbol{Z}_1\boldsymbol{p}+\boldsymbol{Z}_2\boldsymbol{q}) \tag{5.4}$$

where $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ are bases of invariant subspace **P** and **Q**, respectively, **Q** is the orthogonal complement of **P** such that $\mathbf{Q} = \mathbf{P}^{\perp}$, and $\boldsymbol{p} = \boldsymbol{Z}_1^T\boldsymbol{x}$, $\boldsymbol{q} = \boldsymbol{Z}_2^T\boldsymbol{x}$. Thus, the ODEs system equations can be decoupled into two subsystems:

$$\begin{cases} \dot{\boldsymbol{p}} = \mathbf{f}^P(\boldsymbol{p},\boldsymbol{q}) = \boldsymbol{Z}_1^T\mathbf{f}(\boldsymbol{Z}_1\boldsymbol{p}+\boldsymbol{Z}_2\boldsymbol{q}) \\ \dot{\boldsymbol{q}} = \mathbf{f}^Q(\boldsymbol{p},\boldsymbol{q}) = \boldsymbol{Z}_2^T\mathbf{f}(\boldsymbol{Z}_1\boldsymbol{p}+\boldsymbol{Z}_2\boldsymbol{q}) \end{cases} \tag{5.5}$$

By solving the above decoupled equations, $\boldsymbol{p}$ and $\boldsymbol{q}$ can be calculated separately, and the original states can be obtained from $\boldsymbol{x} = \boldsymbol{Z}_1\boldsymbol{p}+\boldsymbol{Z}_2\boldsymbol{q}$.

The critical problem of the recursive projection method is how to detect stiffness, and and how to fix the bases $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ of invariant subspace **P** and **Q.** As the Jacobian matrix of the original ODE systems is updated, the bases of invariant subspace P and **Q** change correspondingly, requiring that the bases $\boldsymbol{Z_1}$and $\boldsymbol{Z_2}$ need to be updated as well. There are several reported methods for estimating eigenvalues to detect stiffness. Reference [41] describes a method which approximates some vectors of the Jacobian matrix and uses QR decomposition to detect the stiffness and fix the basis. Reference [42] utilizes the Cayley transform of the Jacobian matrix and an iterative process to construct the basis $\boldsymbol{Z_1}$. The Arnoldi method is adopted in [26] to identify eigenvalues outside the stability domain of an

explicit method. Stiffness detection based on eigenvalue solution of the Jacobian matrix is precise but computationally time-consuming. The other two methods, QR decomposition and Cayley, also involve obtaining eigenvalues, but the strategy based on QR decomposition may be more efficient since just one step of QR decomposition is implemented in a one step stiffness detection.

### 5.2.2 Stiffness Decoupling Method used in HSET-TDS

The recursive projection method introduced in [41] is able to roughly detect the stiffness in the differential part of a DAE system. The computational cost of the method is the one-step QR decomposition of Jacobian matrix. There are two main disadvantages in the method: 1) one step QR decomposition can be much time-exhausting for large DAE system, and 2) the roughness of stiffness detection may lead to failure of one-step integration.

In HSET-TDS, stiffness decoupling based on error estimation is utilized. The process technique is illustrated in Figure 5.4. Since the stiffness detection needs the error estimation, the stiffness decoupling defaults to use variable-step techniques, which will change the integration step and stiffness basis during the process of each integration step. Figure 5.4 shows an example of the explicit theta method and the implicit theta method for non-stiff and stiff parts, and the forward Euler method for error estimation. The main process of the stiffness-decoupling method associated with explicit-theta and implicit-theta methods can be described as follows:

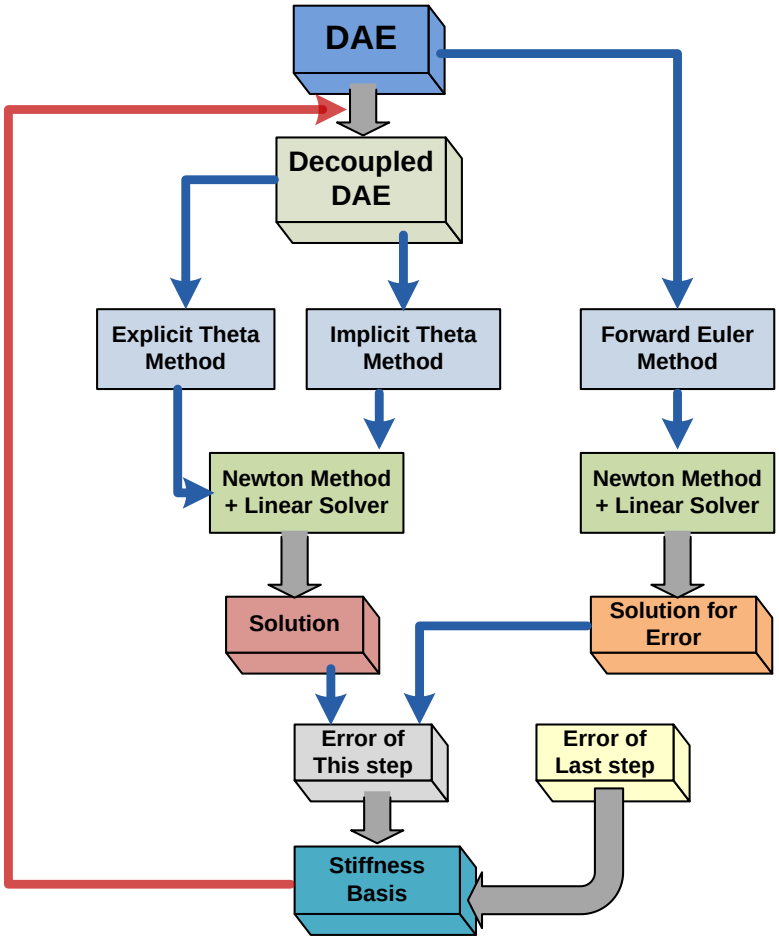



**Figure 5.4 Stiffness Decoupling Technique Adopted in HSET-TDS**

1) The stiffness basis is set to zero at the beginning.
2) The differential part of the original DAE system is separated into a stiff part and a non-stiff part according to the stiffness basis.
3) The explicit-theta method is utilized to differentiate the non-stiff part, and the implicit-theta method is adopted to differentiate the stiff part. Therefore, the differential part of the DAE system is transformed into a set of algebraic equations.
4) The Newton method is used to solve the algebraic equations from the differential part and the algebraic part of the DAE system, and then the solution using the stiffness-decoupling method can be acquired.

5) The forward Euler method is utilized to differentiate the differential part of the original DAE system, and the solution by forward Euler can be acquired after the Newton method.
6) The error of this step can be obtained by error-estimation formula specifically set up for the theta method, and the error must be saved for the next-step calculation.
7) The stiffness basis can be updated with the error of the current step and that of the last step. For one differential equation in the DAE system, it will be updated to be stiff if the equation is non-stiff according to the stiffness basis of the last step and the current error of the variable from this equation is larger than error from the previous step. Similarly, it will be updated to be non-stiff if the equation is stiff according to the stiffness basis of the last step and the current error is less than the previous error or within a certain interval.
8) Continue the variable-step technique.

During the process discussed above, two important issues must be clarified.

1) The error estimation technique for the theta method by forward Euler is feasible, since the estimation part will the part of $h^2$. This does not work if theta is zero since the theta method then becomes explicit Euler, and the error will always be zero. If the theta is 0.5, $x_{n+1}^{[h]}$ by theta method can be formularized as in (5.6) where $x(t_{n+1})$ is the real value of next step. $\bar{x}_{n+1}^{[h]}$ by Forward Euler method can be formularized as in (5.7). The error part is shown in (5.8). Since the estimation will maximize the real error, the estimated error will be multiplied by $h$ and a coefficient.

$$x_{n+1}^{[h]} = x(t_{n+1}) + \alpha(x_{n+1})h^3 + O(h^4) \qquad \textbf{(5.6)}$$

$$\bar{x}_{n+1}^{[h]} = x(t_{n+1}) + \beta(x_{n+1})h^2 + O(h^3) \quad \textbf{(5.7)}$$

$$\bar{x}_{n+1}^{[h]} - x_{n+1}^{[h]} = \beta(x_{n+1})h^2 + O(h^3) \quad \textbf{(5.8)}$$

2) The criterion for updating the stiff part of equations to become the non-stiff part is different from that for updating the non-stiff part to become the stiff part. Assume that a differential equation is non-stiff from the last-step stiffness list, which means the error of the previous step is calculated based on the explicit-theta method. According to the automatic stiffness detection procedure described in 5.1.2, the eigenvalue corresponding to the differential variable of this equation will be outside the stability domain if the error of the current step is larger than the error of the previous step. If a differential equation is stiff from the previous step stiffness list, which means that the error of the previous step was calculated based on the implicit theta method, the eigenvalue corresponding to the differential variable of this equation will still be outside the stability domain if the error of the current step is smaller than the error of the previous step. For example, when theta is 0.5, the stability function of the implicit theta method (Trapezoidal rule) can be expressed as (5.9), and the relationship between the error and the stability function can be described as (5.10).

$$R(z) = \frac{2+z}{2-z} \quad \textbf{(5.9)}$$

$$\frac{\delta_{n+1}}{\delta_n} = R(z) \quad \textbf{(5.10)}$$

If we want $z$ to lie within the stability domain of the explicit method (such as Forward Euler), the interval for $z$ is $|1+z|<1$. From (5.10), we can obtain the interval $[\frac{1}{3},1]$ for $\frac{\delta_{n+1}}{\delta_n}$, where $\delta_n$ and $\delta_{n+1}$ are the errors by the implicit theta method from the current step and the previous step, respectively.

The stiffness-decoupling method combined with variable-step technique can be more efficient than both the variable-step implicit method and variable-step explicit method. The computational cost is just the updating stiffness basis in each integration step by utilizing the error of this step and last step. Because of double-use of error for stiffness detection and integration-step resetting, the stiffnes- decoupling method is of demonstrable practical significance.

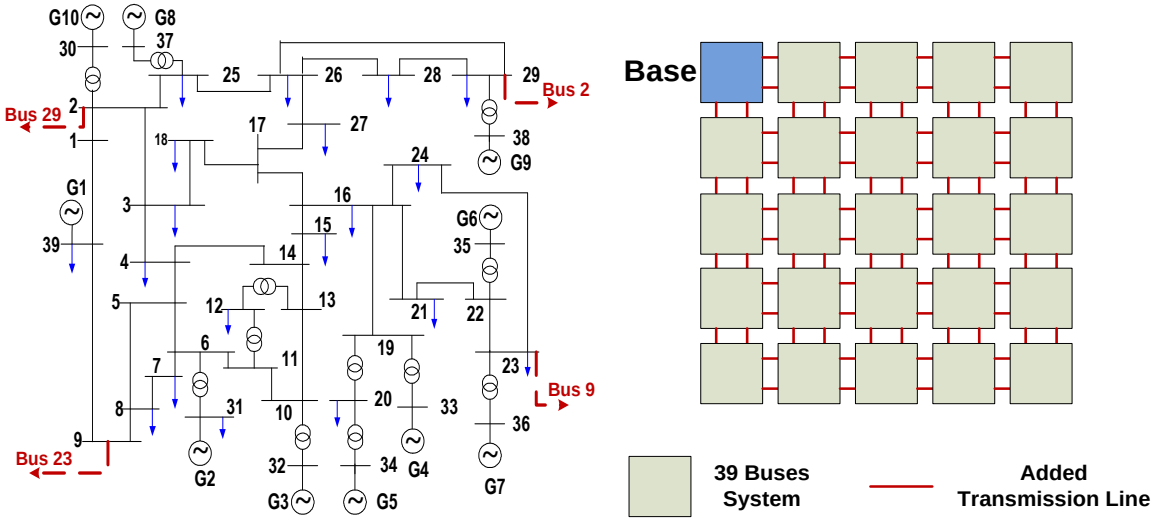
## 5.3 Simulation Results

The expanded 975-bus system expanded by 25 times from the New England 39-bus system is adopted as a test system. The construction of this system is similar to the 3900-bus system introduced in section 4.4.2. Buses 2, 9, 23, 29 are reconnected, and the construction follows the mesh structure shown in Figure 5.5.

The event is selected to be a bus fault on bus 17 starting from 0.5s and lasting for 0.1s. The voltage of bus 37 and the speed of G8 will be monitored. The complete simulation lasts for 10 seconds.

Three integration methods will be compared: the stiffness-decoupling method based on implicit-theta and explicit-theta methods, the variable-step implicit-theta method, and the variable-step explicit-theta method. The theta is set to be 0.5, and the forward Euler method

will be used for error estimation. The upper and lower boundaries for the variable-step technique are 0.001 and 0.0005, respectively. The nonlinear solver for these three methods is the Newton-based method, and the linear solver is SuperLU. The simulation results are shown in Table 5.1 and Figure 5.6. It can be seen that the stiffness decoupling method is relatively more efficient than other two methods. The explicit method takes more time, since its stability domain is limited and it requires smaller integration steps to guarantee precision.

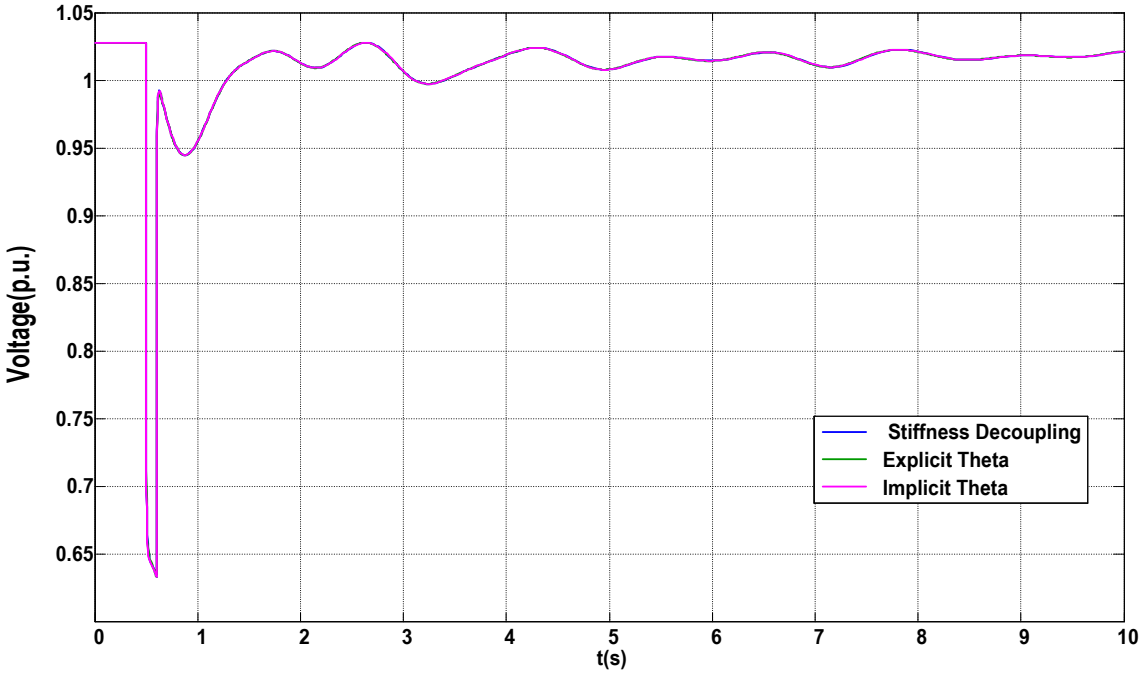



(a) Original 39 Buses System (b) Expanding Mesh

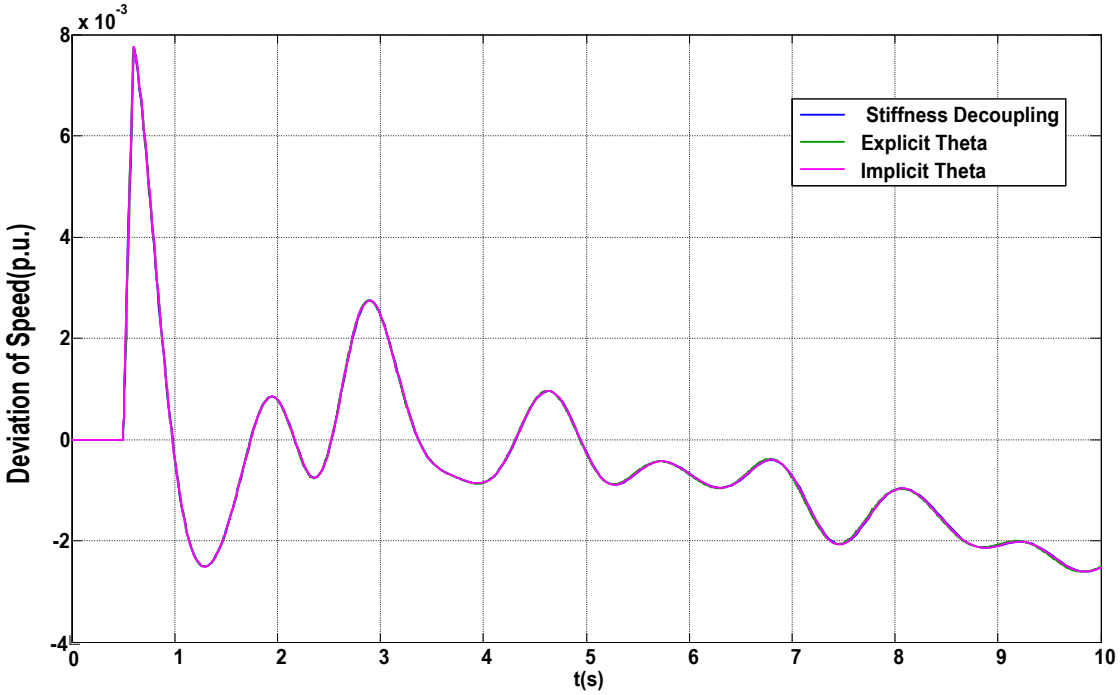
**Figure 5.5 Expanded System from New England 39 Buses System (5×5)**

**Table 5.1 Simulation Results of Stiffness Decoupling, Explicit Theta and Implicit Theta Methods**

| Integration Methods | Maximum Step (s) | Computation Times | Algebraic Solver | Linear Solver Library | Simulation Time (s) |
|---|---|---|---|---|---|
| **Stiffness Decoupling** | 0.050096 | 327 | Newton | SuperLU | 6.332 |
| **Explicit Theta** | 0.006050 | 1674 | Newton | SuperLU | 18.231 |
| **Implicit Theta** | 0.0417 | 405 | Newton | SuperLU | 7.574 |




(a ) Voltage of Bus 37




(b ) Speed Deviation of Generator 8

**Figure 5.6 Simulation Results of Expanded 975 Buses System**

# CHAPTER 6 SEQUENTIAL AND PARALLEL LIBRARY OF SUPERLU SOLVER

## 6.1 Introduction

The process of solving linear algebraic equations is important in time-domain simulation when the Newton-Raphson method is used. In Figure 2.1, it can be seen that the solution of solving linear equations is the bottom stage of all computational processs, and it usually represents the greatest computational load in time-domain simulation. Since coefficient matrices describing a DAE system modeling a power system and all Jacobian matrices of such a DAE system and associated nonlinear equations are highly sparse, it is absolutely necessary to adopt a sparse solver of linear equations to improve the efficiency of the time-domain simulation. In this section, SuplerLU [19, 43, 44, 45, 46, 47], an open source library developed by the Computer Science Division, University of California, Berkeley, CA, is introduced. This library is one of the sparse solvers included in HSET-TDS.

The SuperLU library contains three sub-libraries for both sequential and parallel computing, the status of which is summarized in Table 6.1 from [43]. The sub-library of sequential SuperLU is designed for sequential processors; multithreaded SuperLU (SuperLU_MT) is provided for shared memory parallel processors with Pthreads or OpenMP shared memory routines ; distributed SuperLU(SuperLU_DIST) is a parallel linear solver for distributed-memory parallel processors with MPI for interprocess communication. In the sequential-computing version of HSET-TDS, the sequential SuperLU library is included, while the distributed SuperLU library of has been adapted to further improve the efficiency of time-domain simulation in the parallel computing version of HSET-TDS.

**Table 6.1 Status of SuperLU library**

| | Sequential SuperLU | SuperLU_MT | SuperLU_DIST |
|---|---|---|---|
| **Platform** | Serial | Shared-memory | Districuted-memory |
| **Language** | C | C + OpenMP (or Pthreads) | C + MPI |
| **Data Type** | real / complex single / double | real / complex single / double | real / complex double |

[43] introduces the overall algorithm of sparse Gaussian elimination adopted in SuperLU, which can be described by the following two steps.

a) Compute a triangular factorization $P_r D_r A D_c P_c = LU$ , where $D_r$ and $D_c$ are diagonal matrices to equilibrate the system, and $P_r$ and $P_c$ are permutation matrices to reorder the rows and columns of $A$ . $L$ is a unit lower triangular matrix ( $L_{ii} = 1$) and $U$ is an upper triangular matrix.

b) Solve $AX = B$ by evaluating

$$X = A^{-1}B = (D_r^{-1}P_r^{-1}LUP_c^{-1}D_c^{-1})^{-1}B = D_c(P_c(U^{-1}(L^{-1}(P_r(D_rB)))))$$

SuperLU supplies several different routines to develop the solution, and HSET-TDS adopts some routines which are appropriate for time-domain simulation of power systems.

## 6.2 Sequential SuperLU

The computational routines of sequential SuperLU are described in Figure 6.1. SuperLU supplies two main driver routines with which to call the SuperLU library: i) a simple driver dgssv(), and ii) an expert driver dgssvx(). Both of these driver routines implement the following operations (as shown on the right-hand-side of Figure 6.1): factorization, triangular solving, estimating condition number, equilibrating, and refining solution. In the expert-driver routine, more options are provided to control SuperLU in terms

of solving the linear equations. One useful option is the configuration of methods to factorize the matrix A, which includes i) normal factorization, ii) factorization with identical pattern, iii) factorization with identical pattern and identical row, and iv) no factorization.

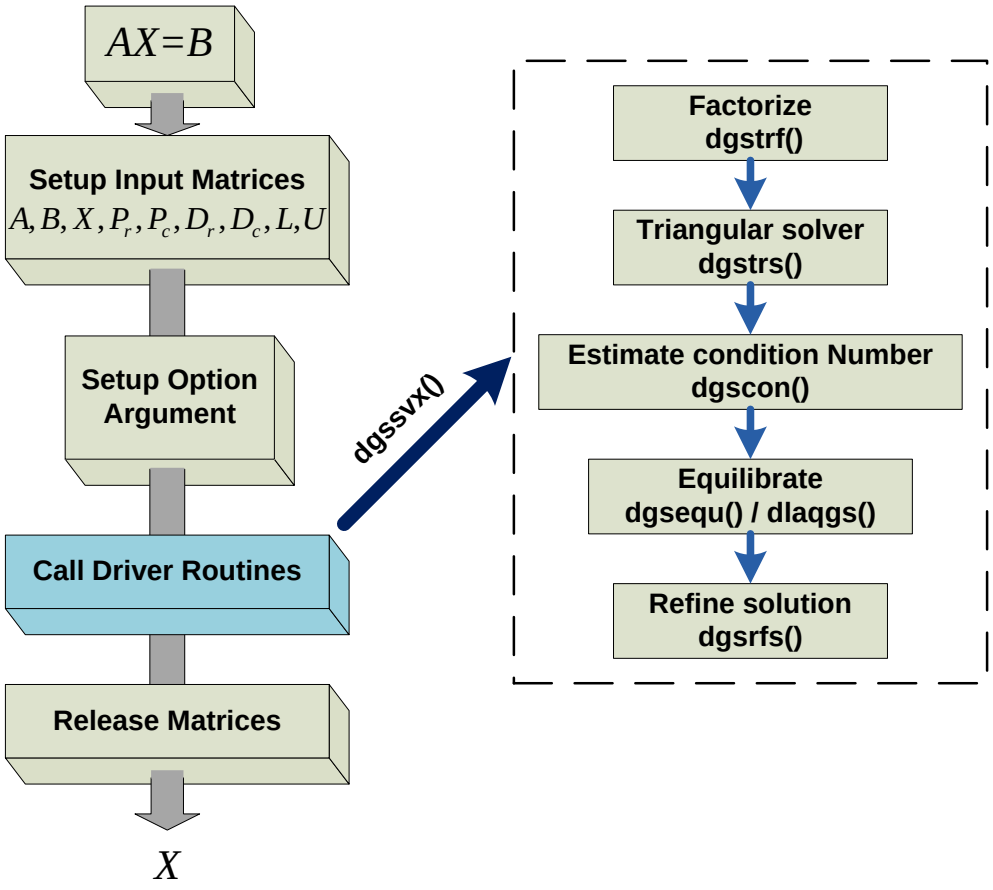



**Figure 6.1 Computational Routines of Sequential SuperLU**

Option ii, factorization with identical pattern, allows SuperLU to repeatedly use the diagonal matrices ($D_r$ and $D_c$) and permutation matrices ($P_r$ and $P_c$) generated from the beginning normal factorization, and thus the efficiency of factorization can be enhanced. Option iii, factorization with identical pattern and identical row, can be more efficient than that of option ii, because option iii directly updates the matrices $L$ and $U$ generated from the previous factorization if the $A$-matrix elements do not change significantly. During extended

periods of time between network-switching events, when the DAE structure remains unchanged, the factorization with identical pattern and identical row is an attractive option. Option iv, no factorization, is used when the matrix *A* does not change and can be utilized repeatedly after a first factorization and so is selected when implementing the nonlinear solver VDHN.

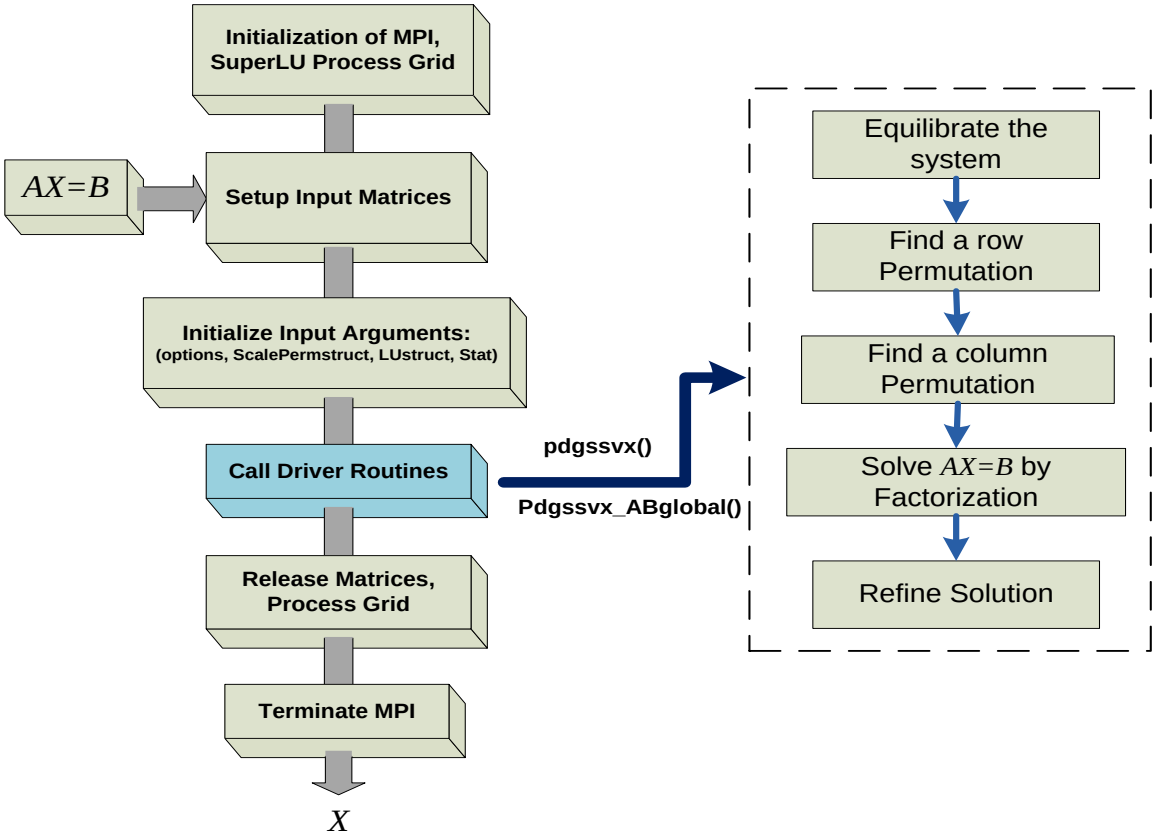



**Figure 6.2 Basic Rroutines of Distributed-Memory Parallel SuperLU**

## 6.3 Distributed-Memory Parallel SuperLU

Figure 6.2 illustrates the basic routines for parallel SuperLU with MPI. There are two main driver routines to solve systems of linear equations, pdgssvx_ABglobal for the global input interface, and pdgssvx for the distributed interface. The introduction in [43] indicates

that the routines implement the following functions: i) equilibrating the system if *A* is poorly scaled, ii) finding a row permutation that makes the diagonal of *A* large relative to the off-diagonal, iii)finding a column permutation that preserves the sparsity of the *L* and *U* factors, iv) solving the system $AX = B$ for $X$ by factoring A followed by forward and backward substitutions, and v) refining the solution. The difference between the sequential SuperLU and the distributed-memory parallel SuperLU is that the matrices *L* and *U* in the parallel SuperLU are distributed in a two-dimenional block-cyclic fashion so that the linear equations can be solved in parallel. The configuration on parallel computing is initialized by the SuperLU process grid. More details about the data structures used in parallel SuperLU are elaborated in paper [43].

## 6.4 Simulation Results Related to SuperLU Performance In HSET-TDS

The performance of a linear solver has great impact on time-domain simulation, since the solution of linear equations represents the greatest part of the Newton-Raphson method. HSET-TDS includes many linear-solver libraries, and the performance of SuperLU can be very efficient. Table 6.2 illustrates the performance of different linear solvers in HSET-TDS, applied to the New England system described in section 4.4.1 with the same contingency and simulation time. Other linear solvers compared with SuperLU include dense LU factorization and the sparse generalized minimum residual method (GMRES) from GMM++. The numerical method used in the sparse GMRES solver is iterative, so it can deal with the ill-conditioned matrix A in solving AX=B. The ill-conditioning of matrix A is possibly caused by different degree level parameters in some electrical elements. Also, when the solution vector produced by the Newton method is near to a final solution, some items of the Jacobian

matrix can be very small, and under this circumstance the matrix A may be ill-conditioned. The iterative method in the solution of linear equations can be more effective in dealing with the ill-conditioning problem as compared to direct solution.

**Table 6.2 Comparison between SuperLU, Dense LU and Sparse GMRES**

| Integration Methods | | Dense LU | Sparse GMRES | SuperLU |
|---|---|---|---|---|
| **Trapezoidal Rule ( h = 0.01s)** | Time(s) | 320.779 | 31.85 | 1.907 |
| | Speed-up | 1 | 10.22 | 168.18 |
| **Trapezoidal Rule ( variable step)** | Time(s) | 261.972 | 24.759 | 1.298 |
| | Speed-up | 1 | 10.85 | 201.83 |
| **HH4 ( h = 0.1s)** | Time(s) | 346.130 | 17.557 | 0.705 |
| | Speed-up | 1 | 19.72 | 491.28 |
| **HH4 ( variable step)** | Time(s) | 395.596 | 19.611 | 0.782 |
| | Speed-up | 1 | 20.17 | 505.81 |

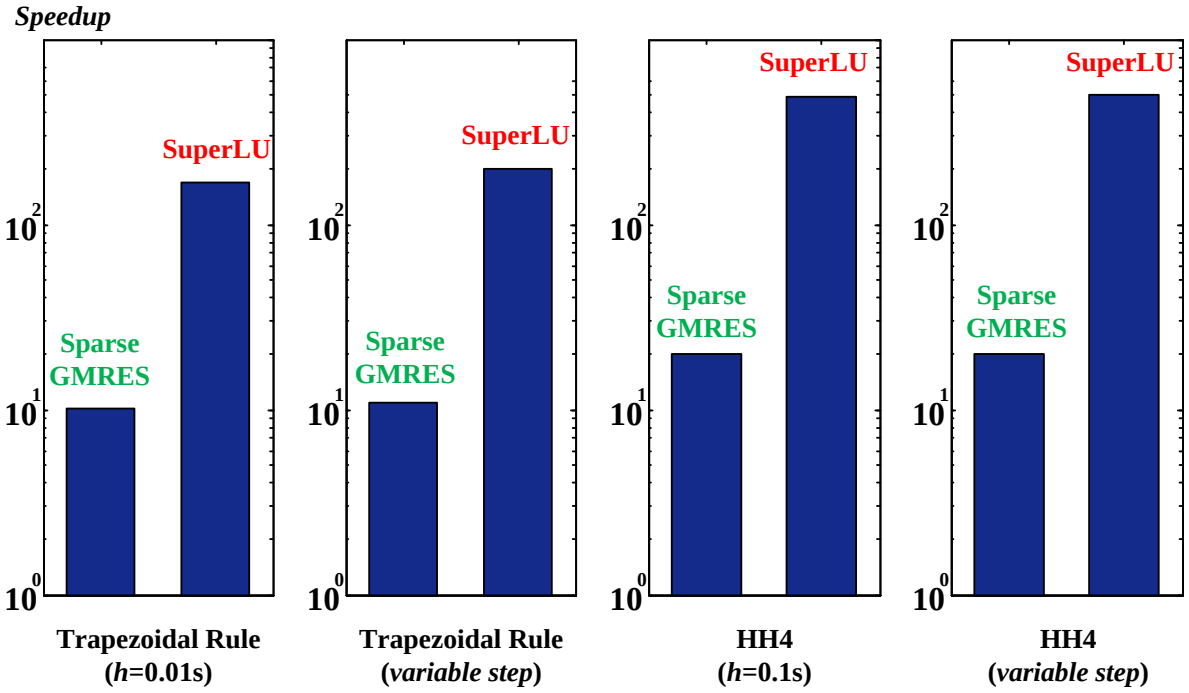



**Figure 6.3 Speedup of SuperLU, Sparse GMRES to Dense LU**

From the simulation results, we can find that SuperLU is significantly more efficient than the dense LU factorization and sparse GMRES approaches. Since the SuperLU linear-solver library fully considers the sparsity of the matrix A, and the Jacobian matrix in power

system problems are usually highly sparse, it is natural that the efficiency of SuperLU will be much higher than that of dense LU factorization. For GMRES, there are two possible reasons why SuperLU is faster. The first is that the algorithm used in GMRES requires more computation because of its iterative character, while SuperLU is more like direct solution, and iteration will also be needed when unacceptably large error is produced. The second reason is related to different programming approaches, such as how to manage memory, that may lead to different performance.

# CHAPTER 7 EXTENDED OBJECT-ORIENTED PROGRAMMING IN HSET-TDS

## 7.1 The Attributes of Object-Oriented Programming

The progress of computer hardware has deeply influenced the development of new software approaches, especially in large-scale software packages for industrial application. In the power system field the main problems related to software coding focuses on the development and maintenance of EMS software, which is usually composed of many parts with different functions. Due to the fact that power systems are comprised of many different types of electrical and magnetic elements, and that different power systems have various topological structures, it is not realistic to code the software in a static mode that requires complete re-coding if there are any changes in the power system. Additionally, in order to permit appropriate software maintenance, it is required that the EMS software must be freely adaptable to any modification condition. A programming methodology able to make some of the parts in EMS software freely changeable, recombinable or deletable, is highly desired in coding software for power-system EMS tools.

Object-oriented programming (OOP), a software coding strategy, has been used in the large-scale software development industry for over twenty years [48, 49]. In the power system field, OOP methodology of has been discussed and applied in EMS software development, and gratifying computational results have been achieved [50, 51, 52, 53, 54, 55] using OOP. The objected-oriented approach and its advantages can be demonstrated as follows [50].

1) Physical objects and their models can be presented as objects in the programming environment. It can be more convenient for engineers to directly focus on the concepts of the element model rather than being concerned be with computer-related problem such as keeping variables and data structures consistent.
2) Objects are highly independent modules, and processing is performed by sending messages between them. The attribute of independence of objects makes it convenient to maintain and update software if there are useless modules to delete or new modules to add.
3) Objects that own the same properties and perform the same operations can be arranged into classes. Therefore, the whole software structure can be organized by designed hierarchies. This technique is very useful to combine some parts of the modules and manage the whole structure.

The advantages of objected oriented programming makes it ideal for EMS software development. A first important reason for this is that EMS software needs to be highly flexible. Usually different utilities or control centers have different requirement for EMS software, and it must meet different practical conditions. Additionally, the power system itself has a certain lifetime over which the requirements may vary.. A second reason is that the function of the EMS tools must be expanded, and more and more new functions are needed for such things as power system security analysis. A software designing strategy such as OOP is helpful to deal with increased complexity in the development of EMS software tools.

## 7.2 Object-Oriented Programming in HSET-TDS

HSET-TDS was developed based on concepts of object-oriented programming(OOP), and OOP ideas may not only inform the construction of power system themselves, but also may be instructive on how to solve the mathematical equations describing the power system. The construction of a power system using OOP is illustrated in many technical papers [50, 51, 52, 53, 54, 55], and the general organization structure is illustrated in Figure 7.1.

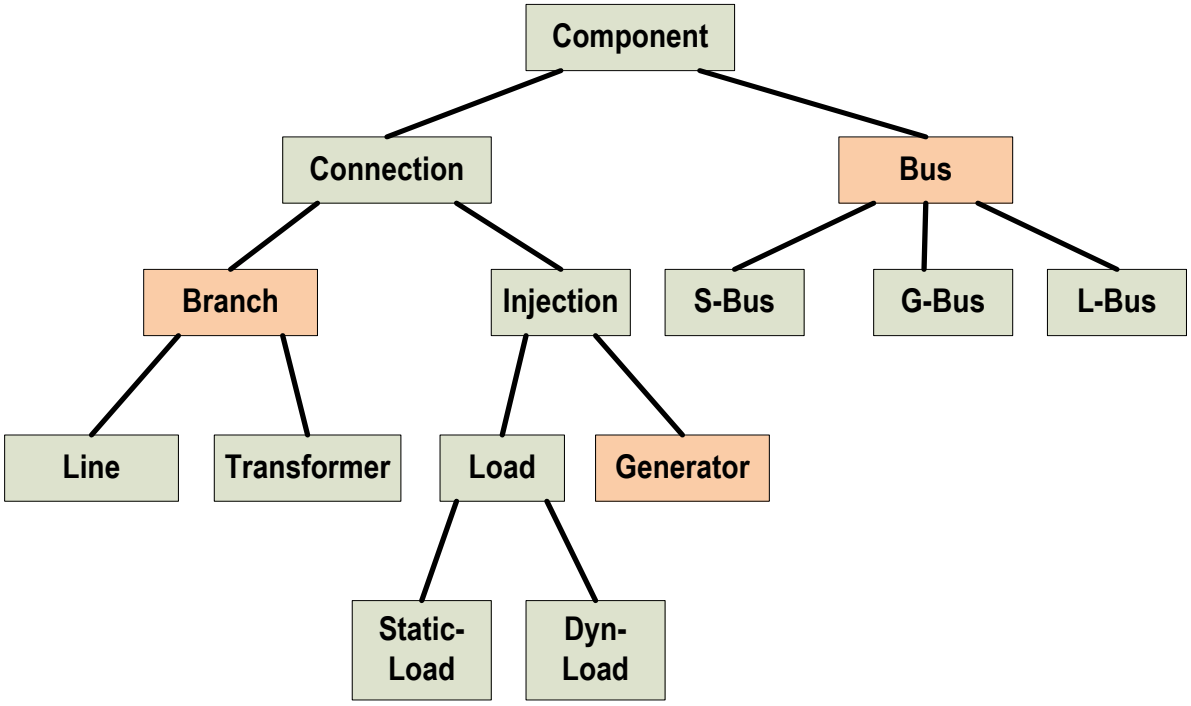



**Figure 7.1 The General Component Structure by OOP**

Figure 7.1 exhibits the basic hierarchial relationships between basic elements. Since there can be many types of electrical elements, we must model all of them in software, which can represent difficult management issues. It would be convenient if we could classify these elements into a small number of categories. In HSET, there are three basic categories: i) bus, ii) generators, iii) branches. In the bus category , all the information about bus is included, and the load information may be added if there is a load connected to the bus. In the

generator category, basic static generators information of is included along with dynamic information when appropriate. In the branch category, transmission lines and transformers are the basic elements

Bus and branch categories mainly control input of load flow data, and the generator category is responsible for input dynamic generator data and some part of the load flow data related to the generator. The relationship between these three categories is shown in Figure 7.2. The load flow data will be the basic data for constructing the classes for branches, buses, and generators. The dynamic data will be used in modeling generators in the classes related to generators. All classes of branches, buses and generators will comprise the final basis for the construction of the entire power system model. In the following sections, the inner structure of these classes will be discussed.

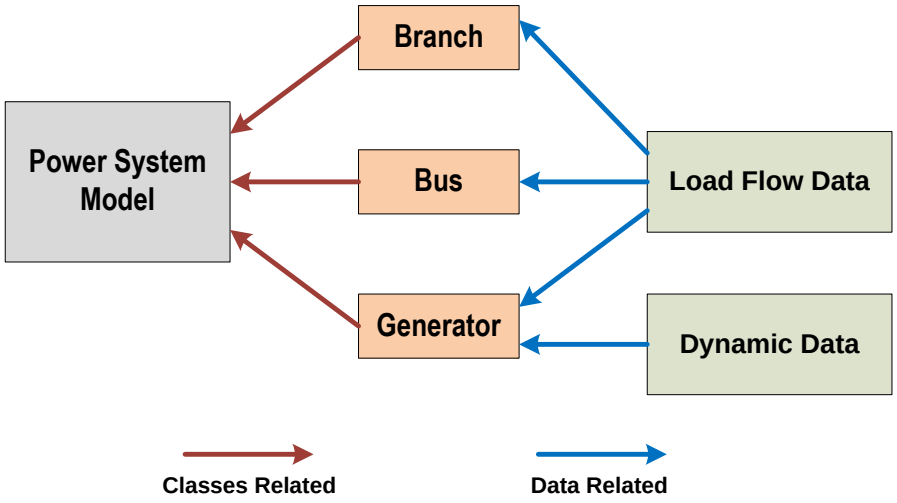



**Figure 7.2 The Basic Structure By OOP in HSET-TDS**

### 7.2.1 The Bus Classes

Buses are the basic electrical elements in power systems that play the role of nodes in the node-branches power system model . Buses can be categorized into many different types

for different usage, such as i) PQ bus, PV bus, swing bus and isolated bus, ii) connection bus, generator bus, and load bus, and iii)normal bus and faulted bus. In the process of coding bus classes in HSET-TDS, all possible cases in the class have been considered. On the other hand, since the bus data are mainly provided by the load-flow data file, such as a raw file for PSSE, it is necessary that the buses class be able to handle all the load flow data, because some data such as area number and zone number may not be used for certain types of calculation, but may be used in future modules. In HSET-TDS, all types of data and variables refer mainly to the description of raw data used by PSSE.

**Table 7.1 Variable Description of the Basic Bus Class**

| No. | Items | Description |
|---|---|---|
| **1** | Bus number | Number of the Bus in raw data |
| **2** | Bus name | Name of the bus in raw data |
| **3** | Bus type | Load bus; Gen bus; Swing bus; Isolated bus |
| **4** | Area ID | Area ID |
| **5** | Zone ID | Zone ID |
| **6** | Bus voltage | Voltage of the bus |
| **7** | Bus angle | Angle of the bus |
| **8** | Voltage Level | Voltage level |
| **9** | Complex power of load | Power of the load if it exists |
| **10** | Complex injective power | Sum injective power |
| **11** | Complex injective current | Sum injective current |
| **12** | Yii of the bus | Diagonal item of admittance Y matrix |
| **13** | Yij of the bus | Non-diagonal item of admittance Y matrix |
| **14** | Bus connection number | The number of buses connected |
| **15** | Bus connection position | The bus position in the array which is connected |
| **16** | Generator connection Number | The number of generators connected |
| **17** | Generator connection position1 | The position of generator in raw data |
| **18** | Generator connection position2 | The position of generator in dynamic data |
| **19** | Load connection number | The number of loads connected |
| **20** | Load connection position | The position of loads in raw data |
| **21** | Load DAE class | The class which describes the DAE of loads connected to the bus |
| **22** | Bus DAE class | The class which describes the Bus of load |

In the main bus class, the configuration of the variables follows the structure shown in Table 7.1. The basic variables are based on the variable configuration in the raw data, and the calculation follows from the load flow computation. Also, since the main function of the basic bus class is to assist in the construction of bus differential and algebraic equations (DAE), the object class of bus DAEs as well as the object class of load DAEs must be added.

The bus DAEs class is another class used to describe bus elements. This class is more hierarchical than the basic bus DAE class since the construction of the bus DAEs class is based on the basic bus class. The main function of bus DAEs class is to organize differential and algebraic equations describing the elements connecting to the bus, such as generators and loads. The construction of the bus DAEs class requires the generator class and load class , and the bus DAE class is used to construct the more hierarchical class, the system DAEs class. The configuration of the bus DAEs class follows the structure shown in Table 7.2.

**Table 7.2 Variable Description of the Bus DAEs Class**

| No. | Items | Description |
|---|---|---|
| **1** | Generator connection number | The number of generators connected |
| **2** | Generator ID connected | The ID of generator connected |
| **3** | Generator connection position1 | The position of generator in raw data |
| **4** | Generator connection position2 | The position of generator in dynamic data |
| **5** | Load connection number | The number of loads connected |
| **6** | Function number | The total number of functions |
| **7** | Differential functions number | Differential functions number |
| **8** | Algebraic functions number | Algebraic functions number |
| **9** | Extra algebraic functions number | Extra algebraic functions number |
| **10** | Generators functions number | Generators functions number |
| **11** | Loads functions number | Loads functions number |
| **12** | Variables number | Variables number |
| **13** | Differential variables position | Differential variables position |
| **14** | Algebraic variables position | Algebraic variables position |
| **15** | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| **16** | Elements position of Jacobian | Elements position of Jacobian in one row |
| **17** | Initial Value | Initial Value of each variable |

### 7.2.2 The Branches Class

The branch is another basic element in the node-branch model of power system, and its function is to connect buses. Branches can be classified into two main categories: i) normal lines or zero-impedance lines, ii)transmission lines or transformer connections. In the branch class , both of these types must be involved, and for basic transmission lines and transformer lines, the impedance must be separately computed. The basic function of the branch class is to construct an impedance matrix for the network. The construction of an impedance matrix also must be based on the basic bus class. In HSET-TDS, the transmission lines' impedances will be transmitted to the basic bus class, since it would be more convenient there to construct impedance matrix. The branch class is constructed based on the structure shown in Table 7.3.

**Table 7.3 Variable Description of the Branch Class**

| No. | Items | Description |
|---|---|---|
| **1** | From bus number | Starting bus of the branch |
| **2** | To bus number | Ending bus of the branch |
| **3** | Label | Label name |
| **4** | Branch state | Normal or zero impedance |
| **5** | Branch type | Transformer or transmission line |
| **6** | Branch name | Branch name |
| **7** | R | Resistance |
| **8** | X | Inductance |
| **9** | Bd2 | Half of susceptance |
| **10** | KT | Transformer rate |
| **11** | CT | Angle shifter rate |
| **12** | Compensation Complex Power I | Compensation Complex Power at starting bus |
| **13** | Compensation Complex Power J | Compensation Complex Power at ending bus |
| **14** | Impedance matrix 2 ×2 | Impedance matrix 2 ×2 |

### 7.2.3 The Class about Generators

Generators are an important type of element that take the role of supplying power to power networks,. In the input files regarding generators, static and dynamic data are usually supplied separately. The separation of static data and dynamic data makes it convenient to implement either static or dynamic analysis of power system. In the process of coding a dynamic simulator, the classes describing statics and dynamics must also be defined separately.

The class of static generator data controls the power injected into power system networks, regardless of what kind of generator models are used to supply the power. In HSET-TDS the construction of the generator class for static data is based on the configuration of load-flow raw data. Similar to the construction of buses and branches, all the data in the load flow raw data needs to be modeled in the class. The configuration of the generator class for static data is shown in Table 7.4.

**Table 7.4 Variable Description of the Generator Class for Static Data**

| No. | Items | Description |
|---|---|---|
| **1** | Bus number | Bus number related to the generator |
| **2** | Gen ID | Generator ID |
| **3** | Area ID | Area ID |
| **4** | Bus scheduled voltage | Scheduled voltage for generator bus |
| **5** | Bus scheduled angle | Scheduled angle for generator bus |
| **6** | P | Output active power |
| **7** | Q | Output reactive power |
| **8** | Max P | Maximum of active power |
| **9** | Min P | Minimum of active power |
| **10** | Max Q | Maximum of reactive power |
| **11** | Min Q | Minimum of rective power |
| **12** | Generator model description | Generator model description detail |
| **13** | Dynamic Data Class | The class which describe the dynamic model for generator |

The class of dynamic data of generators is controls the input of parameters used to model various generators. Since there are many different types of generators, each type must be modeled separately. In HSET-TDS, a basic generator model for dynamic data is constructed, and there are sub-classes describing different generator model as follows:

1) 2nd order simplified Classic model, (just $X_d'$)
2) 2nd order Classic model ($X_d'$ and $X_q'$)
3) 3rd order model ($E_q'$)
4) 4th order model ($E_q'$and $E_d'$)
5) 6th order model (($E_q''$and $E_d''$)
6) 6th order model considering generator speed dynamics in stator equations.
7) 8th order model ( modeling from perspective of flux, no simplification, and considering stator dynamics)
8) GENROU model ($E_d'$, $E_q'$ , $\Psi_d'$, $\Psi_q'$)
9) GENROU model ($E_d'$, $E_q'$ , $\Psi_d'$, $\Psi_q'$, and considering speed dynamics in stator equations)
10) GENROU model ($E_d'$, $E_q'$ , $\Psi_d'$, $\Psi_q'$, and considering stator dynamics)

The basic generator model follows the configuration shown in Table 7.5.

Like the basic generator class, the subclass describing each generator model follows the configuration shown in Table 7.5 in constructing the class. The difference between these classes lies in that the realization that the class is different, since the model equations are themselves different and dependent on the model.

**Table 7.5 Variable Description of the basic class of generator for dynamic data**

| No. | Items | Description |
|---|---|---|
| **1** | Bus number | Bus Number |
| **2** | Gen ID | Generator ID |
| **3** | Complex bus voltage | Complex bus voltage |
| **4** | Complex output power | Complex output power of the generator |
| **5** | Complex output current | Complex output current of the generator |
| **6** | Description of generator model | Description of generator model |
| **7** | Ra | Generator parameter Ra |
| **8** | Xad | Generator parameter Ra |
| **9** | Xd | Generator parameter Xd |
| **10** | Xq | Generator parameter Xq |
| **11** | Xl | Generator parameter Xl |
| **12** | Xd1 | Generator parameter Xd1 |
| **13** | Xq1 | Generator parameter Xq1 |
| **14** | Xd2 | Generator parameter Xd2 |
| **15** | Xq2 | Generator parameter Xq2 |
| **16** | Td1 | Generator parameter Td1 |
| **17** | Tq1 | Generator parameter Tq1 |
| **18** | Td2 | Generator parameter Td2 |
| **19** | Tq2 | Generator parameter Tq2 |
| **20** | H | Generator parameter H |
| **21** | Damping | Generator parameter Damping coefficient |
| **22** | Basis power | Basis power of generator |
| **23** | Omega basis | Omega basis |
| **24** | Time basis | Time basis |
| **25** | TJ | Generator parameter TJ |
| **26** | Model type | Model type |
| **27** | Function number | The total number of functions |
| **28** | Differential functions number | Differential functions number |
| **29** | Algebraic functions number | Algebraic functions number |
| **30** | Extra algebraic functions number | Extra algebraic functions number |
| **31** | Generators functions number | Generators functions number |
| **32** | Loads functions number | Loads functions number |
| **33** | Variables number | Variables number |
| **34** | Differential variables position | Differential variables position |
| **35** | Algebraic variables position | Algebraic variables position |
| **36** | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| **37** | Elements position of Jacobian | Elements position of Jacobian in one row |
| **38** | Initial Value | Initial Value of each variable |
| **39** | Class of simplified $2^{nd}$ order model | Class of simplified $2^{nd}$ order model |
| **40** | Class of classic $2^{nd}$ order model | Class of classic $2^{nd}$ order model |
| **41** | Class of $3^{rd}$ order model | Class of $3^{rd}$ order model |
| **42** | Class of $4^{th}$ order model | Class of $4^{th}$ order model |
| **43** | Class of $6^{th}$ order model | Class of $6^{th}$ order model |
| **44** | Class of $6^{th}$ order model2 | Class of $6^{th}$ order model with extra condition |
| **45** | Class of GENROU model | Class of GENROU model |
| **46** | Class of $8^{th}$ order model | Class of $8^{th}$ order whole model |

In the generator model with order larger than 2, extra models such as an excitation model or a governor model are needed . For these extra models, other subclasses must be inserted in the generator model classes, and the configuration is similar to that shown in Table 7.5.

### 7.2.4 The System Class

Bus, branch and generator classes including interior elements such as excitation and governors are helpful in constructing the system class that will describe the differential and algebraic equations of the entire power system. Since network equations are easy-to-construct algebraic equations,there is no specific class for describing networks in HSET-TDS. The basic configuration of the system class follows the description shown in Table 7.6.

**Table 7.6 Variable Description of the System Class**

| No. | Items | Description |
|---|---|---|
| 1 | System buses number | System buses number |
| 2 | Y matrix arrays | Admittance matrix description |
| 3 | Y matrix elements number | Admittance matrix description |
| 4 | Y matrix buses position | Admittance matrix description |
| 5 | Y matrix elements position | Admittance matrix description |
| 6 | Function number | The total number of functions |
| 7 | Differential functions number | Differential functions number |
| 8 | Algebraic functions number | Algebraic functions number |
| 9 | Extra algebraic functions number | Extra algebraic functions number |
| 10 | Generators functions number | Generators functions number |
| 11 | Loads functions number | Loads functions number |
| 12 | Variables number | Variables number |
| 13 | Differential variables position | Differential variables position |
| 14 | Algebraic variables position | Algebraic variables position |
| 15 | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| 16 | Elements position of Jacobian | Elements position of Jacobian in one row |
| 17 | Initial Value | Initial Value of each variable |

## 7.3 Extended Object-Oriented Programming in HSET-TDS

Some reports describing commercial software for time-domain simulation discuss the numerical methods internal to the software. It can usually be found that there is just one type of method coded in a particular software package. One probable reason for this is that a numerical method like an integration method can be formulated with reference to the specific differential equations, and this makes the coding process easy. Another possible reason is that, during the simulation, one type of software doesn't need many numerical methods, and one method such as the trapezoidal rule as an integration method or the Newton-Raphson method as a nonlinear solver, LU decomposition method as linear solver is often enough,.

Since one of our objectives is to select a best numerical method from various methods, the comparison of different numerical methods is very necessary for our work. If we adopt the traditional approach in developing simulator software, we must code a lot of software to test different numerical methods. In HSET-TDS, we adopt another strategy based on extended object-oriented programming to deal with the coding work for numerical methods.

The object-oriented programming discussed in [50, 51, 52, 53, 54, 55], is mainly focused on expanding the construction of the power system, and there is no information regarding application of OOP to the numerical methods. Extended OOP can make the classes of numerical methods embeddable, and it would be easy to select a particular numerical method for the time domain simulation. Figure 7.3 shows that configuration of integration methods, non-linear solvers, and linear solvers in HSET-TDS, and Figure 7.4 shows the relationship between the system class and the numerical methods classes. The following section will discuss the detain about numerical methods by extended OOP in HSET-TDS

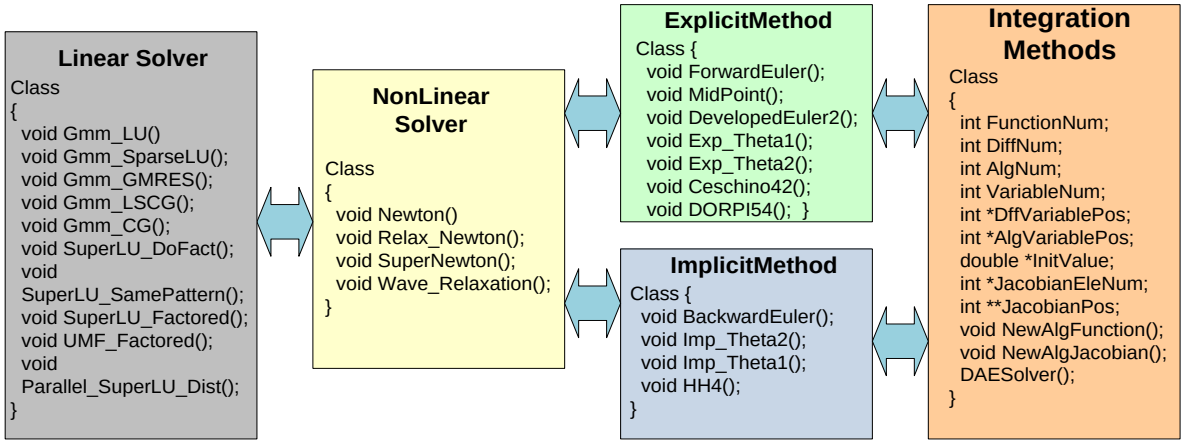



**Figure 7.3 Structure of Numerical Methods in HSET-TDS by Extended OOP**

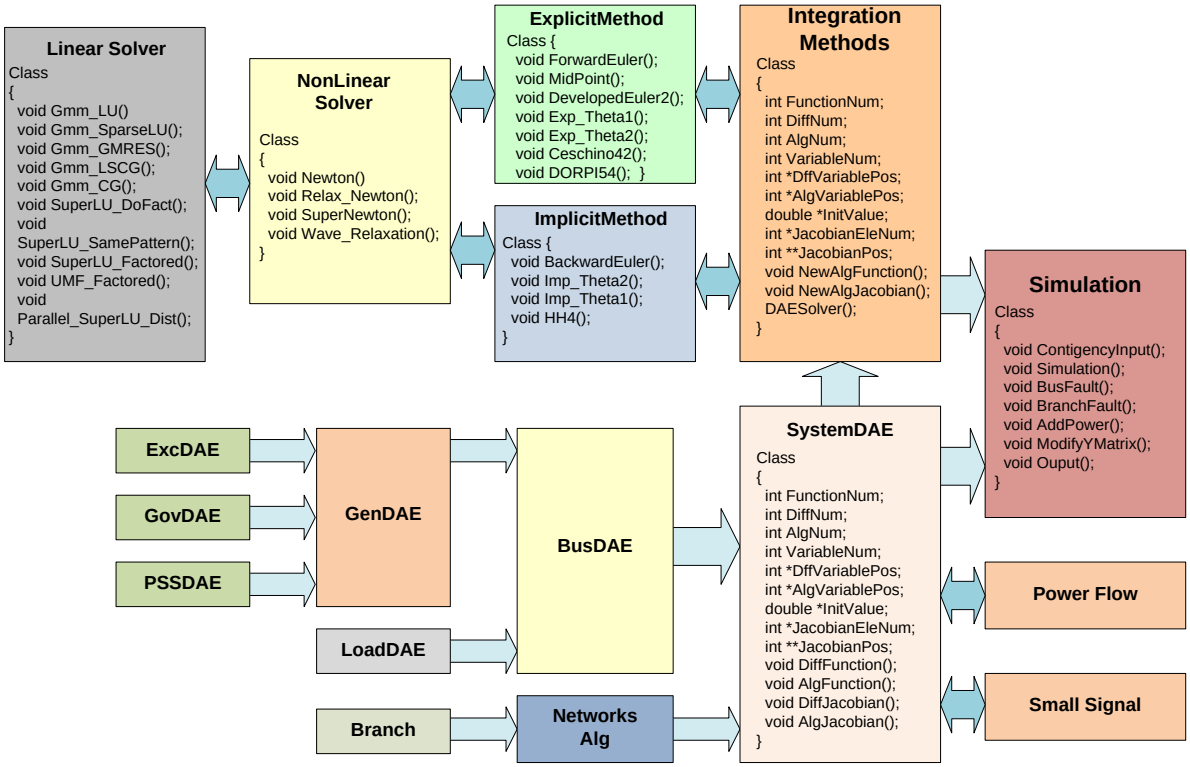



**Figure 7.4 Main classes and their relationship in HSET-TDS by OOP**

### 7.3.1 The Integration Methods Class

The integration method is the core tool for solving differential equations. Since a power system can be described by a large set of differential and algebraic equations, we need to construct integration method classes able to handle all the DAEs. In HSET-TDS, the system can be constructed by the system DAE class introduced in section 7.2.4. Another class which is directed toward solving system DAEs must be constructed in HSET-TDS. Table 7.7 shows the configuration of the class of solving system DAEs.

**Table 7.7 Variable Description in the Class of Solving System DAE**

| No. | Items | Description |
|---|---|---|
| 1 | Function number | The total number of functions |
| 2 | Differential functions number | Differential functions number |
| 3 | Algebraic functions number | Algebraic functions number |
| 4 | Variables number | Variables number |
| 5 | Differential variables position | Differential variables position |
| 6 | Algebraic variables position | Algebraic variables position |
| 7 | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| 8 | Elements position of Jacobian | Elements position of Jacobian in one row |
| 9 | Initial value | Initial value of each variable |
| 10 | Precision | Calculation precision |
| 11 | Time step | Integration time step |
| 12 | Last step | Last integration time step |
| 13 | Solution now | Solution of this step |
| 14 | Solution of last step | Solution of last step |
| 15 | Error now | Numerical error of this step |
| 16 | Explicit method class | Sub class for explicit method |
| 17 | Implicit method class | Sub class for implicit method |

As described in chapter 4, there are two kinds of integration methods, explicit and implicit. In HSET-TDS, we construct classes for each of these methods , and they are sub-classes inside of the class of solving system DAE. Table 7.8 illustrates the configuration of the implicit method class, and the configuration of the explicit method class is similar.

**Table 7.8 Variable Description in the Class of Implicit Method**

| No. | Items | Description |
|---|---|---|
| **1** | Function number | The total number of functions |
| **2** | Differential functions number | Differential functions number |
| **3** | Algebraic functions number | Algebraic functions number |
| **4** | Variables number | Variables number |
| **5** | Differential variables position | Differential variables position |
| **6** | Algebraic variables position | Algebraic variables position |
| **7** | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| **8** | Elements position of Jacobian | Elements position of Jacobian in one row |
| **9** | Initial value | Initial value of each variable |
| **10** | Precision | Calculation precision |
| **11** | Time step | Integration time step |
| **12** | Last step | Last integration time step |
| **13** | Solution now | Solution of this step |
| **14** | Solution of last step | Solution of last step |
| **15** | Error now | Numerical error of this step |
| **16** | Algebraic solver class | Algebraic solver class |
| **17** | Step type | Variable step or fixed step |
| **18** | Sub solver type | Integration solver type |
| **19** | Algebraic method type | Algebraic method type |
| **20** | Linear solver type | Linear solver type |

In the implicit method class, there are four main functions related to different implicit methods: i) the backward Euler method, ii) the Theta 1 method, iii) the Theta 2 method, and iv)HH4. The realization of these functions is based on their specific formulas. An algebraic solver is needed inside each of the implicit methods, to solve the algebraic equations which obtained from the discretization of differential equations and the algebraic part of the DAEs. In HSET-TDS, there is an independent class in charge of solving algebraic equations, as discussed in the following section.

### 7.3.2 The Solving Non-linear Algebraic Equations Class

The solving non-linear algebraic equations class is necessary for both explicit method and implicit method classes, and also includes the power flow class and extra analysis classes for such tasks as determining a balance point. In the solving common algebraic equations

class , several different non-linear solver methods like the basic Newton method, the Newton method with relaxation, and the very dishonest Newton method must be included,. Table 7.9 shows, the basic class configuration of non-linear solvers in HSET-TDS

**Table 7.9 Variable Description in the Class of Non-linear Solver**

| No. | Items | Description |
|---|---|---|
| **1** | Function Source | The origin of the algebraic equations |
| **2** | Function number | Function number |
| **3** | Solution | Solution of the equations |
| **4** | Elements Number of Jacobian | Elements Number of Jacobian in one row |
| **5** | Elements position of Jacobian | Elements position of Jacobian in one row |
| **6** | Precision | Precision of the calculation |
| **7** | Linear solver class | Linear solver class |
| **8** | B | B in AX=B |
| **9** | X | X in AX=B |
| **10** | A | A in AX=B |

In the processing of solving non-linear algebraic equations, a linear solver is needed when the Newton method is used. Since in the nonlinear solver clas all the methods involved are Newton-method like, the linear solver must be involved. Also, in order to save the software storage space, the variables arrays *A, X, B* are defined in the nonlinear solver class.

### 7.3.3 The Solving Linear Algebraic Equations Class

The solving linear equations class is an independent classthat can be connected to a nonlinear solver or power flow solver. In HSET-TDS, we used the open-source linear solver libraries to solve linear equations, and therefore the solving linear solver class is the interface to the open source libraries that include GMM++, SuplerLU and UMFpack. Since each library has different data structures related to the *A*, *X*, *B* variables in equations of form *AX=B*, the linear solver class must define different data structure inside the class. Details about SuperLU has been discussed in chapter 6, and the details of GMM++ and UMFpack

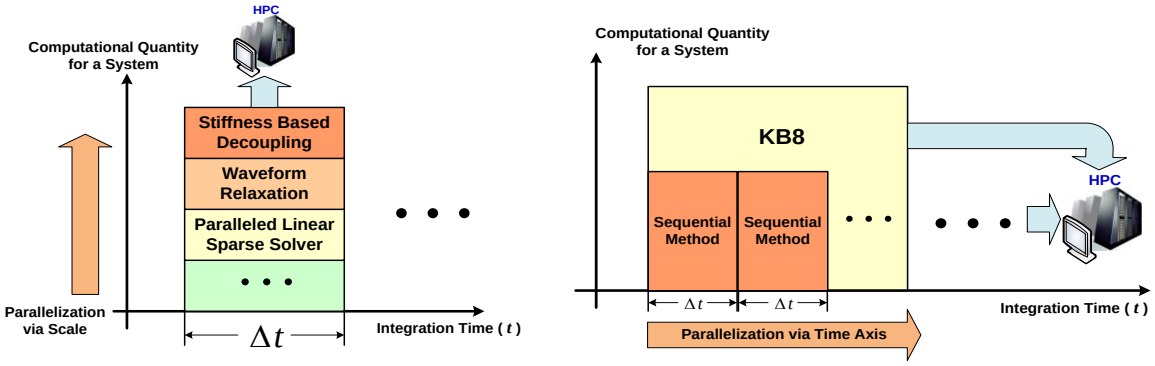
can be found in their manual []. The linear solver functions in the class are shown in Table 7.10.

**Table 7.10 Linear Solver Functions**

| No. | Items | Description |
|---|---|---|
| **1** | Gmm_LU() | Dense LU factorization solver from GMM++ |
| **2** | Gmm_SparseLU() | Sparse LU factorization solver from GMM++ |
| **3** | Gmm_GMRES() | GMRES solver from GMM++ |
| **4** | Gmm_LSCG() | Unpreconditioned Least Square Conjugate Gradient from GMM++ |
| **5** | Gmm_CG() | Unpreconditioned Least Square Conjugate Gradient from GMM++ |
| **6** | SuperLU_DoFact() | Basic solver in SuperLU |
| **7** | SuperLU_SamePattern() | Solver with setting of Same-Pattern in SuperLU |
| **8** | SuperLU_SamePattern_Row () | Solver with setting of Same-Pattern-Same-Row in SuperLU |
| **9** | SuperLU_Factored() | Solver with setting of Factored in SuperLU |
| **10** | UMF_CBLAS() | Basic solver in UMFpack with CBLAS |

# CHAPTER 8 PARALLEL DESIGN FOR EXTENDED-TERM TIME-DOMAIN SIMULATION

With the development of high-performance computers, parallel computing has become a popular topic in the area of time-domain simulation of power systems. One important problem for any parallel computing activity is how to efficiently and optimally select a partition scheme for separating the original problem into several sub-problems that can be distributed to a collection of processors. For time-domain simulation, there are two kinds of partition scheme: 1) partitioning via the scale of the power system as shown in Figure 8.1(a) , 2) partitioning via the time axis asshown in Figure 8.1 (b).




(a) Partitioning via Scale (b) Partitioning via Time-axis

**Figure 8.1 Partitioning Scheme for Parallel Computing of Time Domain Simulation**

1) Parallelization via scale can be efficient for time-domain simulation of large systems because of the parallelized aspects of integration methods, non-linear solvers, and sparse linear solvers.

2) Parallelization via the time axis has greater potential for parallelization because of the infinite time axis, and it eventually can significantly speed up the simulation.

This chapter will discuss some designs for extended-term time-domain simulation using parallel computing. Although some of designs may not be able to render the simulation as efficient as when the number of processors is increased, the objective of this chapter is to supply some experience of parallel computing for extended-term time-domain simulation. Additionally, a partitioning scheme via the time axis, leading to partitioning of a series of cascading events, is introduced to improve the efficiency of time domain simulation.

## 8.1 Parallel Computing Design A

Figure 8.2 shows our overall design for high-speed execution of extended-term dynamic-security assessment. The solution process is divided into three distinct levels. The first level is the partition of DAEs into stiff and non-stiff parts by the recursive projection method (RPM). For the non-stiff part, the explicit method will be used to formulate the differential equations that can be directly assigned into *m* processors since these formulated differential equations are naturally decoupled. The second level is the partition of the stiff part of the DAE using a Waveform Relaxation Method (WRM) with an Epsilon decomposition algorithm to continue partitioning the stiff part of the DAE. Each partitioned fragment can be formulated by an implicit method to ensure stability. The third level is the solution process using a Newton-like method. The very dishonest Newton (VDHN) can be used to fix the Jacobian matrix constant for several steps, and the Multi-frontal Massively Parallel sparse direct Solver (MUMPS) [56,57] will be adopted to solve the sparse linear

equations. The final solution for one step will be integrated, and the computational error will be estimated to control stiffness detection and the step values for the stiff and non-stiff parts.

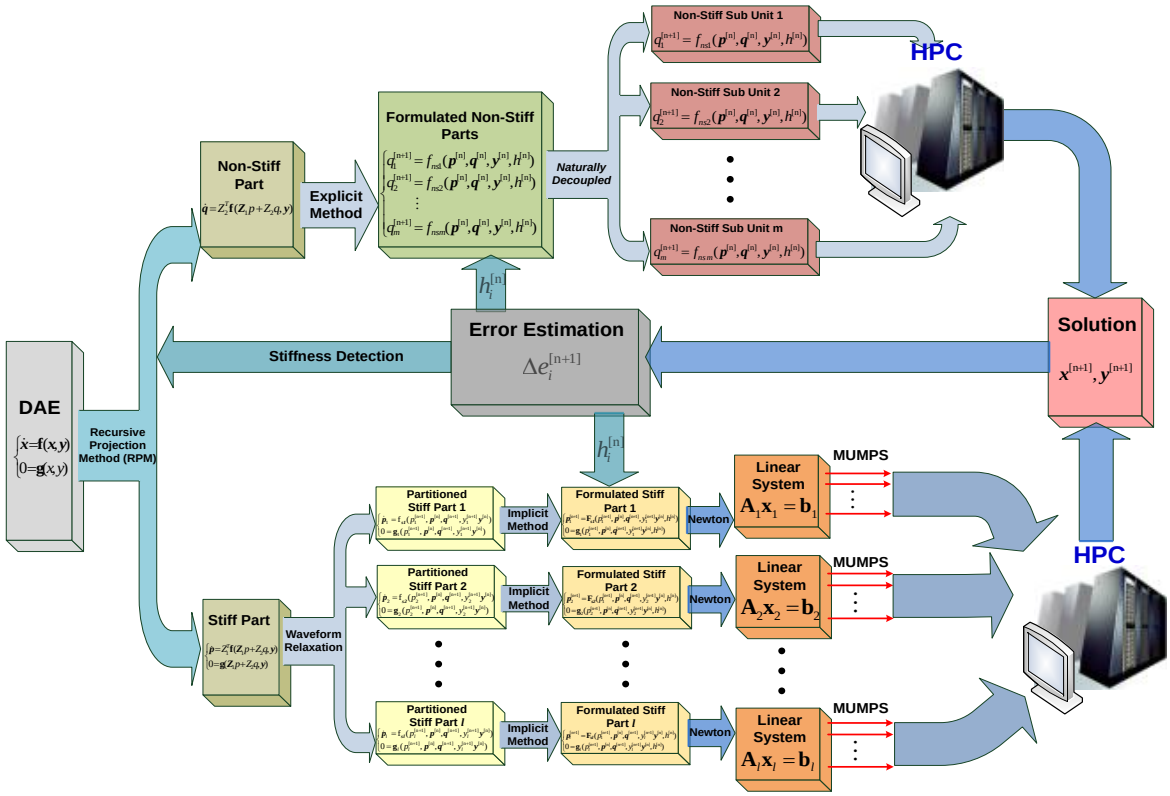



**Figure 8.2 Parallel computing strategy of DAEs for Blue Gene/L**

There are 3 special cases of concern in the design of Figure 8.2

- At some integration steps, the ODE parts may not be very stiff so that all differential equations can be solved by the explicit method, and thus the bottom branch of Figure 8.2 need not be applied.
- Even in the case where stiff parts are detected, it may be difficult to find an epsilon decomposition scheme that results in WRM convergence, and so in this case, the lower branch of Figure 8.2 will be applied without WRM; here, load balance can be maintained by appropriate parallelization via MUMPS.

- Under the circumstance of failure in stiffness detection, the entire integration is done by WRM.

The ability to detect and respond to these special cases results in a design in which the different partition methods and linear solution methods cooperate to balance the DAE solution load at each processor. A key feature of this design is that the DAE is divided into two different integration schemes, one for stiff and one for non-stiff parts, and both parts are then solved via a parallelized implementation.

Chapter 5 discussed a stiffness decoupling method that can be used in the partition process of design A. According to the simulation results in section 5.3, the simulation by the stiffness decoupled method is not significantly more efficient than the implicit method. There are probably two reasons for this:

1) The stiffness decoupling process only focuses on the differential equations of a DAE system. If the number of differential equations in the DAE system is relatively smaller than that of algebraic equations, the efficiency of stiffness decoupling method will be not significant.

2) The application of asparse linear solver such as SuperLU increases the efficiency of simulation by implicit method.

Also, the stiffness decoupling method can change the sparsity pattern of the Jacobian matrix in solving $AX=B$, and VDHN cannot be used. The stiffness decoupling method may not be as efficient as the implicit method with VDHN.

## 8.2 Parallel Computing Design B

Figure 8.3 and Figure 8.4 show another design using parallel computing for time-domain simulation. In contrast to design A, the main idea of design B is try to divide the time axis, and distribute each of the time subintervals into different processors. The advantage of this scheme is that the computational activity within each processor can be totally independent, and the communication time between each processor can thus be decreased substantially. However, the problem for design B is how to obtain the initial values, necessary for each processor simulation, for each time subinterval. .

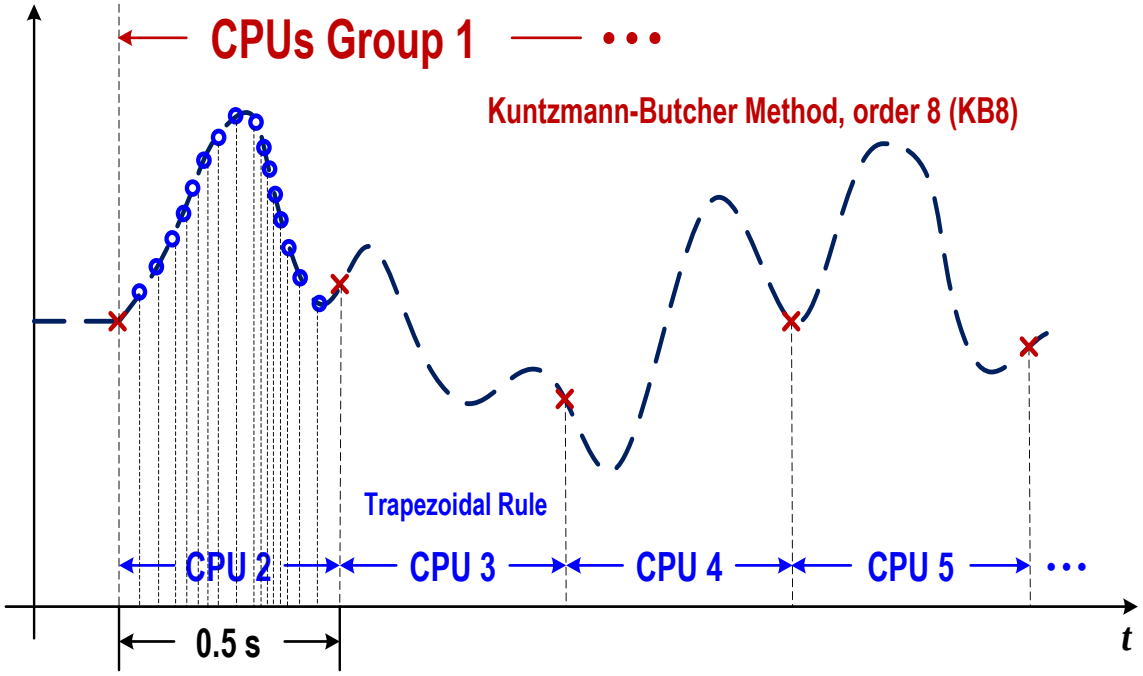



**Figure 8.3 Design B-1: Parallel Computing Strategy of DAEs**

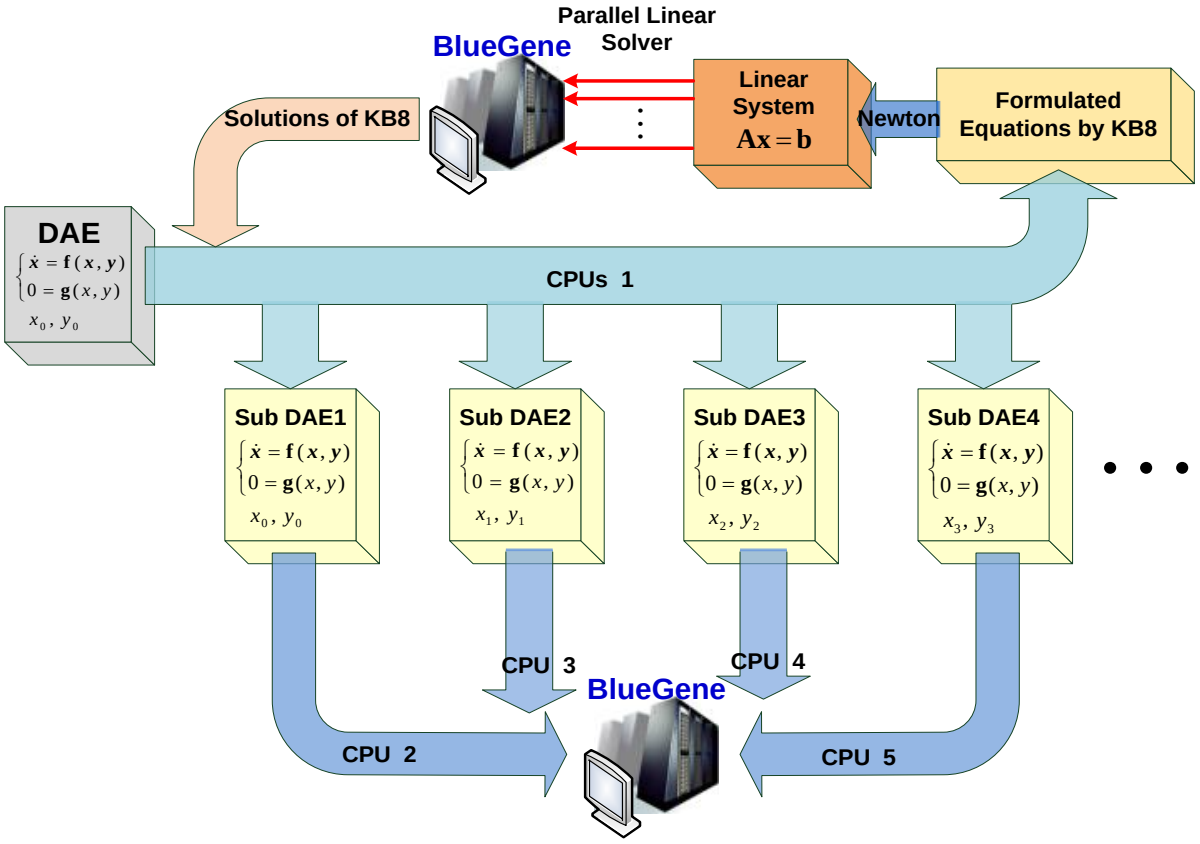



**Figure 8.4 Design B-2: Parallel Computing Strategy of DAEs**

In order to deal with this initial-values problem, we propose to use a new integration method, the Kuntzmann-Butcher Method – order 8 (KB8)[27, 36]. Like HH4, KB8 is symmetrical A-stable and therefore presents no stability problem. Furthermore, KB is much more precise than HH4, and we expect that larger integration steps can be used for simulation without leading to numerical stability problems. The concept of design B can generalized as follows.

1) One processor is used to calculate the initial points with larger integration step and KB8.

2) The entire time axis will be separated into several subintervals at the points calculated by processor 1.
3) Other processors will be used to simulate within the time subintervals with the initial points provided by processor 1.
4) The final simulation results are constructed by the cumulative simulation results from all processors.

In Design B, the most important process is to partition the simulation via time axis by the Kuntzmann-Butcher Method – order 8 (KB8K). The formula of KB8 can be described as follows.

$$\begin{cases} K_1 = f[x_n + h[\omega_1 K_1 + (\omega_1' - \omega_3 + \omega_4')K_2 + (\omega_1' - \omega_3 - \omega_4')K_3 + (\omega_1 - \omega_5)K_4]] \\ K_2 = f[x_n + h[(\omega_1 - \omega_3' + \omega_4)K_1 + \omega_1' K_2 + (\omega_1' - \omega_5')K_3 + (\omega_1 - \omega_3' - \omega_4)K_4]] \\ K_3 = f[x_n + h[(\omega_1 + \omega_3' + \omega_4)K_1 + (\omega_1' + \omega_5')K_2 + \omega_1' K_3 + (\omega_1 + \omega_3' - \omega_4)K_4]] \\ K_4 = f[x_n + h[(\omega_1 + \omega_5)K_1 + (\omega_1' + \omega_3 + \omega_4')K_2 + (\omega_1' + \omega_3 - \omega_4')K_3 + \omega_1 K_4]] \end{cases} \quad \textbf{(8.1)}$$

$$x_{n+1} = x_n + h(2\omega_1 K_1 + 2\omega_1' K_2 + 2\omega_1' K_3 + 2\omega_1 K_4) \quad \textbf{(8.2)}$$

where $\omega_1 = \frac{1}{8} - \frac{\sqrt{30}}{144}$ $\qquad \omega_1' = \frac{1}{8} + \frac{\sqrt{30}}{144}$

$\omega_2 = \frac{1}{2}\left(\frac{15 + 2\sqrt{30}}{35}\right)^{1/2}$ $\qquad \omega_2' = \frac{1}{2}\left(\frac{15 - 2\sqrt{30}}{35}\right)^{1/2}$

$\omega_3 = \omega_2\left(\frac{1}{6} + \frac{\sqrt{30}}{24}\right)$ $\qquad \omega_3' = \omega_2'\left(\frac{1}{6} - \frac{\sqrt{30}}{24}\right)$

$\omega_4 = \omega_2\left(\frac{1}{21} + \frac{5\sqrt{30}}{168}\right)$ $\qquad \omega_4' = \omega_2'\left(\frac{1}{21} - \frac{5\sqrt{30}}{168}\right)$

$\omega_5 = \omega_2 - 2\omega_3$ $\qquad \omega_5' = \omega_2' - 2\omega_3'$

Similar to the formula of HH4, the stability domain of KB8 is the whole left part of the complex domain according to reference [27], and therefore KB8 is symmetrical A-stable. Also, KB8 possesses a high order of $h^8$. We expect a large speedup for design B, which means that the speedup of KB8 must be large. We are still trying to explore different strategies to accelerate the efficiency of KB8.

Table 8.1 and Figure 8.5 show the test results of KB8 on the PJM 13029 system, which is already introduced in section 4.4.3. It can be seen that the KB8 method can make the simulation successful when a large integration step is used. However, simulation time is not decreased significantly. The reason for this phenomenon is that the iteration times of Newton methods become greater when very large integration steps are used. Specifically, when VDHN is used, a new Jacobian matrix must be updated within nearly every integration step, and the updating makes the whole simulation take more time.

We expect a large speedup for design B, which means that the speedup of KB8 must be large. We are still trying to explore different strategies to accelerate the efficiency of KB8.

**Table 8.1 Simulation Results of PJM 13029 Buses System with Fixed Integration Step**

| Integration Methods | Integration Step (s) | Algebraic Solver | | Linear Solver Library | Simulation Time (s) | Correctness by Observation |
|---|---|---|---|---|---|---|
| | | Solver | Precision | | | |
| **Trapezoidal** | 0.001 | VDHN | $10^{-6}$ | SuperLU | 363.696 | (exact solution) |
| **Trapezoidal** | 0.01 | VDHN | $10^{-6}$ | SuperLU | 54.249 | Correct |
| **Trapezoidal** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 7.641 | Incorrect |
| **HH4** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 20.375 | Correct |
| **KB8** | 0.1 | VDHN | $10^{-6}$ | SuperLU | 50.324 | Correct |
| **KB8** | 0.2 | VDHN | $10^{-6}$ | SuperLU | 24.617 | Correct |
| **KB8** | 0.3 | VDHN | $10^{-6}$ | SuperLU | 21.381 | Correct |
| **KB8** | 0.4 | VDHN | $10^{-6}$ | SuperLU | 20.102 | Correct |
| **KB8** | 0.5 | VDHN | $10^{-6}$ | SuperLU | 18.056 | Incorrect |

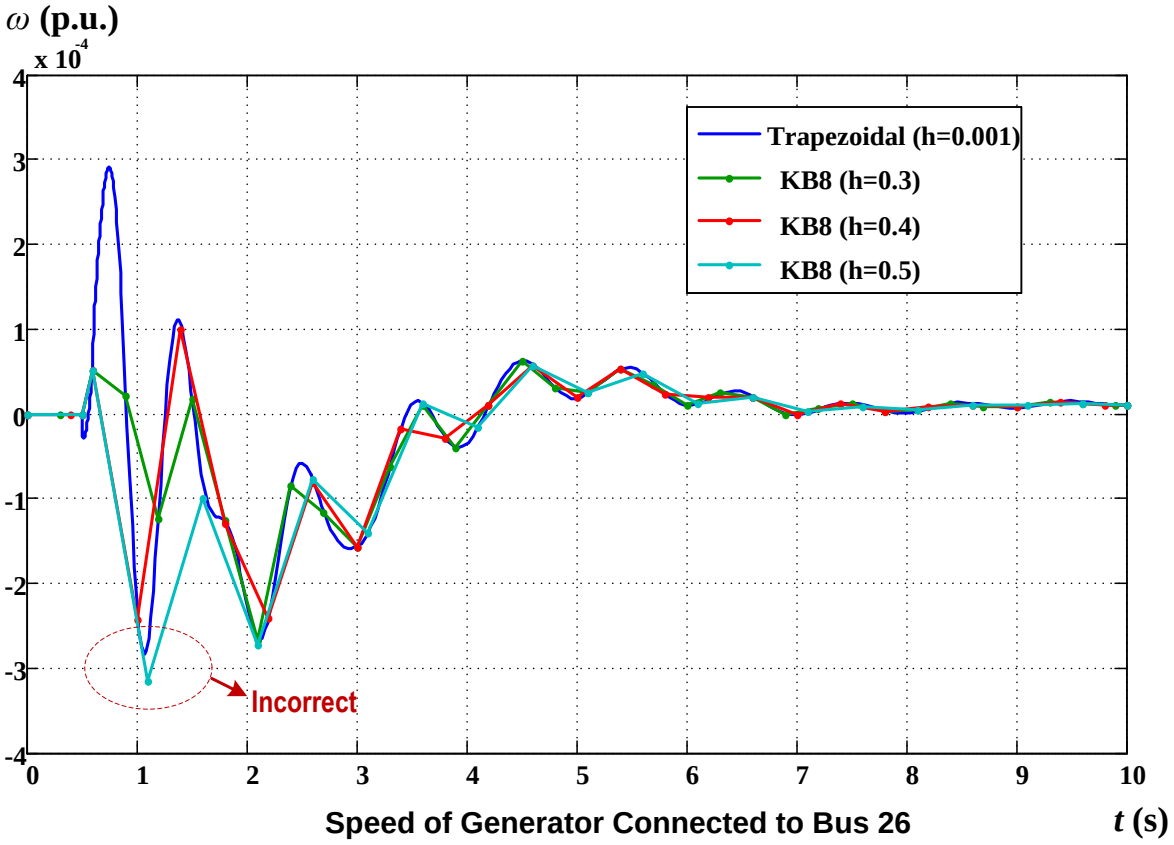



**Figure 8.5 Simulation Results of PJM 13029 Buses System by method of KB8**

## 8.3 The Partitioning Scheme Based on Numerical Methods

The strategies related to or based on design A and design B for parallel computing of time-domain simulation can be generalized as follows.

1) Alternating Solution Methods, partitioning the system into buses with dynamics and buses without dynamics -- The weakness of this method is that the bus voltages need to be iterated during each integration step. Although the sub-system scale is decreased, the cost of extra iterations of bus voltage can make the overall computation slow. This phenomenon

was observed from the simulation of HSET-TDS, but more simulation experience is still needed for clarification.

2) Alternating Solution Methods, partitioning the system into sub-systems via several cut sets -- The weakness of this method is similar to that of the previous method. The iteration of buses' cut-set voltages can make the whole computation slow. This method would be attractive either when the cut set does not exist (natural islands) or is very small (just one transmission line). However, for realistic power systems, this situation virtually never exists. We still need more evidence to verify the method.

3) Waveform Relaxation, with a partitioning scheme of buses with dynamics and networks (Design A) -- There are two components of cost in this method: i) We need to use a symmetrical A-stable implicit integration method to avoid numerical instability, but waveform relaxation actually makes the implicit integration method semi-explicit. ii) Some algebraic variables must be iterated between each set of partitioned sub-systems. In simple terms, there are two main problems, a) numerical stability becomes bad, and b) the total computation time cannot be significantly decreased, especially when there are many sub-systems. The phenomenon was observed from the simulation on HSET-TDS, but more tests are needed.

4) Stiffness-decoupled method, partitioning differential equations into stiffness and non-stiffness parts (Design A) -- This method works, but the total computation time cannot be significantly decreased. The problem is that, in a realistic power system, the dimension of algebraic equations (not just including network equations, but also including algebraic equations in dynamic elements) are usually much greater than those of differential equations. More computation time may actually be spent in solving algebraic equations. Saving only

differential equation solution time cannot be very effective for the overall solution. The phenomenon is observed in the simulation on HSET-TDS, also discussed in section 5.3 and section 8.1

5) Time-axis decoupled method, by KB8 (Design B) -- The problem of this method is how to quickly obtain the initial points for each sub-interval. The practical test of KB8 shows that KB8 is unable to determine the sub-interval initial points very quickly. I think the reason is that KB8 is required to solve a system with a four-fold increase in the original dimensions. Besides, large integration steps also increase the iteration times of the Newton method, creating an other possible problem. The problem related to KB8 is also discussed in section 8.2.

6) Parallelization in the process of solving linear equations by using a different distributed-memory linear solver, such as distributed SuperLU (Design A) -- The parallelization of distributed linear solver generally is not able to remarkably decrease simulation time. The problem is that the Jacobian matrix cannot be randomly partitioned into several independent sub-matrices. Thus, there can be a lot of communication between processors. If the matrix can be apportioned into independent sub-matrices, the system actually becomes a set of islands, a situation that essentially never exists. This phenomenon was observed in the simulation on HSET-TDS when distributed parallel SuperLU is used. However, more simulation cases are needed to verify the conclusion.

7) Parallelization in the process of solving linear equations by a different shared-memory (multi-thread) linear solver, such as multi-threads SuperLU -- There are two main problems here: i), the good news is that the shared-memory linear solver may work for parallelization, since the entire loop can be applied to parallelization. However, the speed-up

can be worse if more cores are used. Since the Jacobian matrix essentially cannot be partitioned into independent sub-matrices, there will be more communication between cores. ii) Most supercomputers do not support multi-threads. At ISU, the Bluegene and Moutain supercomputer doesn't support a shared-memory solver. This conclusion is based on my experience, and more practical evidence is needed.

## 8.4 Partitioning Scheme via Cascading Events

The discussion of section 8.3 describes the difficulty in finding an appropriate partition scheme based on numerical methods. The main reason for this difficulty is that the power system is usually strongly-coupled, and it is thus difficult to find several relatively independent sub-systems that can be effectively distributed to a collection of processors. Also, because of the high efficiency of a sparse liner solver like SuperLU, the computational time during one integration step can be much less than 1 second for a large power system like the PJM system. It is often difficult to get a desirable speedup using parallel computing on a collection of processors, since there may be relatively considerable time spent in processor-processor communication. In order to utilize the supercomputer using parallel computing, we must find another method to partition the whole simulation task.

For extended-term time-domain simulation, a series of cascading events must simulated, and hours of simulation time may be required. If the time interval between two continuous events is long enough, for example 10 minutes, it is possible to use two processors to simulate these two events because the transient process of an event can be finished in 30 seconds or less, usually in 10 seconds. If the two collections of events lie within an interval of 10 minutes, we can distribute them to two processors. The initial point

of the simulation for each processor can be calculated by power flow. For the given series of events using time-domain simulation, the overall process of parallel computing can be described as follows.

1) Analyze the events sequence, and split it into several parts. The criterion for splitting is that those events which occur during a given period of time, say 10 minutes, can be categorized as one group. The start time of the last event in one group must be longer than a certain period so that the dynamics of the last event can be completed when the first event of the next group starts.
2) Distribute each group of events into different processors, and the initial points for time-domain simulation by each processor can be calculated by power flow, such as line outage or injective power compensation, based on the topology effected by the last group of events,.
3) If all the simulation results from processors are stable, the final simulation results are the combination the results from these processors. However, if any one of the simulation results is unstable, that the final simulation result is unstable, and the simulation curves are the combination of the stable curves before occurrence of the unstable one.
4) When the instability case occurs, HSET-TDS should supply corrective action to prevent the instability. The corrective action can be supplied by optimum power flow and unit commitment. After the corrective action is implemented, a new sequence will be obtained. HSET-TDS should recheck the stability of the system. The process can be repeated from step (1) until the simulation is stable.

The process of parallel computing can be illustrated based on the example in section 4.4.4 and shown in Figure 8.6. Table 8.2 shows the simulation results of the 39-bus New England case for a 1-hour simulation on 6 CPUs.

From Table 8.2, it can be seen that when 6 processors are used, the speedup of HSET-TDS versus PSSE can be as much as 64.7 times when the integration method of HH4 is used. The advantage of partitioning via simulation events lies in the fact that the tasks distributed to different processors are independent, so there is no communication time. However, the disadvantage is that the partitioning number cannot be freely configured, since it is limited by the distribution of simulation events along the time axis.

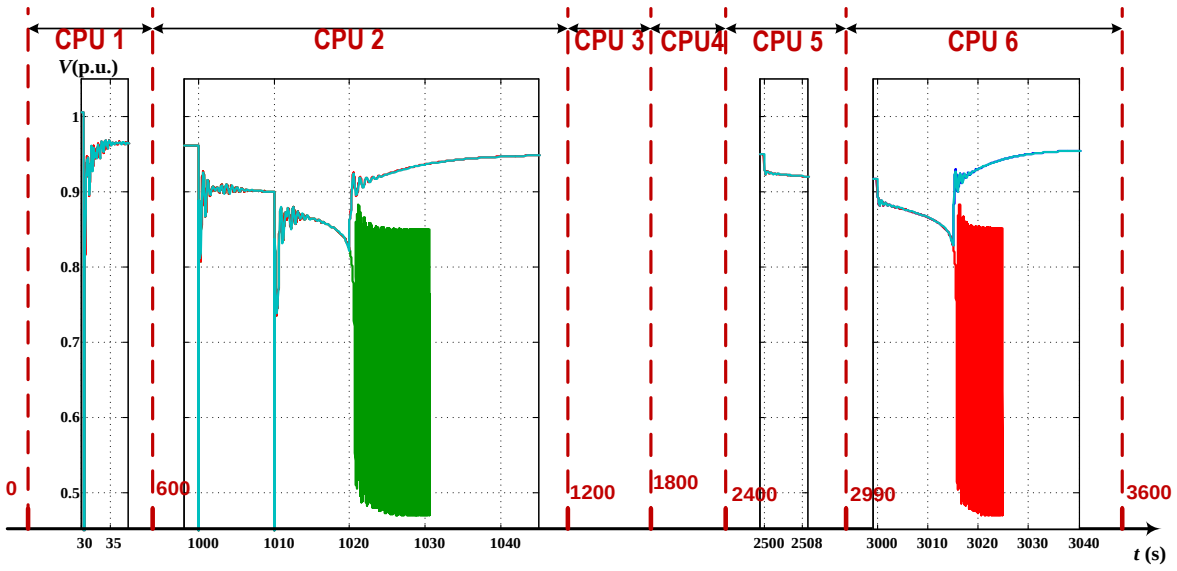



**Figure 8.6 Partitioning via Different Events for Parallel Computing**

**Table 8.2 Simulation Results for 3600 sec. with Variable Integration Step on 6 CPUs**

| CPU | Simulation Interval | Integration Methods | Algebraic Solver | Linear Solver Library | Sim. Time (s) | Speedup (HSET/ PSSE) |
|---|---|---|---|---|---|---|
| **CPU1** | **[0s, 600s]** | Trapezoidal | VDHN | SuperLU | 3.962 | |
| **CPU2** | **[600s, 1200s]** | Trapezoidal | VDHN | SuperLU | 5.366 | 22.9 |
| **CPU3** | **[1200s, 1800s]** | Trapezoidal | VDHN | SuperLU | 1.112 | |
| **CPU4** | **[1800s, 2400s]** | Trapezoidal | VDHN | SuperLU | 1.112 | |
| **CPU5** | **[2400s, 2990s]** | Trapezoidal | VDHN | SuperLU | 3.921 | |
| **CPU6** | **[2990s, 3600s]** | Trapezoidal | VDHN | SuperLU | 3.932 | |
| **CPU1** | **[0s, 600s]** | HH4 | VDHN | SuperLU | 1.210 | |
| **CPU2** | **[600s, 1200s]** | HH4 | VDHN | SuperLU | 1.900 | 64.7 |
| **CPU3** | **[1200s, 1800s]** | HH4 | VDHN | SuperLU | 0.420 | |
| **CPU4** | **[1800s, 2400s]** | HH4 | VDHN | SuperLU | 0.420 | |
| **CPU5** | **[2400s, 2990s]** | HH4 | VDHN | SuperLU | 1.110 | |
| **CPU6** | **[2990s, 3600s]** | HH4 | VDHN | SuperLU | 1.032 | |
| **Simulation in PSS/E with integration step of 0.001** | | | | | around 123s | |

# CHAPTER 9 CONCLUSION

## 9.1 Summary

A new control center functionality called high-speed extended term (HSET) time-domain simulation (TDS) is proposed for online analysis of cascading. The key to this functionality is computational speed. The work reported uses state-of-art algorithms and hardware to identify maximum on-line computational speed for HSET-TDS. The computational speed is intended to be enhanced from five hierarchical aspects: hardware, strategies, integration methods, nonlinear solvers, and linear-solver libraries. The work elaborated herein can be generalized as follows.

- HH4 is a promising integration algorithm for enhancing efficiency of power system time-domain simulation because it is not only symmetrically A-stable but also highly precise. The theoretical analysis of numerical stability shows that HH4 has capability for avoiding numerical instability and hyper-stability while maintaining numerical precision. It is therefore reasonable to expect that HH4 can decrease the number of integration steps and thereby save computation time. Case studies using the developed algorithm are presented and discussed in the second part of the thesis, along with the introduction of other numerical methods adopted in the simulation.
- A high-speed extended-term (HSET) time-domain simulator (TDS) has been developed within a direct-solution strategy using different integration methods, and various nonlinear and linear solvers. We have employed HH4, VDHN, and SuperLU to accelerate computational speed. Results of simulation tests support the assertion

that the integration method HH4 is significantly faster than the commonly-used trapezoidal rule, because it can enlarge the integration step while still maintaining acceptable numerical precision. Our results on short-term (10 seconds) time domain simulation show that, for a given precision, the average speedup of HH4 versus the trapezoidal rule is greater than 2 times if the same nonlinear and linear solvers are used. The results on the extended-term (1 hour) time-domain simulation show that the speedup of variable-step HH4 versus variable-step trapezoidal rule is greater than 3 times, and the speedup of variable-step HH4 versus fixed-step method in PSSE is greater than 20 times.

- The stiffness-decoupling method is advantageous for application of integration methods in time-domain simulation of power systems. With the proposed method for error estimation, the stiffness for certain DAE systems can be determined and the whole DAE system can be decoupled into stiff and non-stiff parts. The simulation in section 5.3 shows that the simulation efficiency can be enhanced through this method..
- A linear-solver library is an indispensible tool when solving non-linear equations by the Newton method. SuperLU, an ope- source linear-solver library from Berkeley, is introduced and applied in HSET-TDS. The goal of the application of SuperLU in HSET-TDS is to expand the linear-solver libraries in HSET-TDS, especially for the parallel version. The simulation results for sequential computing shows that SuperLU is competitive and suitable for time-domain simulation of power system.
- The software of HSET-TDS has been developed based on the theory of object-oriented programming (OOP) methodology. OOP enables the functionality and maintenance of HSET-TDS to be more versatile and convenient. Additionally, we

extended OOP concepts into the programming of numerical methods, which therefore lets HSET-TDS include various numerical methods in categories of integration methods, non-linear solvers, and linear solvers. This advantage makes newly-developed numerical methods more easily embedded into HSET-TDS.

- We have reported on algorithmic design A and design B for deployment on high-performance parallelized computer architectures shown in section 8.1 and section 8.2. A key feature of these two designs is that the simulation of DAE results employs system-scale (design A) and time-axis (design B) partitioning. Moreover, another strategy, partitioning the simulation task via time axis, and specifically via cascading events, is desribed. The advantage of this strategy is that the partitioned tasks can be totally independent. The simulation results shown in section 8.4 demonstrates the speedup, which can be as high as 60 compared with PSSE simulation software.

## 9.2 Contribution

The achieved and expected contribution of this work may be summarized as follows.

1) HSET-TDS, a new software package for control centers and based on object-oriented programming methodology has been developed. HSET-TDS has both sequential and parallel versions. Each version includes many different numerical methods intended to significantly enhance computational speed. This high-speed capability of HSET-TDS makes it both possible and practical to effectively analyze high-consequence events such as cascading, making it beneficial for the operation of control centers.

2) A new integration method, Hammer-Hollingswoth 4 (HH4), has been first applied in the field of power systems. Compared to the Trapezoidal rule, a traditional integration method adopted by many commercial software packages, HH4 is not only symmetrical A-stable, but is also highly precise, and therefore the integration step can be enlarged while maintaining precision. HH4 can be a very efficient integrator for time-domain simulation of power systems.

3) An overall design of various numerical methods for time-domain simulation has been developed. Of all these methods, the direct-solution method, HH4, very-dishonest Newton, and linear solver of SuperLU have been recommended. The simulation results on New England 39-bus system, an expanded 8775-bus system and the PJM 13029-bus system show that this design is able to accelerate the efficiency of time domain simulation as much as 20 times compared with industrial software PSSE.

4) An efficient stiffness-decoupling strategy has been explored. The main idea of this scheme is to detect the stiffness of ODE using numerical error analysis, and this technique is more efficient than the stiffness-detection strategy performed by seeking Jacobian matrix eigenvalues . The stiffness-decoupling strategy makes it possible to solve the stiff part of an ODE by implicit methods and to solve the non-stiff part of the ODE by explicit methods. Since explicit methods are much more efficient than implicit methods, the computation speed of time-domain simulation can be further improved.

5) The exploration of sequential and parallel linear solver libraries, such as SuperLU, UMFpack, GMM++, MUMPS, and LAPACK will be carried on. The comparison of these available linear solver libraries will be based on their use in HSET-TDS. The

simulation results can supply helpful information to the power industry regarding suitability of various sparse linear solvers for time-domain simulation of power systems.

6) Several parallel designs for deployment on super computers have been proposed. These are intended to enhance the computational speed from perspectives of partitioning via the scale of system and the time axis. The strategy of partitioning the simulation task via cascading events is suggested, since the partitioned tasks for each processor can be totally independent. The simulation results show that this strategy can enhance the efficiency of simulation by as much as 60 times compared to PSSE.

## 9.3 Future Work

HSET-TDS has been developed with certain functions, and more functions need to further improved from the following aspects:

1) Various models for generator, excitation and governors have been developed in the HSET-TDS shown in Chapter 3, and these models are needed to both short-term and extended-term time domain simulation. More models may need to be developed for the practical extended-term time domain simulation, such as boiler, automatic generation control (AGC).
2) HH4 is an attractive integration method for time domain simulation, especially when variable step technique is used. In order to enhance the efficiency further, the following aspects may need to be investigated: auxiliary integration method for error estimation, more efficient linear solver.

3) The parallel computing in the process of solving DAE system can improve the simulation efficiency, especially for the multi-cores supercomputer where share-memory programming is used. The algorithm of how to partition the solution process of DAE system for shared-memory programming needs to be explored so that the efficiency of the extended term time domain simulation can be improved tremendously.

# ACKNOWLEDGEMENTS

I would like to acknowledge and extend my heartfelt gratitude to my major professor and advisor, Dr. James McCalley, for his guidance, support and encouragement throughout my doctorate study in Iowa State University. With his academic, technical, and editorial advice, this dissertation and relative journal papers can be completed smoothly. Especially, when I met difficulties about the development of HSET-TDS, his insightful suggestion plays a very important role in solving all intricate problems. Besides, he has taught me a number of invaluable lessons about how to implement academic research and how to continue the future work in industry.

I would also like to express my warmly gratitude to Dr. Ajjarapu and Dr. Aliprantis for their classes of EE554, EE590, and presiding EPRC Seminar, where I strengthened my understanding of power system dynamics and simulation. My thanks also go to Dr. Govindarasu and Dr. Hou for the discussion about the numerical algorithm of computing programming in the platform of parallel supercomputer. Additionally, I would like to appreciate the help from Dr. Terry Smay who supplied me with much constructive suggestion of revising this dissertation.

In addition, special thanks go to my parents and friends for their consideration and motivation.

Part of this work was sponsored by PSerc S-32 “Fast simulation, monitoring, and mitigation of cascading failure” and DOE project “New Security Tools for Real-Time Operations”. I am grateful for their support.

# BIBLIOGRAPHY

[1] Q. Chen and J. McCalley, "Identifying High-Risk N-k Contingencies for On-line Security Assessment," IEEE Trans. on Power Systems, Vol. 20, Issue 2, May 2005 pp. 823 – 834.

[2] U.S.-Canada Power System Outage Task Force, "Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations," April 2004.

[3] Q. Chen, "The probability, identification and prevention of rare events in power system" PhD Thesis, ECE, Iowa State University, 2004.

[4] S. K. Khaitan, "On-line cascading event tracking and avoidance decision support tool" PhD Thesis, ECE, Iowa State University, 2008.

[5] S. Khaitan, C. Fu and J. McCalley, "Fast Parallelized Algorithms for On-Line Extended-Term Dynamic Cascading Analysis", invited paper at IEEE PES Power Systems Conference and Exposition, Seattle WA, March 15-18, 2009.

[6] B.Stott. " Power system dynamic response calculations," Proc, IEEE, vol. 67, pp. 219-240, Feb. 1979.

[7] Taoka H., Abe S., Takeda S., "Fast Transient Stability Solution Using An Array Processor", IEEE Transactions on Power Apparatus and Systems Volume PAS-102, Issue 12, Dec. 1983 Page(s):3835 – 3841.

[8] F. De Mello, J. Feltes and T. Laskowski, "Simulating fast and slow dynamic effects in power systems," IEEE Computer Applications in Power, Vol. 5, Issue 3, July 1992, pp. 33 -38.

[9] SIEMENS, "Power Transmission and Distribution, PSS/E TM 30.2 Online Documentation," Nov, 2005.

[10] EPRI EL 4610, "Extended Transient Midterm Stability Program," Jan, 1987.

[11] J. Sanchez-Gasca, R. D'Aquila and W. Price, "Variable time step, implicit integration for extended-term power system dynamic simulation", Proc. of the Power Industry Computer Application Conf, 1995, 1995 IEEE, 7-12 May 1995, pp. 183 – 189.

[12] J. Astic, A. Bihain and M. Jerosolimski, "The mixed Adams-BDF variable step size algorithm to simulate transient and long term phenomena in power systems", IEEE Trans. on Power Systems, Vo. 9, Issue 2, May 1994, pp. 929 – 935.

[13] O. Fillatre, C. Evrard and D. Paschini, "A powerful tool for dynamic simulation of unbalanced phenomena," Advances in Power System Control, Operation and Management, 1997. APSCOM-97. Fourth International Conference on (Conf. Publ. No. 450), Vol. 2, 11-14 Nov. 1997, pp. 526 - 531 vol.2.

[14] A. Zecevic and N. Gacic, "A partitioning algorithm for the parallel solution of differential-algebraic equations by waveform relaxation" IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, Vol. 46, Issue 4, April 1999, pp. 421 – 434.

[15] E. Lelarasmee, A. Ruehli, and A. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large-scale integrated circuits," IEEE Trans. Computer-Aided Design, vol. 1, pp. 131–145, July 1983.

[16] M. Crow and M. Ilic, “The waveform relaxation method for systems of differential/algebraic equations,” Proc. of the 29th IEEE Conf. on Decision and Control, 1990., 5-7 Dec. 1990 Page(s):453 - 458 vol.2.

[17] An IEEE Commitee Report, "Parallel processing in power systems computation", Power Systems, IEEE Transactions on Volume 7, Issue 2, May 1992 Page(s):629 – 638.

[18] http://home.gna.org/getfem/gmm_intro.html

[19] http://www.cs.berkeley.edu/~demmel/SuperLU.html

[20] http://www.cise.ufl.edu/research/sparse/umfpack/

[21] I. Decker, D. Falcao and E. Kaszkurewicz, “Conjugate gradient methods for power system dynamic simulation on parallel computers,” IEEE Trans. on Power Sys, Vol. 11, Issue 3, Aug. 1996, pp. :1218 – 1227.

[22] I. Decker, D. Falcao and E. Kaszkurewicz, “Parallel implementation of a power system dynamic simulation methodology using the conjugate gradient method,” IEEE Trans. on Power Sys, Vol. 7, Issue 1, Feb. 1992, pp. 458 – 465.

[23] W. Xue, J. Shu and Y. Wu, “Parallel algorithm and implementation for realtime dynamic simulation of power system,” International Conf. on Parallel Processing, 2005. ICPP 2005, 14-17 June 2005, pp. 137 – 144.

[24] Gross G. and Bergen A., “An efficient algorithm for simulation on transients in large power systems”, Circuits and Systems, IEEE Transactions on Volume 23, Issue 12, Dec 1976 Page(s): 791 – 799.

[25] Gross, G. and Bergen, A.R., "A class of new multistep integration algorithms for the computation of power system dynamical response", Power Apparatus and Systems, IEEE Transactions on Volume 96, Issue 1, Part 1, Jan. 1977 Page(s): 293 – 306.

[26] D. Yang and V. Ajjarapu, "A decoupled time-domain Simulation method via Invariant subspace partition for power system analysis," IEEE Transaction on Power System, Volume 21, Issue 1, Feb. 2006 Page(s): 11 – 18

[27] E. Hairer and G. Wanner, "Solving ordinary differential equations II: stiff and differential-algebraic problems", Springer-Verlag, 1996.

[28] M.L. Crow, and J.G. Chen, "The multi-rate method for simulation of power system dynamics", Power Systems, IEEE Transactions on Volume 9, Issue 3, Aug. 1994 Page(s): 1684 – 1690.

[29] Crow, M.L.; Chen, J.G.; "The multi-rate simulation of FACTS devices in power system dynamics", Power Systems, IEEE Transactions on Volume 11, Issue 1, Feb. 1996 Page(s): 376 – 382.

[30] J.S. Chai, N. Zhu and Bose, A. "Parallel Newton type methods for power system stability analysis using local and shared memory multiprocessors", Power Systems, IEEE Transactions on Volume 6, Issue 4, Nov. 1991 Page(s):1539 - 1545

[31] Paul M. Anderson, A. A. Fouad, "Power system control and stability, the Institute of Electrical and Electronic Engineers", Inc., 1994.

[32] Yixin Ni, Shousun Chen, Baolin Zhang, "Theory and analysis of power system dynamic", Tsinghua University publication house, 2002.

[33] Kundur, P. (Prabha), “Power System Stability and Control”, New York : McGraw-Hill, c1994.

[34] SIMENS, “Power Transmission and Distribution PSS/E TM 30.2 Online Documentation”, Nov, 2005

[35] Schulz, R. P. Synchronous machine modeling. Symposium on Adequacy and Philosophy of Modeling: System Dynamic Performance. IEEE Publ. 75 CH 0970-PWR, 1975.

[36] E. Hairer, S.P. Norsett and G. Wanner, “Solving ordinary differential equations I: Nonstiff Problems”, Springer-Verlag, 1987.

[37] M. Berzins and R.M. Furzeland, “An Adaptive Theta Method for the Solution of Stiff and Non-stiff Differential Equations”, Applied Numerical Mathematics, Vol 9, pp 1-19, 1992

[38] W. Hundsdorfer and V. Savcenco, “Analysis of a multirate theta-method for stiff ODEs”, Applied Numerical Mathematics, Volume 59, Issues 3-4, March-April 2009, Pages 693-706

[39] F.L. Alvarado, R.H. Lasseter and J.J. Sanchez, “Testing Of Trapezoidal Integration With Damping For The Solution Of Power Transient Problems”, IEEE Transactions on Power Apparatus and Systems Volume PAS-102, Issue 12, Dec. 1983 Page(s): 3783 – 3790.

[40] P.C. Hammer, J.W. Hollingsworth, “Trapezoidal methods of approximating solutions of differential equations”, MATC, volumn 9, 1959, Page(s) 92-96.

[41] G.M. Shroff, H.B. Keller, "Stabilization of unstable procedures: the recursive projection method", SIAM Journal on Numerical Analysis, vol.30, pp.1099-1120, August, 1993.

[42] V. Janovsky, O. Liberda, “Continuation of invariant subspaces via the recursive projection method,” Applications of Mathematics, vol.48, pp. 241-255, 2003.

[43] James W. Demmel, John R. Gilbert, Xiaoye S. Li, “SuperLU Users’ Guide”, Nov, 2007, available at (http://www.cs.berkeley.edu/ ~demmel/SuperLU.html ).

[44] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, Joseph W. H. Liu, “A supernodal approach to sparse partial pivoting”, SIAM J. Matrix Analysis and Applications, 1999, Vol 20-3, pp.720-755.

[45] Xiaoye S. Li and James W. Demmel, “Making Sparse Gaussian Elimination Scalable by Static Pivoting”, High Performance Networking and Computing Conference, Orlando, Florida, Nov. 7-13, 1998.

[46] James W. Demmel, John R. Gilbert, Xiaoye S. Li, “An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination”, SIAM J. Matrix Analysis and Applications, 1999, Vol 20-4, pp:915-952.

[47] Xiaoye S. Li and James W. Demmel, “A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems”, ACM Trans. Mathematical Software, 2003, Vol 29-2, pp:110-140.

[48] B. J. Cox, Object-Oriented Progmmming. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1986

[49] E. White, R. Malloy, et al., “Object-oriented programming,” Byte, pp. 137-235, August 1986. Series of Articles on Objectoriented Programming.

[50] Neyer, A.; Wu, F.F.; Imhof, K., "Object-oriented programming for flexible software: example of a load flow," IEEE Trans on Pwr Sys., Volume 5, Issue 3, 1990 Page(s): 689 – 696

[51] Foley, M.; Bose, A.; Mitchell, W., " An object based graphical user interface for power systems ," IEEE Trans on Pwr Sys., Volume 8, Issue 1, 1993 Page(s): 97 – 104

[52] Hakavik, B.; Holen, A.T., "Power system modelling and sparse matrix operations using object-oriented programming", IEEE Trans on Pwr Sys., Volume 9, Issue 2, 1994 Page(s): 1045 – 1051

[53] Foley, M.; Bose, A., " Object-oriented on-line network analysis", IEEE Trans on Pwr Sys., Volume 10, Issue 1, 1995 Page(s): 125 – 132

[54] Zhou, E.Z., "Object-oriented programming, C++ and power system simulation", IEEE Trans on Pwr Sys., Volume 11, Issue 1, 1996 Page(s): 206 – 215

[55] Manzoni, A.; e Silva, A.S.; Decker, I.C., "Power systems dynamics simulation using object-oriented programming", IEEE Trans on Pwr Sys., Volume 14, Issue 1, 1999 Page(s): 249 – 255.

[56] P.Amestoy, I. Duff, J. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," ENSEEIHT-IRIT Tech. Report, revision apprd in Cmput. Mthds Appl. Mech. Eng., 184, 501-520, 2000.

[57] http://graal.ens-lyon.fr/MUMPS/