

Reducing the Vulnerability of Electric Power Grids to Terrorist Attacks

Final Project Report

**Report Number DOE/ER/25671-1
(DOE Research Project DE-FG02-05ER25671)**

**Ross Baldick
Thekla Boutsika
Jin Hur
Manho Joung
Yin Wu
Minqi Zhong**

January 31, 2009

The University of Texas at Austin

Table of Contents

1	Introduction.....	2
1.1	Project Overview	2
1.2	Overview of Cascading Outage Analyzer.....	2
2	Design Patterns of COA software.....	5
2.1	Introduction.....	5
2.2	Design Patterns used in the implemented COA	9
3	Development of COA software	19
3.1	Data workflows	19
3.2	Operational algorithm.....	22
3.3	Implementation of outage checkers	24
3.3.1	Line overload and under voltage checkers	24
3.3.2	Under/Over frequency checker	25
4	Conclusion.....	50
	Appendix A: User’s Manual	51
	User’s Manual Index.....	83
	References.....	84

1 Introduction

1.1 Project Overview

This project (‘Reducing the Vulnerability of Electric Power Grids to Terrorist Attacks’) is collaborative research with the Naval Postgraduate School (NPS), under the sponsorship of the U.S. Department of Energy. The research at The University of Texas focuses on analyzing cascading outages in large-scale electricity grids, both as a standalone tool and also as a component to eventually be added to the Vulnerability of Electric Grids Analyzer (VEGA) [1].

We refer to the proposal (Wood, Salmeron and Baldick 2003), and references therein, for detailed background on the problem of electric power-grid vulnerability. In that document, goals were established for this research and its critical importance. One aspect of that research, conducted primarily at University of Texas at Austin (UT), is the development of a cascading analysis tool, building on the experiences of a prototype tool developed by Commonwealth Associates and described in a white paper [2].

This development is aimed at including the short-term effects of cascading outages into the analysis of network vulnerability. In addition to integrating representation of cascading outages into VEGA, informal feedback from industry suggested that it was also important that for this research to develop a standalone cascading analysis tool. Development as a standalone tool has also facilitated the division of activities between NPS and UT. This report details the development of the prototype standalone tool. Future work includes extending the types of initial disturbances that can be considered and integration of the tool with VEGA.

1.2 Overview of Cascading Outage Analyzer

UT has had the primary responsibility for developing a cascading outage tool that has its own graphical user interface (GUI). The work has concentrated on developing the simulation algorithm and the GUI, both in a windows-based environment. Before beginning the cascading outage analysis, we need to specify the initial disturbances to be investigated for their progression to cascading outages. In the current algorithm, line outage type is considered as an initial disturbance. The extension to other types of outages such as generator outages, bus outages, and substation outages remains future work.

Following the initial disturbance, the Cascading Outage Analyzer software has three outage checking algorithms, namely ‘Frequency Checker’, ‘Line Overload Checker’, and ‘Under Voltage Checker’, that determine the status of the resulting operating state, or ‘equilibrium’. Some resulting operating states would result in protection equipment removing more elements from the system and therefore potentially precipitating further outages. Among the three checkers, the ‘Line Overload Checker’ and ‘Under Voltage Checker’ use the AC power flow as a basic simulation engine. In the case of the frequency checker, we use the System Frequency Response (SFR) model as a frequency change model.

The Cascading Outage Analyzer (COA) program is PC-based Windows software. COA simulation engine is written on the Microsoft .NET common language runtime (CLR). The Microsoft .NET Framework (ver. 1.1 higher) must be installed on the computer before running COA software. The following table shows the summarized specifications of Cascading Outage Analyzer.

Table 1.1 Specification of COA program

Cascading Outage Analyzer	Specifications
Operating System	Window 32 bit System (Wind XP OS system)
Input Database	Microsoft Access DB (MS Access 2003 higher)
Load Flow program	AC power flow Decoupled power flow (Commonwealth Associates Inc.)
Platform	MS .NET Framework
Development Languages	Visual Basic .NET (MMI) Visual C# (Algorithms)

As indicated in the table, the Cascading Outage Analyzer (COA) uses MS Access as a database. When running COA program, users can select either of two load flow programs: Full AC power flow or Decoupled power flow, both provided by Commonwealth Associates Inc. That means that COA program requires the hardware lock key for running these load flow programs.

The following sections of this report will be organized as follows. In section II, design specification of COA software; design patterns will be introduced. In section III,

Functional specification for developing COA software focusing on data workflows, operating algorithm and implementation of outage checkers will be presented. A conclusion and future extension of COA software are given in section IV. In addition to the technical report, the user manual of COA software is provided in the Appendix.

2 Design Patterns of COA software

2.1 Introduction

Computers and specially designed software have been used extensively in the past decades in the power system industry, mainly for simulation analysis so as to assess the system's future performance and reliability. The increase in size and complexity of interconnected power systems and the automation of their operation has led to more computer applications with large and complicated software, most of which has been based on procedural programming languages. However, the liberalization of the electricity markets imposes great needs for more accessible, flexible, and expandable power system software, since there are many differences in market and system operations between regions and the deregulated environment brings along frequent and sometimes drastic changes of these operations. For these reasons the power system industry has started viewing object-oriented programming techniques as a good alternative to procedural programming languages, because they allow for more flexibility and expandability. Most recently, a further step has been taken and design patterns have been deployed in power system software to successfully address issues as flexibility, expandability, and the reusing of legacy systems.

Design patterns are part of the cutting edge of object-oriented programming (OOP) in software development. A design pattern is 'a generalized solution to a commonly occurring problem', meaning that a design pattern is specific to the problem at hand, yet general enough to address future problems and requirements. Essentially, a design pattern is a combination of two things: a description of a problem and a description of its solution. Design patterns make it easier to reuse successful software designs and architectures and the use of proven techniques like design patterns enhances their accessibility to developers of new systems. In addition, design patterns allow developers to communicate using well-known, well understood names for software interactions.

Design patterns arose from architecture and anthropology. A great architect named Christopher Alexander defined a pattern as 'a solution to a problem in a context'. Though many people had already been working on design patterns in the early 1990s, trying to establish the connections between design patterns in architecture and software design, the book that had the greatest influence on this emerging issue was *Design Patterns: Elements of Reusable Object-Oriented Software*, by Gamma, Helm, Johnson and Vlissides, also known as the Gang of Four in recognition of their contribution to the field.

Design patterns are composed of four essential parts: a pattern name, a problem context, a generalized approach to a solution, and a set of consequences. The **pattern**

name is used to describe the design problem, its solutions, and consequences in a word or two. The **problem** describes when the pattern is applicable. The **solution** describes the elements of which the design is composed along with their relationships, responsibilities, and collaborations. Finally, the **consequences** are the results and trade-offs of applying the pattern, including its impact on a system's flexibility and expandability. Design patterns are most commonly described using graphical notations. However, apart from the design pattern's name and structure other parameters should also be recorded in order to facilitate its reusability. These parameters include, but are not limited to, participants, collaborations and consequences. Participants are the classes and/or objects participating in the design pattern and their responsibilities, while collaborations are the ways the participants collaborate to carry out their responsibilities.

Following Gamma et al. [3] design patterns can be classified by two criteria: **purpose** and **scope**. According to the **purpose** classification, which reflects what a pattern does, patterns can be grouped into three general categories which are **creational**, **structural** and **behavioral**. Creational patterns concern the process of object creation. Structural patterns deal with the composition of classes or objects. Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility. The **scope** classification of a design pattern specifies whether the pattern applies primarily to classes or to objects. **Class** patterns deal with relationships between classes and their subclasses, which are established through inheritance and are static—fixed at compile-time. **Object** patterns deal with object relationships, which can be changed at run-time and are more dynamic. Almost all patterns use inheritance to some extent but most design patterns are object patterns.

Prior to presenting the design patterns used specifically in the outage checkers of the implemented cascading outage analyzer some important definitions and parameters of object-oriented programming will be given along with some clarifications on notations used in design pattern diagrams (which follow the standard UML notation).

Object-oriented programs are made up of objects, which package both data and the procedures, typically called methods or operations, which operate on that data. An object performs an operation when it receives a request (or message) from a client. The operation's signature consists of the operation's name, the objects it takes as parameters, and the operation's return value. The set of all signatures defined by the operations of an object is called the interface to the object. A type is a name used to denote a particular interface. A class defines an object's implementation. The class specifies the object's internal data and representation and defines the operations the object can perform. It is important to understand the difference between an object's class and its type. An object's class defines how the object is implemented, while an object's type only refers to its interface—the set of requests to which it can respond.

In the following a class will be depicted as a rectangle with the class name in bold. Operations appear in normal type below the class name. Any data that the class defines comes after the operations. Horizontal lines separate the class name from the operations and the operations from the data.

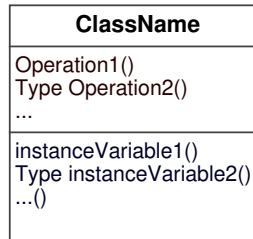


Figure 2.1 Class diagram

The object is considered an instance of a class, thus objects are created by instantiating a class. The process of instantiating a class allocates storage for the object's internal data (made up of instance variables) and associates the operations with these data. A dashed arrowhead line indicates a class that instantiates objects of another class. The arrow points to the class of the instantiated objects.

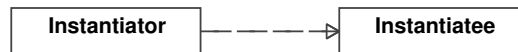


Figure 2.2 Class instantiating objects of another class

Class inheritance allows the definition of new classes in terms of existing classes. Objects that are instances of the subclass contain all data defined by the subclass and its parent classes, and are able to perform all methods defined by this subclass and its parent classes. Overriding refers to a subclass handling a request instead of its parent class, in which case the subclass overrides a method defined by its parent class. The subclass relationship is indicated with a vertical line and a triangle:

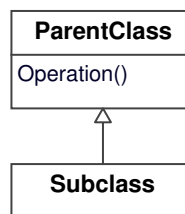


Figure 2.3 Class inheritance – Subclass and parent class relationship

Classes can be concrete or abstract. The main purpose of an **abstract** class is to define a common interface for its subclasses. An abstract class cannot be instantiated, because some or all of its implementations come from operations defined in subclasses.

The operations that an abstract class declares but does not implement are called abstract operations. Classes that are not abstract are called **concrete** classes. To distinguish them from concrete classes the names of abstract classes appear in italics.

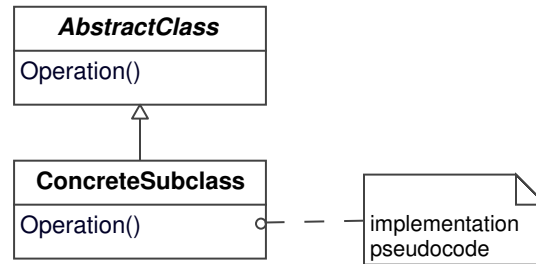


Figure 2.4 Abstract class and concrete subclass diagram

Apart from the difference between an object's class and its type it is also important to understand the difference between class inheritance and interface inheritance. Class inheritance defines an object's implementation in terms of another object's implementation while interface inheritance (or subtyping) describes when an object can be used in place of another. Many of the design patterns depend on this distinction.

Class inheritance is one of the two most common techniques for reusing functionality in object-oriented systems. The other one is object composition. In class inheritance an implementation of a class is defined in terms of the implementation of another class. Reuse by subclassing is called **white-box reuse**, where the term 'white box' refers to visibility, since with inheritance the internals of parent classes are often visible to subclasses. In object composition objects are composed to achieve more complex functionality. Reuse by composition is called **black-box reuse**, because no internal details of objects are visible. Objects appear only as 'black boxes'.

Composition is made into a reuse tool that is as powerful as inheritance with **delegation**, which allows two objects to be involved in handling a request with a receiving object delegating operations to its **delegate**. In the following diagram the Window class is depicted delegating its Area operation to a Rectangle instance. Class Window is not a subclass of Rectangle but it reuses the behavior of Rectangle by keeping a Rectangle instance variable and delegating Rectangle-specific behavior to it. Window must now forward requests to its Rectangle instance explicitly, whereas with class inheritance it would have inherited those operations. A plain arrowhead line indicates that a class keeps a reference to an instance of another class. The reference has an optional name, 'rectangle' in this case. Delegation shows that class inheritance can always be replaced by object composition for code reuse. Though delegation enables the composition of behaviors at run-time, it is very dynamic and thus harder to understand than more static reuse techniques.

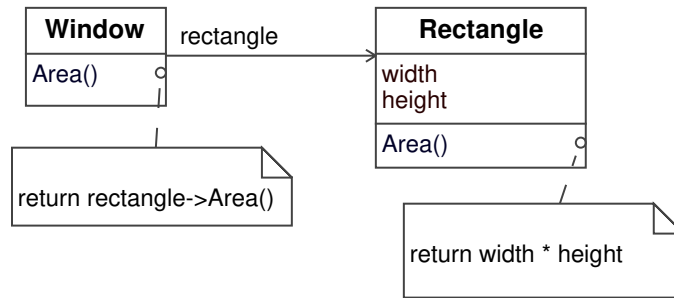


Figure 2.5 Delegation representation

Finally, two important notions are object aggregation and acquaintance. Object **aggregation** is used to denote that an object owns or is responsible for another object, while **acquaintance** implies that the two objects merely know each other, they are able to request operations from each other but are not responsible for each other. Acquaintance is a weaker relationship than aggregation and suggests much looser coupling between objects. Aggregation relationships are fewer but permanent while acquaintances are more frequent and more dynamic. In the design pattern diagrams a plain arrowhead is used to denote acquaintance, while an arrowhead line with a diamond at its base denotes aggregation.



Figure 2.6 Aggregation representation

2.2 Design Patterns used in the implemented COA

The design of an application program should cover three main priorities: internal reuse, maintainability and extension. Design patterns can help address all of these priorities, since they provide dependency reduction. Using design patterns that loosen coupling between classes and reduce algorithmic and representational dependencies can increase the internal reuse. Reduced coupling along with class hierarchy extension can also increase extensibility. When platform dependencies are limited the maintainability of the application is increased.

Reusing legacy systems

It has already been pointed out that the increase in size and complexity of the power systems and the automation of their operation has led to an increase in computer applications used in power systems and maintains the need for new power systems

application and simulation software. Financial constraints on development effort together with frequent changes in the system operation schemes result in limited development time. Under these constraints, reusing previously developed software is very useful.

As mentioned previously there are two fundamental techniques for reusing functionality in object-oriented systems: white-box reuse, which refers to class inheritance, and black-box reuse, which refers to object composition. When a **white-box** approach for reusing legacy systems is adopted the code of the inner structure of the legacy system is studied and reengineered. The size and complexity of the power systems software often results in a white-box approach requiring more effort than completely rebuilding the software. Moreover, the source code or the inner structure of the legacy system may not be available. In these cases a black-box approach can be applied instead. Under a **black-box** approach the legacy system is wrapped with a software layer that hides the unwanted complexity of the old system and exports a modern interface. The two most frequently design patterns used for this purpose are **Adapter** and **Decorator**.

The main purpose of the **Adapter** design pattern, also known as wrapper, is to convert the interface of a class into another interface that clients expect. Using the Adapter allows classes to work together, which could not otherwise because of incompatible interfaces. The Adapter design pattern can be used in the following circumstances:

- there is desire to use an existing class, which has an incompatible interface to the client's class,
- a reusable class needs to be created which should cooperate with classes that do not necessarily have compatible interfaces, and
- several existing subclasses need to be used, but subclassing every one of them so as to adapt their interface is impractical.

The class diagram of an object Adapter design pattern is given in figure 2.7. The participants of the Adapter design pattern class are:

- Target: defines the domain-specific interface that Client uses.
- Client: collaborates with objects conforming to the Target interface.
- Adaptee: defines an existing interface that needs adapting.
- Adapter: adapts the interface of Adaptee to the Target interface.

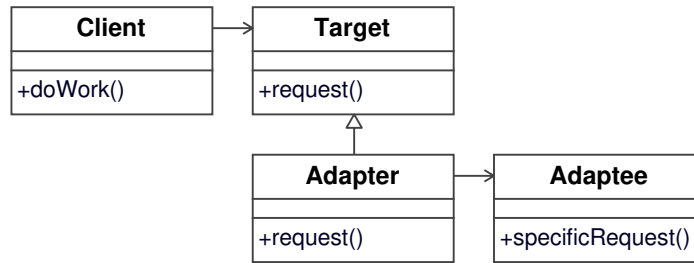


Figure 2.7 Adapter design pattern class diagram

As can be seen from the above structure diagram the Adapter class inherits the Target class, so the main purpose of the Target class is to provide a compatible interface with which the Client class can communicate with the Adapter class. The Client class, through the doWork method, calls the request method on the Adapter class, which in turn calls the corresponding specificRequest method on the Adaptee class. So the purpose of the Adapter is to adapt the interface of the existing object Adaptee to a matching interface to the Client. Using the Adapter class the Client class and the Adaptee class are completely decoupled from each other and legacy software shown as the Adaptee class can be encapsulated.

The basic procedure described previously is that the Client calls methods on an Adapter instance and, in turn, the Adapter calls Adaptee methods that carry out the request. One of the advantages of using an Adapter is that a single Adapter can work with the Adaptee itself and all of its subclasses and thus the Adapter can add functionality to all Adaptees at once. On the other hand, using an Adapter makes the overriding of an Adaptee behavior, that is allowing a subclass of the Adaptee to handle a request instead of the Adaptee, more difficult, since this would require making Adapter refer to the subclass rather than the Adaptee itself.

In the Cascading Outage Analyzer the Adapter design pattern is utilized in order to reuse previously implemented power flow software. As already stated the outage checkers play an essential role in the implemented Cascading Outage Analyzer, since they determine if further outages will occur after some disturbances. To determine the sequence of equilibrium states for checking if further outages will occur, a power flow algorithm is necessary. The implemented outage checkers utilize an independent AC power flow module, PFlow, which has been developed by Commonwealth Associates Inc. and is licensed to The University of Texas. The structure diagram of the Adapter design pattern application to the implemented outage checkers is given in figure 2.8. This Figure is a specialization of figure 2.7, where the Adaptee is the previously implemented power flow module PFlow.

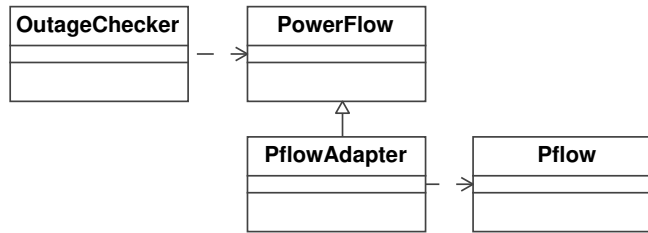


Figure 2.8 Adapter design pattern application class diagram to utilize PFlow

Flexible expandability

The need for flexible expandability is driven foremost by one of the traditional requirements of power systems software, which is high computational efficiency. As described previously there is an ongoing development of new computational models and algorithms. In order to meet the permanent demand on computational performance the existing software must provide the capability of including these new models and algorithms. Hence, the existing and newly developed power systems software must present flexible expandability. This need is growing bigger under the current deregulation of electricity markets, since the deregulation will pose frequent and sometimes big changes in the system and market operation regimes.

A solution to this challenge can again be provided by using design patterns. The relevant design patterns used in the implemented Cascading Outage Analyzer are **Strategy**, **Factory Method**, and **Abstract Factory**.

The main purpose of the **Strategy** design pattern, also known as policy, is to define a family of algorithms, encapsulate each one of them and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

The Strategy pattern can be used in the following circumstances:

- there are many related classes, which differ only in their behavior, i.e. in the methods their objects can perform,
- there is a need for different variations on one algorithm,
- there is desire to avoid exposing complex algorithm-specific data structures to clients,
- there are many behaviors defined by a class, which appear as multiple conditional statements in its methods.

The class diagram of a Strategy design pattern is given in figure 2.9. The participants of the Strategy design pattern class are:

- Strategy: declares an interface common to all supported algorithms, which Context uses to call the algorithm defined by a ConcreteStrategy.
- ConcreteStrategy: implements the algorithm using the Strategy interface.
- Context: is configured with a ConcreteStrategy object, maintains a reference to a Strategy object

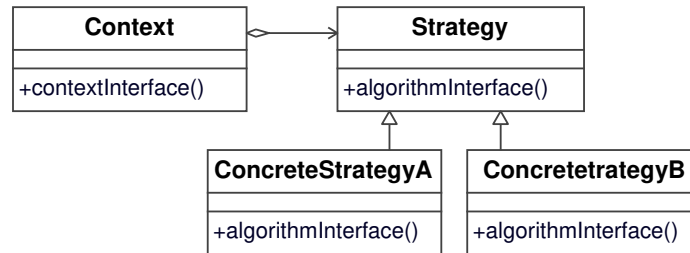


Figure 2.9 Strategy design pattern class diagram

One of the most frequent uses of Strategy pattern is when it is necessary to interchange the algorithms used in an application. This is more than often the case in power systems analysis applications, where there can be various algorithms playing essentially the same role. The choice of the algorithm to be used is affected by many factors, such as the purpose of the algorithm, the specific conditions under which it will be used and the model representation. The Strategy pattern is extremely useful when the purposes or conditions for selecting appropriate algorithms are determined dynamically. As can be seen from the structure diagram in figure 2.9, the contextInterface method of the Context class calls the algorithmInterface method of the Strategy class which will be determined dynamically between those of the ConcreteStrategyA and the ConcreteStrategyB classes.

In the Cascading Outage Analyzer the Strategy design pattern is utilized in order to dynamically swap between two power flow algorithms. The previously implemented Pflow program offers two solution algorithms, the fast decoupled power flow and the full AC power flow. In the Cascading Outage Analyzer application, the specific choice of power flow algorithm is determined at run time as a user input.

The structure diagram of the Strategy design pattern application to the implemented Cascading Outage Analyzer is given in figure 2.10. This Figure is a specialization of figure 2.9, where the PowerFlowStrategy class is applied to select one of the two existing algorithms, ACPowerFlow and DecoupledPowerFlow. The aggregation association between the PowerFlowContext class and the PowerFlowStrategy class enables the runPowerFlow method of the PowerFlowContext class to call the powerFlow method of the PowerFlowStrategy class which will be replaced by either the ACPowerFlow class or the DecoupledPowerFlow class at run time.

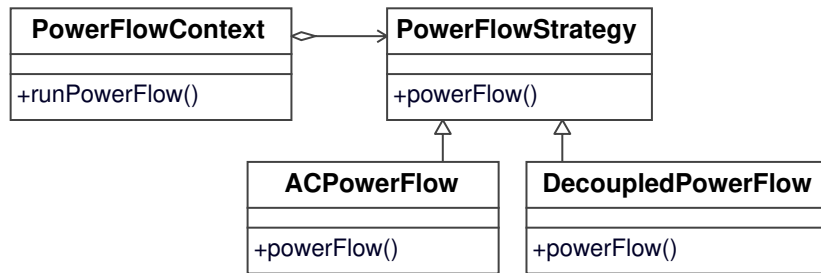


Figure 2.10 Strategy design pattern application class diagram to dynamically swap between the two power flow algorithms, ACPowerFlow and DecoupledPowerFlow.

The main purpose of the **Factory Method** design pattern, also known as virtual constructor, is to define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

The Factory Method pattern can be used in the following circumstances:

- the class of objects that must be created by a class cannot be predicted,
- there is desire to let the subclasses specify the class of objects a class creates,
- there is delegation of a class responsibilities to one or more helper subclasses.

The class diagram of a Factory Method design pattern is given in figure 2.11. The participants of the Factory Method design pattern class are:

- Product: defines the interface of objects that the Factory Method creates.
- Creator: declares the Factory Method, which returns an object of type Product.
- ConcreteCreator: overrides the Factory Method to return an instance of a Product.

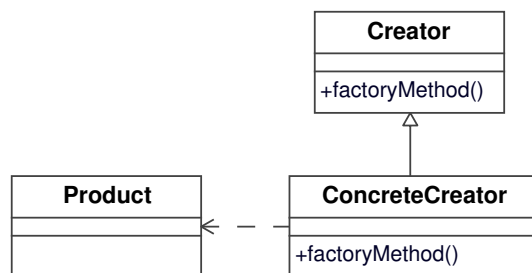


Figure 2.11 Factory Method design pattern class diagram

As described previously constant advances of computational models and algorithms result in a permanent need for the ability to add new algorithms into the existing power system software. The Factory Method design pattern can provide a solution to this challenge, since it allows for adding new algorithms while minimizing the disturbance to the existing code. When there is need to create objects without specifying the exact class

of object that will be created the Factory Method can be applied and the problem is resolved by defining a separate method for creating the objects. The main attribute of this pattern is that it helps to model an interface for creating an object, which at creation time delegates decision of which class to instantiate to its subclasses. By utilizing the Creator class whenever Product objects are created, as shown in figure 2.11, all that is needed to add a new algorithm into the existing code is to modify the Creator and add a new ConcreteCreator class.

The Factory Method design pattern has been used in the implemented Cascading Outage Analyzer in order to provide the ability to add new power flow algorithms to the system, as shown in figure 2.12, which is a specialization of figure 2.13. By utilizing the Factory Method design pattern only the PowerFlowFactory class among the existing code needs to be modified whenever a new algorithm needs to be added to the system, since the other part of code communicates with the PowerFlowStrategy class (product) without knowing the corresponding concrete class. The combined class diagram displaying the Adapter, Strategy and Factory Method design patterns applied to the power flow application in the implemented Cascading Outage Analyzer is given in figure 2.13.

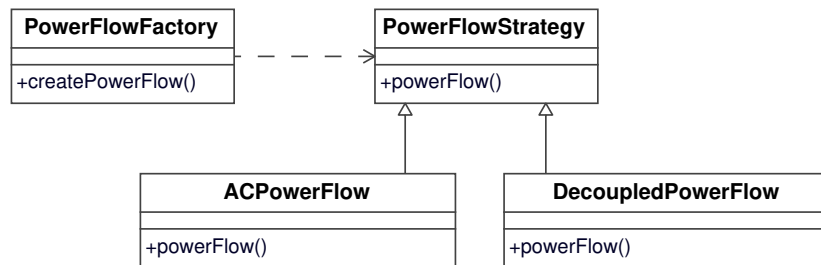


Figure 2.12 Factory Method design pattern application class diagram

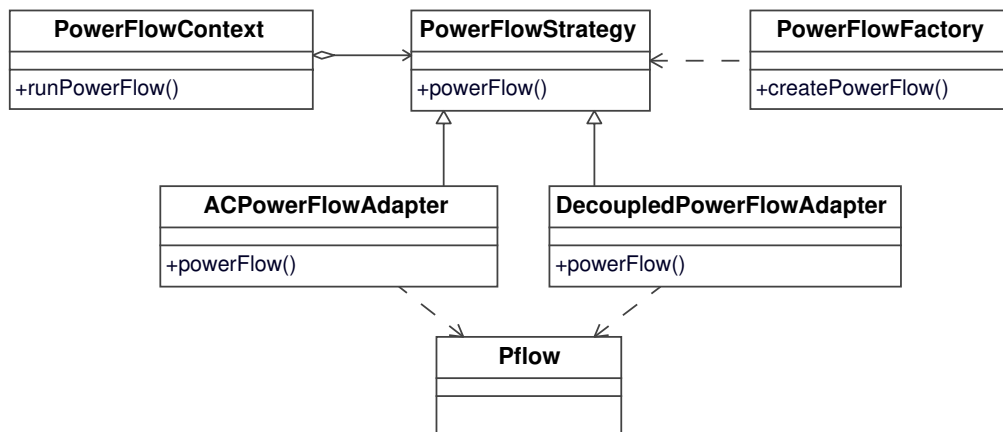


Figure 2.13: Three design pattern (Adapter, Strategy, Factory Method) class diagram utilized for the power flow application

The main purpose of the **Abstract Factory** design pattern, also known as kit, is to provide an interface for creating families of related or dependent objects without specifying their concrete classes.

The Abstract Factory pattern can be used in the following circumstances:

- the way in which products are created, composed and represented should be independent of the system,
- there are multiple families of products and the system should be configured with one of them,
- to enforce the constraint of a family of related product objects being used together,
- there is desire to reveal just the interfaces and not the implementations of a class library of products.

The class diagram of an Abstract Factory design pattern is given in figure 2.14. The participants of the Abstract Factory design pattern class are:

- **AbstractFactory**: declares an interface for methods that create abstract product objects.
- **ConcreteFactory**: implements the methods to create concrete product objects.
- **AbstractProduct**: declares an interface for a type of product object.
- **ConcreteProduct**: defines a product object to be created by the corresponding concrete factory and implements the **AbstractProduct** interface.
- **Client**: uses only interfaces declared by **AbstractFactory** and **AbstractProduct** classes.

As shown in the class diagram of figure 2.14, **AbstractProductA** and **AbstractProductB** objects are created by a **Client** object through either a **ConcreteFactory1** object or a **ConcreteFactory2** object which inherit the **AbstractFactory** class. The **AbstractProductA** and the **AbstractProductB** classes associate the relevant product class group and encapsulate a group of algorithms which are relevant to each other. Thus, the **ProductA1** from the **ConcreteFactory1** and the **productA2** from **ConcreteFactory2** classes are grouped in the **AbstractProductA** class, and the **ProductB1** from **ConcreteFactory1** and the **productB2** from **ConcreteFactory2** classes are grouped in the **AbstractProductB** class.

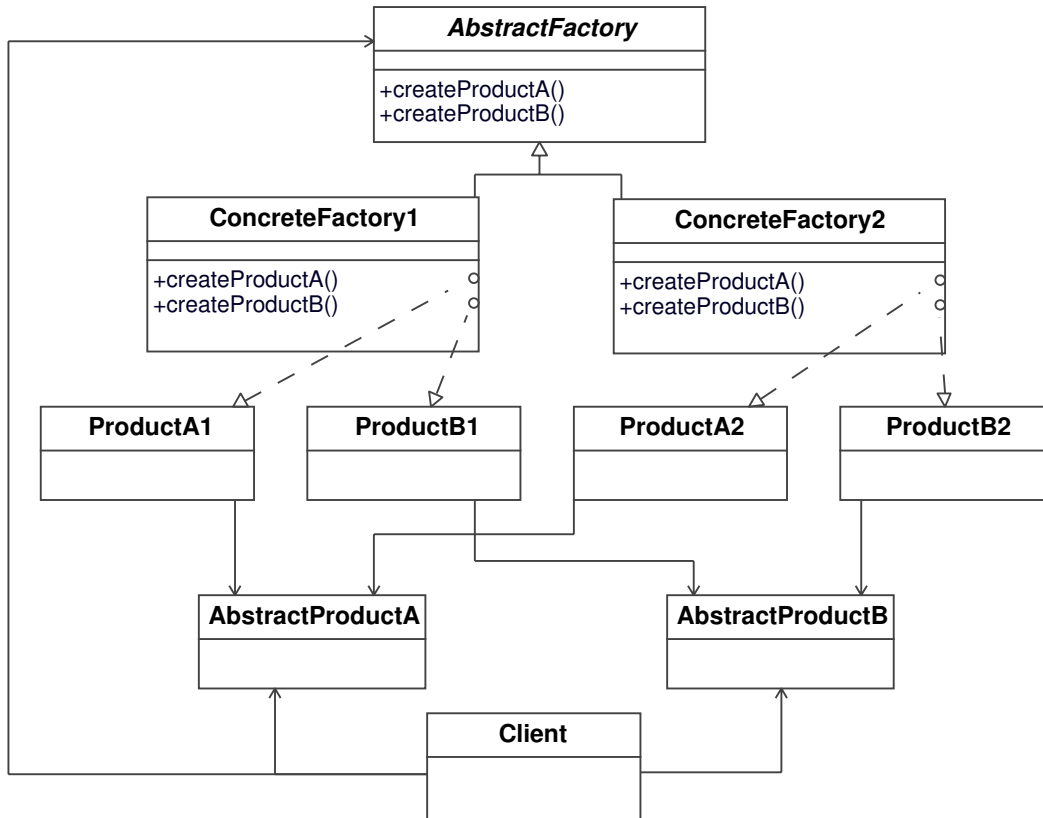


Figure 2.14 Abstract Factory design pattern class diagram

In the implemented Cascading Outage Analyzer there are three outage checkers, LineOverload, UnderVoltage, and Frequency, all of which use the power flow algorithm. The Pflow program offers two power flow algorithms, the fast decoupled power flow and the AC power flow, and the specific power flow algorithm is chosen dynamically, thus each outage checker might, in principle, utilize a different power flow algorithm. To ensure that all the outage checkers use the same power flow algorithm to assess system equilibrium in a consistent manner the Abstract Factory design pattern is applied and the class diagram representation of this is shown in figure 2.15.

As shown in the diagram, the implemented Cascading Outage Analyzer uses either the OutageCheckerConcreteFactory1 object or the OutageCheckerConcreteFactory2 object to create the corresponding sets (indices 1 and 2) of three outage checker objects, namely LineOverloadChecker, UndervoltageChecker and FrequencyChecker. The indices 1 and 2 refer to the two different power flow algorithms. Using the AbstractFactory design pattern prevents mingling of inconsistent outage checker objects with each other, e.g. a combination of LineOverloadChecker1 – UndervoltageChecker2 – FrequencyChecker2 is not allowed.

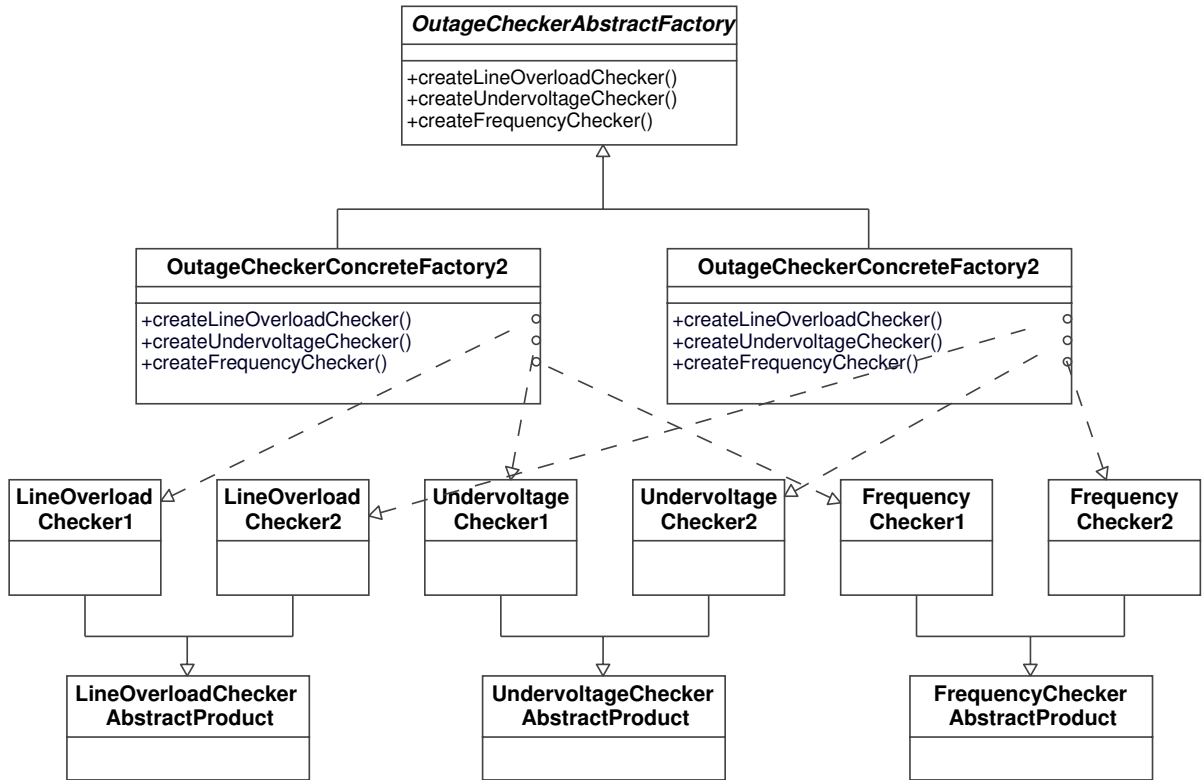


Figure 2.15 Abstract Factory design pattern application class diagram

The Abstract Factory design pattern enables flexible expandability of the program. The addition of new outage checkers into the existing Cascading Outage Analyzer is simplified using the same mechanism as that of the Factory Method design pattern. If a new outage checker for a distance relay needs to be added then the code change for the new outage checker will be limited to the OutageChecker factory part (Abstract and Concrete), by including a createDistanceChecker method.

3 Development of COA software

In this section, the functional specification of COA software will be described on a basis of operation workflow and implementation of cascading outage checkers considering the adopted design patterns.

3.1 Data workflows

The cascading outage analyzer (COA) program is built around the windows-based .NET framework 2.0 in order to support various distributed computing environments. The user interface and the cascading outage analysis algorithm have been implemented by visual basic .NET [4] and visual C# [5] respectively. To provide flexibility and extensibility, the application design follows the object-oriented design principle using Microsoft Visual Studio 2005.

The following figure shows the data flow of COA software. After setting input data in the system database, cascading outage checkers are activated according to the determined initial disturbance. Simulation results will be displayed by message trees based on graphic user interface (GUI).

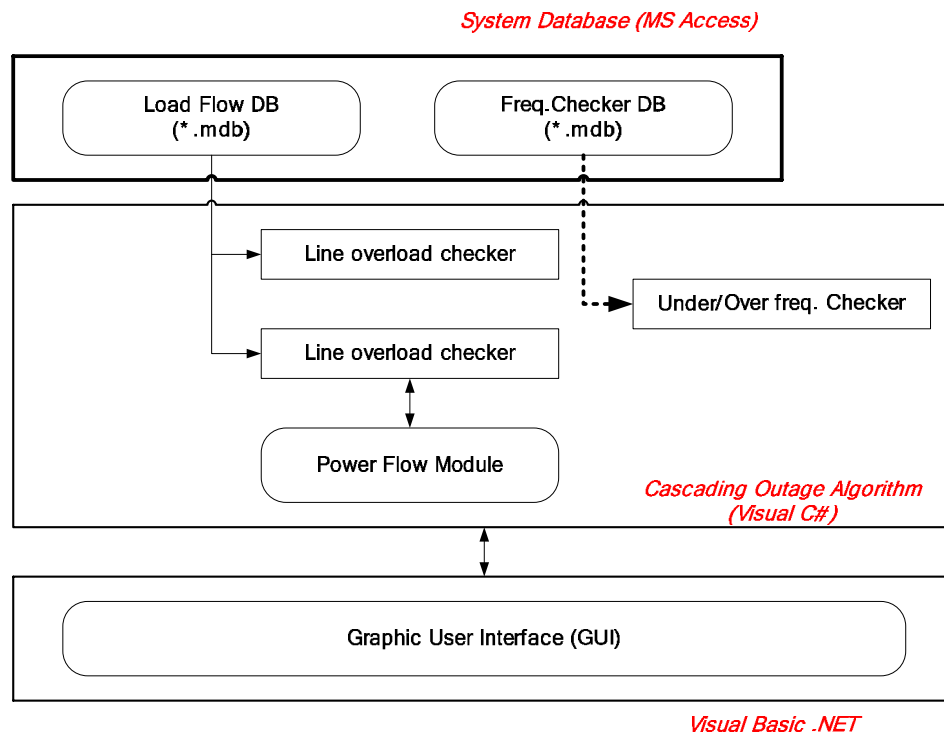


Figure 3.1 Data flows of COA software

From figure 3.1, the system database has two input database: **Input database** (**'Datainput.mdb'**) for power flow calculations, and **Simulation option database** (**'COChecker'**) for implementing cascading outage scenarios, which are stored in Microsoft Access format. MS Access program (MS access 2003 higher) should be installed to user's PC. These input files should be located at the following folder (directory):

C:\project\Dataset

In order to run the power flow module, which has been provided by Commonwealth Associates Inc., the user must create a temporary file (temp.mdb) by renaming the input load flow file to the following folder (directory).

C:\project\implementation\source\temp.mdb

If users want to make new scenarios (cases), we recommend that users should modify the existing case files (Datainput.mdb and COChecker.mdb). Although 'Datainput.mdb' file has many tables, typical modifications involve four main tables relevant to power flow calculation:

- Bus
- Generator
- Load
- Line

The power flow module determines sequential equilibrium states for checking further outages. The cascading outage analysis program utilizes an independent AC power flow module, 'PFlow', which has been developed by Commonwealth Associates Inc. and licensed to The University of Texas.

Under and over frequency checker module obtains information for calculating time duration of relay from Frequency checker database (COChecker.mdb). This database includes some tables as follows.

- *FOChecker* table; It includes frequency checker information. This table has the following columns:
 - Bus: Bus number of power system
 - BusType: Type of bus (1: Generation bus and 2: Load bus)
 - Threshold: Input value for threshold frequency
 - Delay: Input value for setting time duration of relay

- *FOChecker_org* table; It has structure that corresponds to the ‘FOChecker’ table and it also acts as a source table during calculation.
- *CTDOutput* table; Simulation results of frequency checker will be recorded to this table. This table has the following columns:
 - BusNum: Input value for bus number of network configuration
 - CTD_Under: Output value of the calculated Time duration for Under-frequency
 - CTD_Over: Output value of the calculated Time duration for Over-frequency

In order to reuse this previously implemented module and provide flexibility to switch to or add different power flow modules and determine a specific module to run at run-time, three design patterns, the Adapter, Strategy, and Factory Method design patterns, have been adopted as described in the previous section.

3.2 Operational algorithm

After initiating the disturbance events, the developed checkers will be operated as user pre-defined the orders and types of three outage checkers. The following figure shows the operating logic (algorithm) of the developed Cascading Outage Analyzer (COA) software.

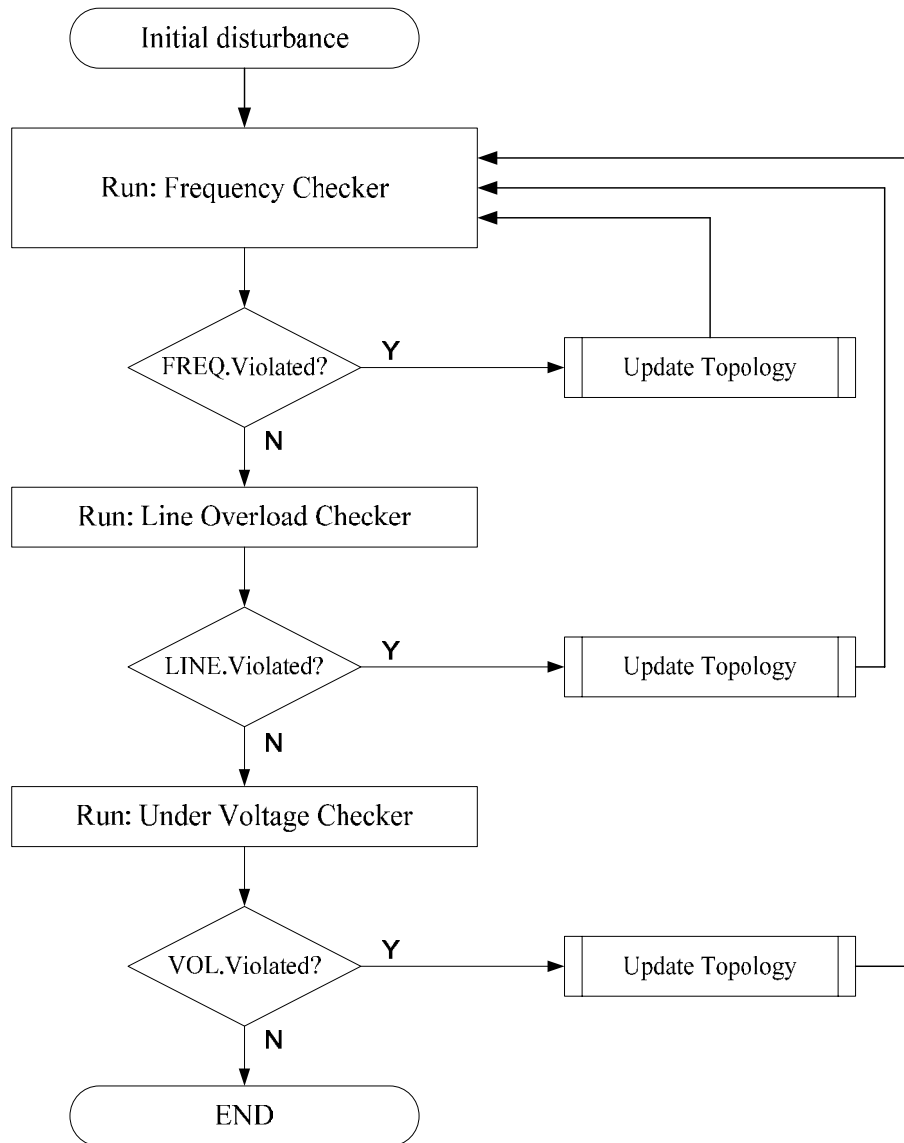


Figure 3.2 Operation logic of Cascading Outage Analyzer

Initial Disturbance

In order to begin cascading outage simulation, we need to specify the initial disturbances to be investigated for their progression to cascading outages. Line outage type can be considered as an initial disturbance and the extension to other types of outages can be considered including not only line outages but also generator outages, bus outages, and substation outages. Initial disturbances that could occur in simulating cascading outages include:

- Line outage (implemented in current COA algorithm),
- Multiple initial disturbances (future work),
- Additional initial disturbance types (future work):
 - Generator outage
 - Substation outage
 - Load bus trip

Outage Checkers & Update Topology

Following the initial disturbance, the outage checkers such as ‘Frequency Checker’, ‘Line Overload Checker’ and ‘Under Voltage Checker’, determine the status of the resulting operating state, or ‘equilibrium’, which would result in protection equipment removing more elements from the system and therefore potentially precipitating further outages.

If there are several protection actions (or violations) identified by the outage checkers, then timing information from the outage checkers will determine which element would be, in fact, first disconnected. This element is removed from the power flow model using the ‘Update topology’ blocks. As shown in figure 3.1, the process then repeats until either a complete system blackout occurs (indicated by failure to solve remaining system) or no more protection actions (no violations) are predicted to occur.

3.3 Implementation of outage checkers

With the initial disturbance determined, we need to set up the type and order of cascading outage checkers that will be considered. An ‘outage checker’ is a module that tests the state of the power system to see if additional outages will be precipitated due to a particular protection criterion. The developed outage analysis tool is designed to have three types of outage checkers; line overload, under-voltage checkers, and over/under frequency checkers that model the behavior of line overload, under-voltage protections, and system frequency, respectively.

3.3.1 Line overload and under voltage checkers

The ‘Line Overload Checker’ and the ‘Under Voltage Checker’ use the power flow as a basic simulation engine and users can select load flow program, either Full AC power flow or Decoupled power flow, which are provided by Commonwealth Associates Inc [6].

In order to choose the specific power flow module from multiple power flow modules dynamically in a consistent manner and to flexibly expand the outage checkers, the Abstract Factory design pattern was utilized as mentioned in section 2.

3.3.2 Under/Over frequency checker

In addition to the steady-state analysis based approach, an outage checker is developed for protection against system over/under-frequency events. The following figure shows the data flow of frequency checker.

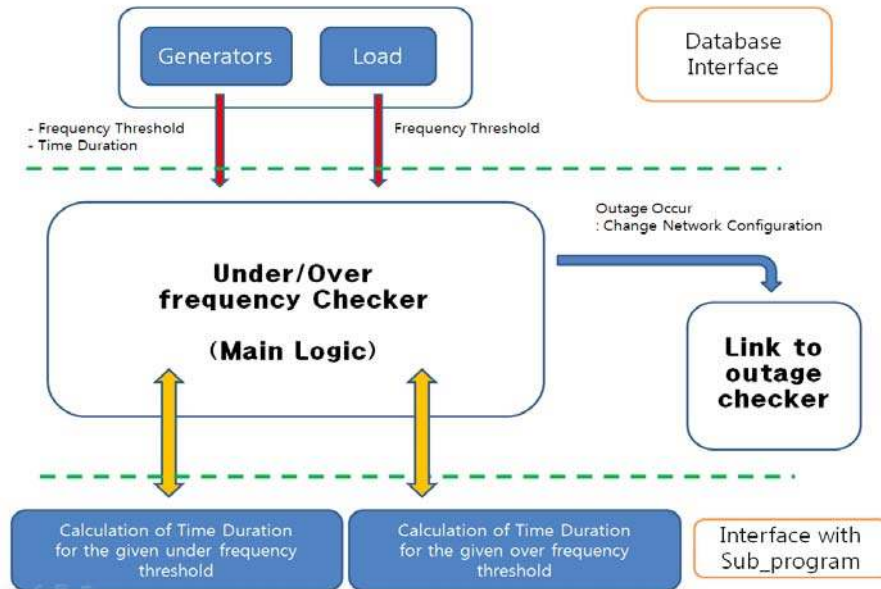


Figure 3.3 Data flow of frequency checker

From figure 3.3, the main logic of the frequency checker inquires about threshold frequencies and time duration for generator and load frequency relays from the database interface.

Using these data, the engine of the main logic block calculates the time duration for the given under and over frequency threshold interfacing sub-program (time duration calculation module). If the violations for under or over frequency checker occur then the checker moves to another checker, either the Line overloading checker or Under-Over voltage checker.

The following figure shows the algorithm of frequency checker.

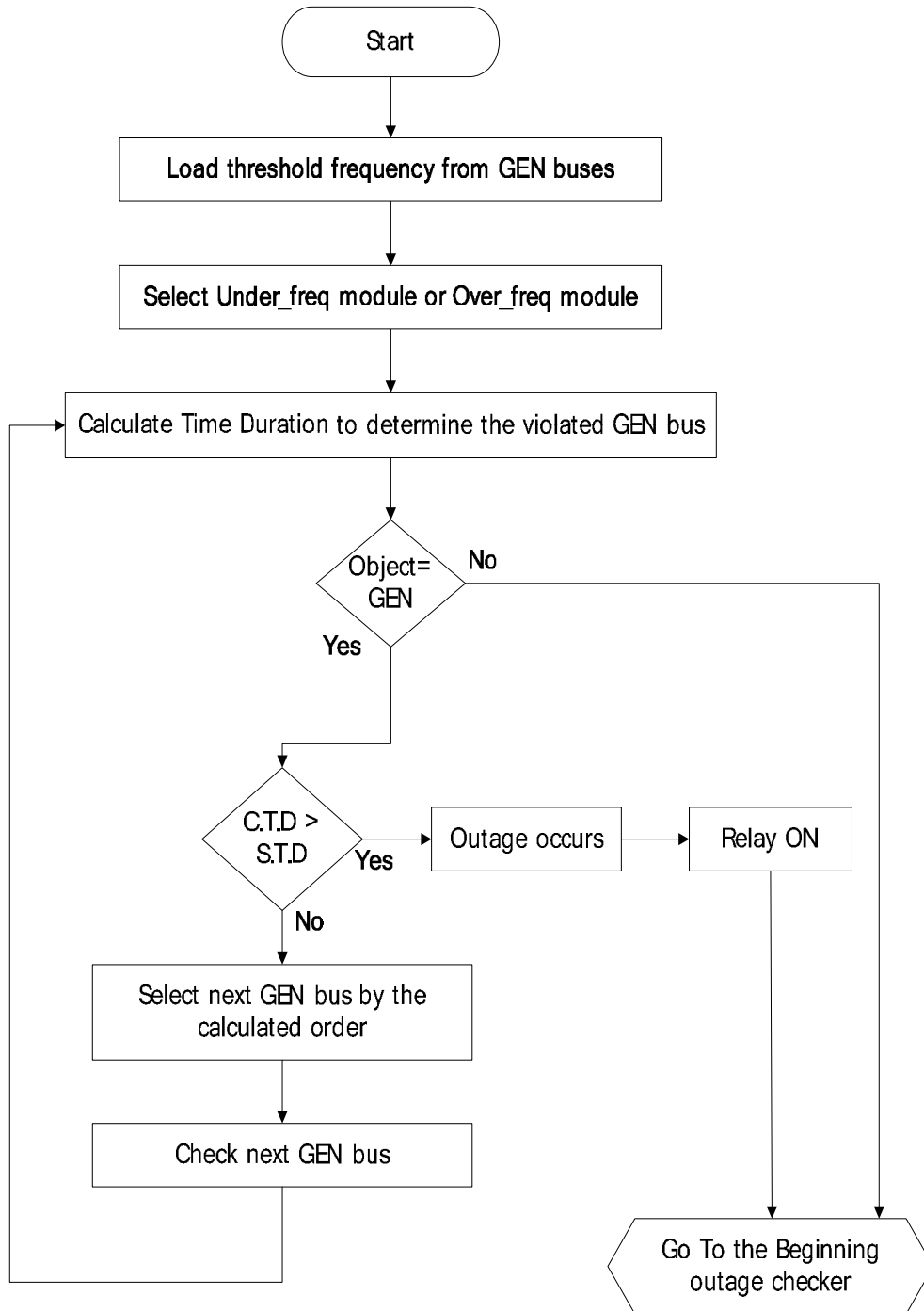


Figure 3.4 Frequency checker algorithm

Step1: Load threshold frequency from GEN buses

From system database, 'FOChecker' table from COChecker.mdb has threshold frequency information. The following figure illustrates the example of 'FOChecker' table.



ID	FOID	Bus	BusType	LevelFO	Threshold	Delay
1	1	1	1	1	59.9869	6
2	1	2	1	2	59.9879	6
3	1	3	1	3	59.9889	6
4	1	4	2	4	59.9879	7
5	2	5	2	1	59.9879	7
6	2	6	2	2	59.9879	7
7	2	7	2	3	59.9879	7
8	2	8	2	4	59.9879	7
9	2	9	2	1	59.9879	7

Figure 3.5 FOChekcer table

After loading the threshold frequencies, the frequency checker module sorts the generation buses on the basis of frequency threshold values from high to low and determines under frequency and over frequency with system frequency criteria (60.0 Hz).

Step 2: Calculate time duration to determine the violated GEN bus

In this step, frequency checker module calculates times the duration for the selected bus in order to determine whether the Calculated Time Duration (CTD) of frequency violation exceeds the Set Time Duration (STD). The algorithm for calculating time duration will be described in the next section.

Step 3: Move to other checkers

If the violated bus is detected In the previous step (step 2), the system topology would be updated and a new round of outage checking would occur.

In the following subsections, details about the algorithm will be described.

3.2.2.1 Model of the Frequency Outage

(1) Frequency Function

For frequency change model, we use the **System Frequency Response (SFR)** model introduced in ‘A Low-Order System Frequency Response Model’ by Anderson and Mirheydar (1990) [7]. The idea of uniform or average frequency is the basic concept of representing SFR model, where synchronizing oscillations between generators are filtered out, but the average frequency behavior is retained.

The example of synchronizing oscillations is illustrated in figure 3.6, taken from the Florida simulations of reference. We seek to average these individual machine responses with a smooth curve that can be used to represent the average frequency for the system. As a result, the solid curve shows the trend of overall system frequency in figure 3.6.

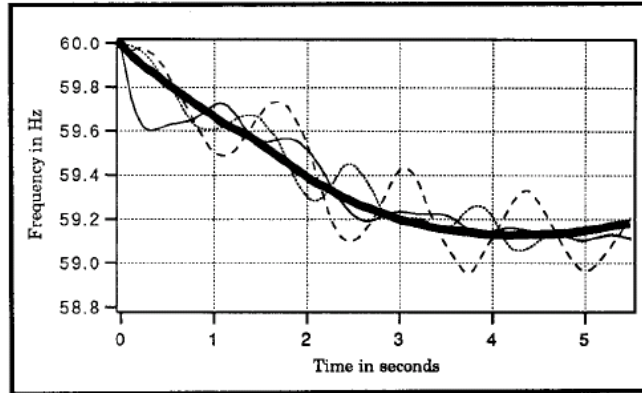


Figure 3.6 Simplified SFR Model with disturbance input¹

The basic SFR model averages the machine dynamic behavior in a large system into an equivalent single machine and it is a representation of only the average system dynamics, while ignoring the inter-machine oscillations shown in figure 3.6.

According to this model, we have the frequency change function in frequency domain

$$\Delta\omega = \left(\frac{R\omega_n^2}{DR + K_m} \right) \left(\frac{(1 + T_R s)P_{step}}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)} \right)$$

¹ A. N. Darlington, "Response of under-frequency Relays on the Peninsular Florida Electric System for Loss of Generation," A paper presented at the Georgia Institute of Technology Relay Conference, May 4, 1978

where,

- $\Delta\omega$ = Incremental speed, per unit
- T_R = Reheat time constant, seconds
- P_{step} = Disturbance magnitude in per unit (based on the system voltage base SSB)
- D = Damping Factor
- R = Governors droop
- K_m = Mechanical Power Gain Factor
- FH = Fraction of total power generated by the HP turbine

After transforming this into time domain and simplifying some parameters, we have

$$\Delta\omega(t) = \frac{RP_{step}}{DR + K_m} \left[1 + \alpha e^{-\zeta\omega_n t} \sin(\omega_r t + \phi) \right]$$

where,

- $\omega_n^2 = \frac{DR + K_m}{2HRT_R}$
- $\zeta = \left(\frac{2HR + (DR + K_m F_H)T_R}{2(DR + K_m)} \right) \omega_n$
- $\alpha = \sqrt{\frac{1 - 2T_R\zeta\omega_n + T_R^2\omega_n^2}{1 - \zeta^2}}$
- $\omega_r = \omega_n \sqrt{1 - \zeta^2}$
- $\phi = \phi_1 - \phi_2 = \tan^{-1} \left(\frac{\omega_r T_R}{1 - \zeta\omega_n T_R} \right) - \tan^{-1} \left(\frac{\sqrt{1 - \zeta^2}}{-\zeta} \right)$

We have an example of frequency change curve as in figure 3.7 if we use the typical parameters below from the paper ‘A Low-Order System Frequency Response Model’ by Anderson and Mirheydar (1990).

- R = 0.05
- H = 4.0s
- Km = 0.95
- FH = 0.3
- TR = 8.0s
- D = 1.0

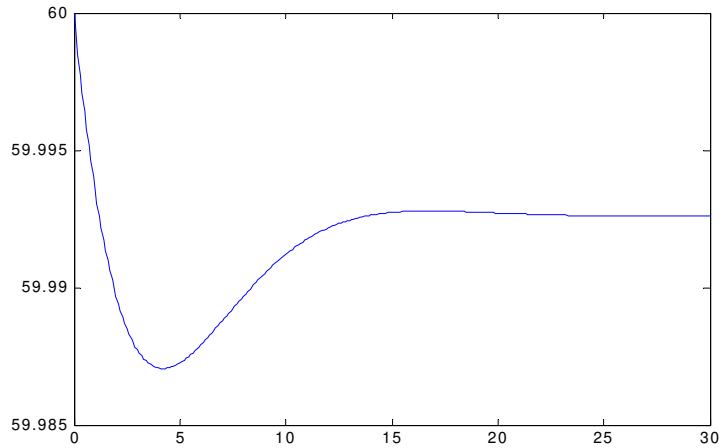


Figure 3.7 Example of Under Frequency Curve

(2) Definition of a Frequency Outage

In this report, we will use the frequency outage standard from ERCOT as the definition of a frequency outage. According to this standard, when the system frequency is out of a certain range for a certain amount of time, we define this as a frequency outage. Table 3.1 below demonstrates the characteristic of the under frequency outages.

Table 3.1 Standard of a Frequency Outage

Frequency Range (Hz)	Time Delay (Sec)
Above 59.4	Infinite
58.4 to 59.4	270
58.0 to 58.4	30
57.5 to 58.0	2
Below 57.5	0

In table 3.1, the left column is the frequency range, and the right column is the time length threshold that the frequency in the left column must last in order to initiate a trip. For example, if the frequency drops under 59.0 Hz, which falls into the second frequency range, for more than 270 seconds, an under-frequency trip will be initiated in the system. In the extreme case, there is no frequency outage in the system as long as the frequency is higher than 59.4 Hz, like 59.6, no matter how long the 59.6 value will last. In the other

extreme case, as soon as the frequency drops below than 57.5, a frequency outage will be defined immediately.

3.2.2.2 Description of the Algorithm

(1) Output and Input of the frequency outage checker

According to the definition of a frequency outage, we need to know two parameters to define an under-frequency trip: lowest frequency value ‘min_f’, and the time that the frequency is below the provided frequency threshold ‘time_delay’. If min_f is lower than 57.5, or time_delay is longer than the time delay in table 3.1, we will decide there is a frequency trip in the system. Therefore, we have the output: min_f, and time_delay.

The input of the checker comes from the database of the analysis tool. We need to have disturbance power P_{step} in the frequency response function. And also, we need to provide the frequency threshold to get time_delay when the frequency is below the threshold. We call this frequency threshold ‘f_given’. So the input parameters are disturbance power ‘Pstep’, and frequency threshold ‘f_given’. The output parameters are lowest frequency value ‘min_f’, and the below-the-threshold time ‘time_delay’.

(2) Main task of algorithm

As described in the last section, we are given a frequency response function $f(t) = \frac{RP_{step}}{(DR + K_m)2\pi} [1 + \alpha e^{-\zeta\omega_n t} \sin(\omega_r t + \phi)] + 60$ and a horizontal curve $f(t) = f_given$. What we need to calculate are min_f and time_delay, as shown in figure 3.8 below.

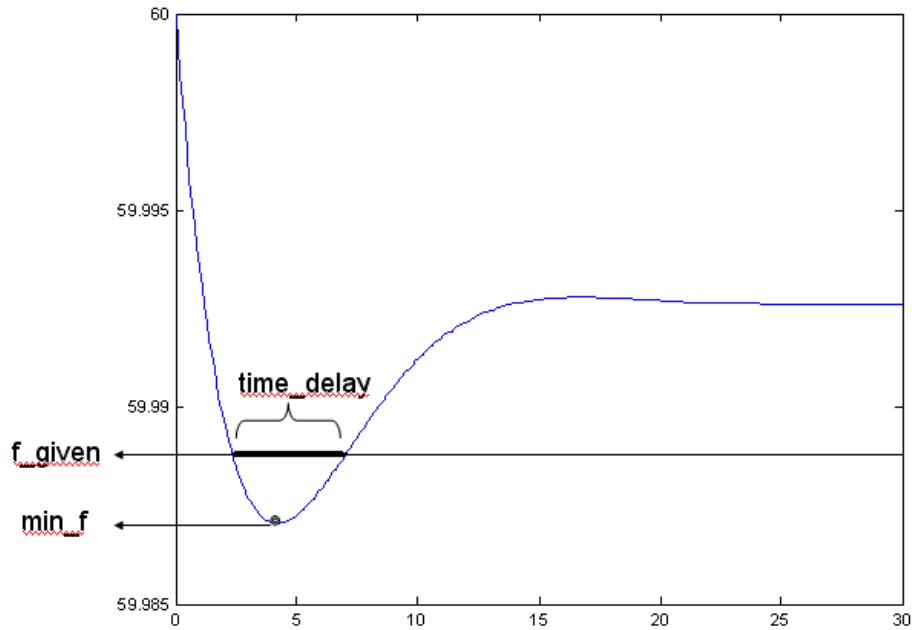


Figure 3.8 Parameters to calculate in the Frequency Function

The lowest point of frequency value should be the first local minimizer of the frequency curve, since the curve is a damped sine wave and the following local minimizers will involve frequencies that are closer to 60 Hz.

In an optimization problem like calculating \min_f , we can apply the first order condition to find the local minimizer. That is to say, if the Jacobian of the function at one point is zero, then this point is a local minimizer. In this problem, we need to find a point t which satisfies the equation $\nabla f(t) = 0$ in the first cycle of the curve. Now we have converted the optimization problem into solving the equation

$$\nabla f(t) = 0$$

To calculate time_delay , we can first find the time when the frequency first drops under the frequency threshold 't1', and the time when the frequency first rises up to the threshold again 't2'. Apparently, t_1 and t_2 are two intersections of two functions:

$$f(t) = \frac{RP_{step}}{(DR + K_m)2\pi} \left[1 + \alpha e^{-\zeta\omega_n t} \sin(\omega_r t + \phi) \right] + 60$$

$$f(t) = f_given$$

To calculate the intersection of two functions, we can simply solve the equation

$$\frac{RP_{step}}{(DR + K_m)2\pi} \left[1 + \alpha e^{-\zeta\omega_n t} \sin(\omega_n t + \phi) \right] + 60 = f_{given}$$

Hence, we can conclude that to calculate \min_f and time_delay , our main task is to solve equations.

3.2.2.3 Descriptions of the Steps of Algorithm

(1) Main steps of the algorithm

Before writing the code, we need to work out the main steps of the algorithm. There are two parameters that need to be calculated: lowest frequency value 'min_f', and the below-the-threshold time 'time_delay'. Below is the flow chart of the main steps of the algorithm.

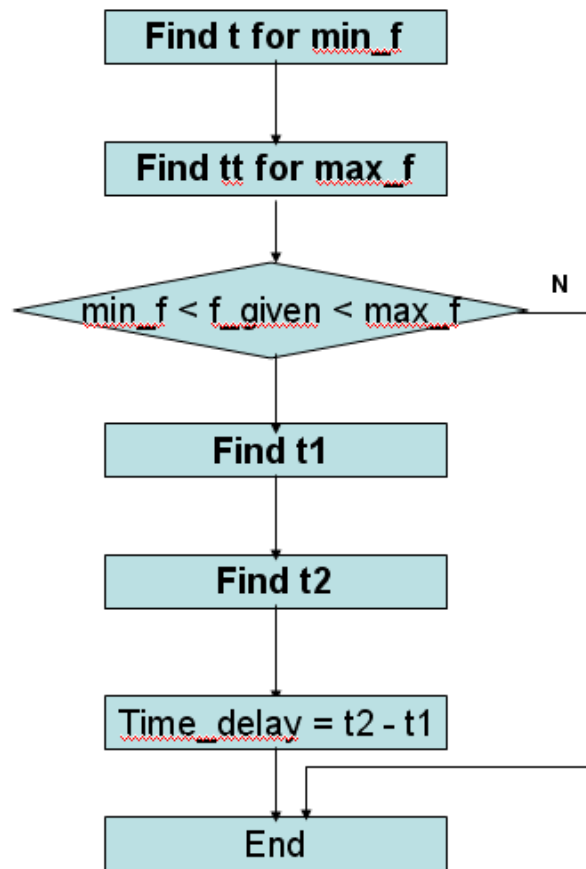


Figure 3.9 Flow Chart of the Code

Following figure 3.9 above, we can see that the first step is to find point t and \min_f , and we can calculate them using Newton-Raphson Method, which will be presented in detail in the next section.

Then, before we calculate the two intersection points t_1 and t_2 , we need to make sure that there will actually be more than one intersection of the frequency function and the horizontal function. So we find the first local maximizer tt after t , and the frequency value of this point. As shown in figure 3.10, between \min_f and \max_f is the 'two-intersections

range' for the given frequency threshold f_{given} .

If the f_{given} is between min_f and max_f , it is obvious that we can find two intersections between t and tt point. But if f_{given} is larger than max_f , there will only be one intersection $t1$. If f_{given} is smaller than min_f , there will be no intersection. So only when $min_f < f_{given} < max_f$ can we get an actual value of $time_delay$.

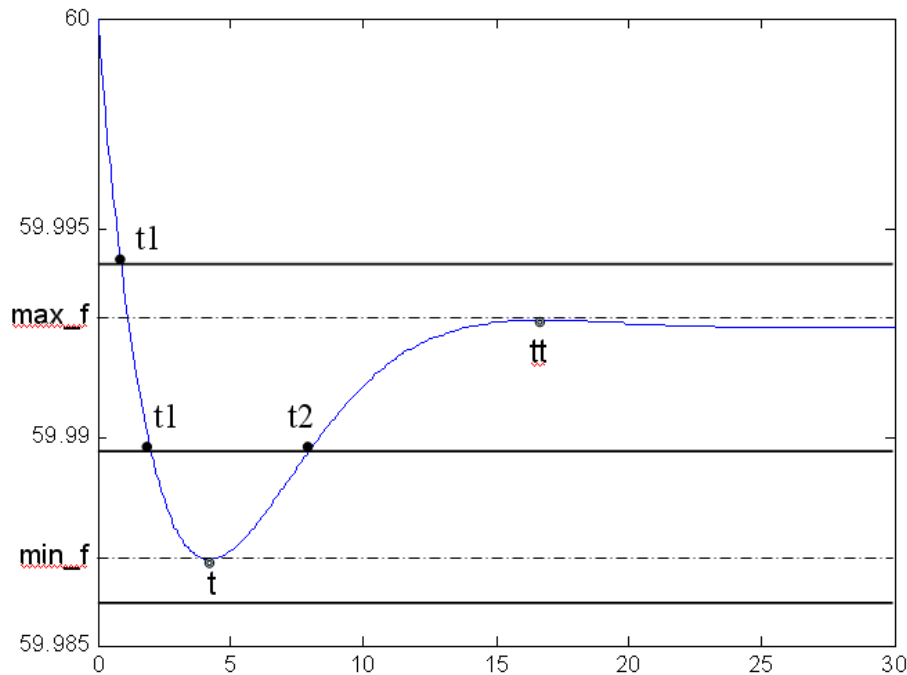


Figure 3.10 'Two-intersections range' for f_{given}

So from the flow chart, if the given frequency threshold is in the range from min_f to max_f , we can find $t1$ and $t2$, and $time_delay$, which is $t2-t1$.

(2) Find t for min_f

According to the flow chart of the code in Figure 3.9, the first step is to find t and min_f . Since we already have the frequency response function, as long as we have t , we can calculate min_f immediately after inserting t into the frequency function. So we need to calculate the first local minimizer t first. We know that to seek t , we just have to take the derivative of the frequency function and set to zero. The derivative of the frequency function is

$$f'(t) = \frac{RP_{step} \alpha}{(DR + K_m) 2\pi} e^{-\zeta \omega_n t} [\omega_r \cos(\omega_r t + \phi) - \zeta \omega_n \sin(\omega_r t + \phi)]$$

Now our problem is solve the equation

$$\frac{RP_{step} \alpha}{(DR + K_m) 2\pi} e^{-\zeta \omega_n t} [\omega_r \cos(\omega_r t + \phi) - \zeta \omega_n \sin(\omega_r t + \phi)] = 0$$

and solution should be the first minimizer from $t = 0$.

We can use Newton-Raphson method and Armijo rule to solve equations. Figure 3.11 below is the flow chart to solve this problem.

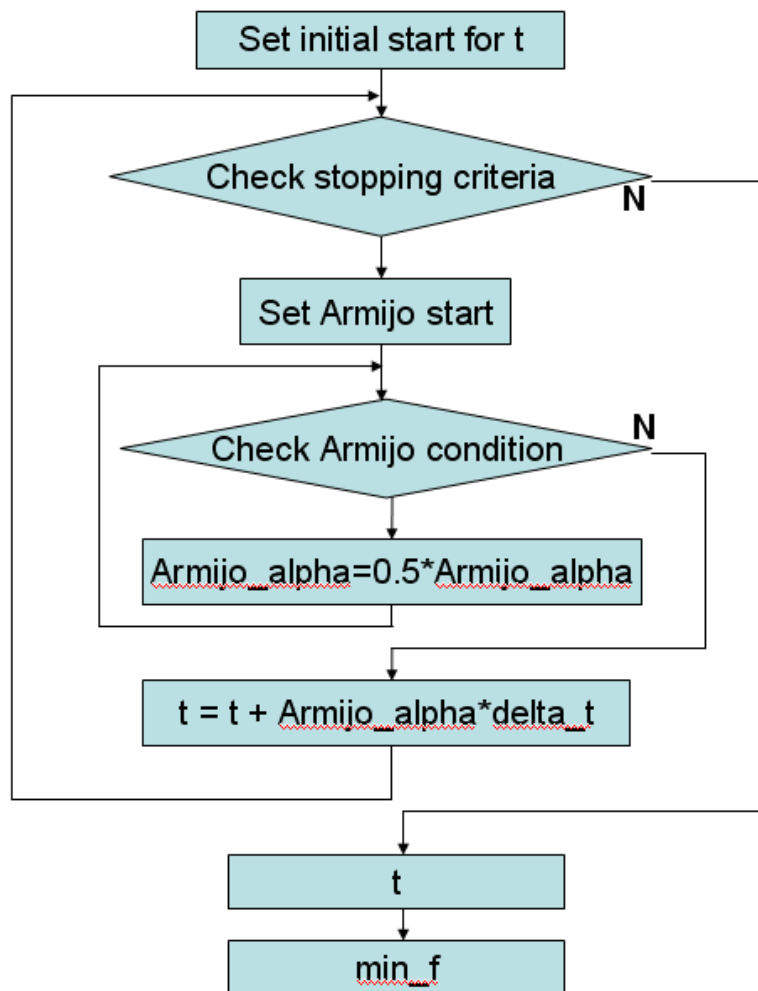


Figure 3.11 Flow Chart of Calculating min_f

The first step is to ‘set initial start for t’. We need to find a point which is closer to the solution, not only to decrease the number of iterations to make the checker more efficient,

also to make sure the solution is the first minimizer from $t = 0$. Since the frequency response function is a damped sine curve, the local minimizer of the function should be close to the local minimizer of the sine curve.

Furthermore, we will use Armijo rule to make sure each iteration will only move closer to the solution. So by choosing the local minimizer of the sine curve as an initial start for t , we can exclude the possibility that the t we find is a minimizer of other cycles of the curve. Therefore, we calculate the first minimizer of $-\sin(\omega_r t + \phi)$, which is

$$\omega_r t + \phi = \frac{\pi}{2}, \text{ and we have the initial start } t = \frac{(\frac{\pi}{2} - \phi)}{\omega_r}.$$

The second step is to ‘check stopping criteria’ to see if the checker has found the solution. If the slope of some point of the frequency curve is very small, we can stop the iteration and call this point as solution to finding the time of minimum frequency.

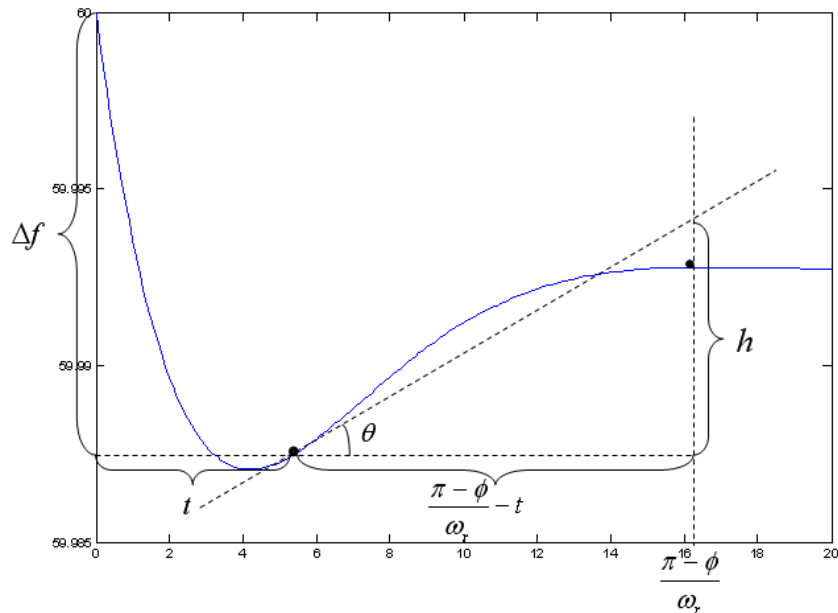


Figure 3.12 Parameters in Stopping Criteria for t

In this problem, we define the slope is ‘very small’ when h , the projection of the tangent angle on the vertical line at next maximizer, is smaller than $0.01 \Delta f$, as shown in figure 3.12 above, where Δf is the absolute value of the frequency drop at time t . $\Delta f = \|f(t) - 60\|$. That is to say if $h \geq 0.01 \Delta f$, then the checker will go to the next iteration, until $h < 0.01 \Delta f$.

As presented in figure 3.12, here is how we get h . Firstly, since the curve is the damped sine wave, we can approximately find the first maximizer t after t . By setting $\omega_r t + \phi = \pi$, we can get the approximate maximizer t , which is $\frac{\pi - \phi}{\omega_r}$. Secondly, we compare the distance from t to zero and the distance from t to $\frac{\pi - \phi}{\omega_r}$. Then we choose whichever is the larger and multiply by the slope of point t . In the case of figure 3.12, for example, the distance from t to $\frac{\pi - \phi}{\omega_r}$ is larger than the distance from t to zero. $\max(t, \frac{\pi - \phi}{\omega_r} - t) = \frac{\pi - \phi}{\omega_r} - t$ Then we will get h by multiplying the larger distance by the slope of point t , which is $\tan(\theta)$.

$$h = \max(t, \frac{\pi - \phi}{\omega_r} - t) \tan(\theta)$$

$$h = (\frac{\pi - \phi}{\omega_r} - t) \tan(\theta)$$

And we can tell that $\tan(\theta)$ is the derivative of the frequency function at point t , which is $f'(t)$.

So, we have the expression of h

$$h = (\frac{\pi - \phi}{\omega_r} - t) f'(t)$$

where,

$$f'(t) = \frac{RP_{step} \alpha}{(DR + K_m) 2\pi} e^{-\zeta \omega_n t} [\omega_r \cos(\omega_r t + \phi) - \zeta \omega_n \sin(\omega_r t + \phi)]$$

As defined above, if $h \geq 0.01 \Delta f$, then the checker will go to the next iteration, until $h < 0.01 \Delta f$. So according to the flow chart, if $(\frac{\pi - \phi}{\omega_r} - t) f'(t) < 0.01 \|f(t) - 60\|$, we check Armijo step size rule before we update the solution and go to the next iteration.

We begin checking Armijo rule by ‘setting Armijo start’. As discussed in 5.3.2, we start from setting $\alpha^{(v)} = 1$. If $\|g(x^{(v)} + \alpha^{(v)}\Delta x^{(v)})\| \geq (1 - \delta\alpha^{(v)})\|g(x^{(v)})\|$, we multiply the step size $\alpha^{(v)}$ by 0.5, and we check again until $\|g(x^{(v)} + \alpha^{(v)}\Delta x^{(v)})\| < (1 - \delta\alpha^{(v)})\|g(x^{(v)})\|$ is satisfied.

After checking Armijo rule, and the step size being decided, we can move to update the iteration by $t = t + \alpha^{(v)}\Delta t$. Then we go back to the top of the flow chart and check the stopping criteria again.

After the iteration stops, we can get the solution of t. Then by plugging it into the frequency response function, we will have the minimum frequency value min_f.

(3) Find tt for max_f

To find the first local maximizer tt after t, we can use the same method as in 6.2.1. We have the derivative of the frequency function

$$f'(t) = \frac{RP_{step}\alpha}{(DR + K_m)2\pi} e^{-\zeta\omega_n t} [\omega_r \cos(\omega_r t + \phi) - \zeta\omega_n \sin(\omega_r t + \phi)]$$

and set it to zero.

Now we need to solve the equation

$$\frac{RP_{step}\alpha}{(DR + K_m)2\pi} e^{-\zeta\omega_n t} [\omega_r \cos(\omega_r t + \phi) - \zeta\omega_n \sin(\omega_r t + \phi)] = 0$$

and the solution should be the first maximizer after the first minimizer t.

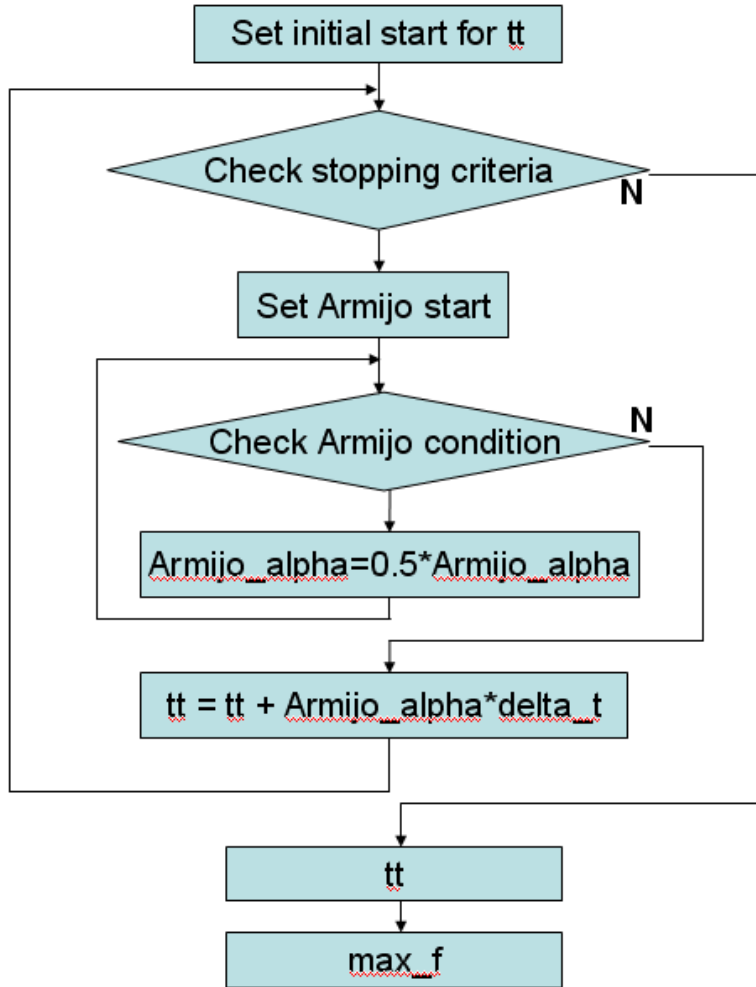


Figure 3.13 Flow Chart of Calculating tt and max_f

Similarly, we will use Newton-Raphson method and Armijo rule to solve this equation. figure 3.13 is the flow chart, which is very similar to figure 3.11, except that the initial start and stopping criteria are different. Two problems are seeking the solutions in different ranges with the same equation.

According to the flow chart in figure 3.13, we start with setting initial start for tt, similar to setting initial start for t. We already know that the local maximizer of the frequency curve should be very close to the local maximizer of the sine wave function $-\sin(\omega_r t + \phi)$. So we can use the first maximizer after t of the sine function as the initial start for tt. The Armijo rule can ensure each iteration moves closer to the solution. So by choosing the first maximizer after t of the sine function, we can exclude the possibility that the tt we find is some maximizer of other cycles of the curve. Therefore, we calculate the first maximizer after t in the first cycle of $-\sin(\omega_r t + \phi)$, which is $\omega_r t + \phi = \frac{3\pi}{2}$, and

we have the initial start $t = \frac{3(\frac{\pi}{2} - \phi)}{2\omega_r}$.

As in flow chart of figure 3.13, we should then check for stopping criteria for tt . We use the similar criteria, which is if the slope of some point of the frequency curve is very small, we can stop the iteration and call this point as solution.

We define the slope is ‘very small’ when h , the projection of the tangent angle on the vertical line at next minimizer, is smaller than $0.01\Delta f$, as shown in figure 3.14 below, where Δf is the absolute value of the frequency drop at time tt . $\Delta f = \|f(tt) - 60\|$. That is to say if $h \geq 0.01\Delta f$, then the checker will go to the next iteration, until $h < 0.01\Delta f$.

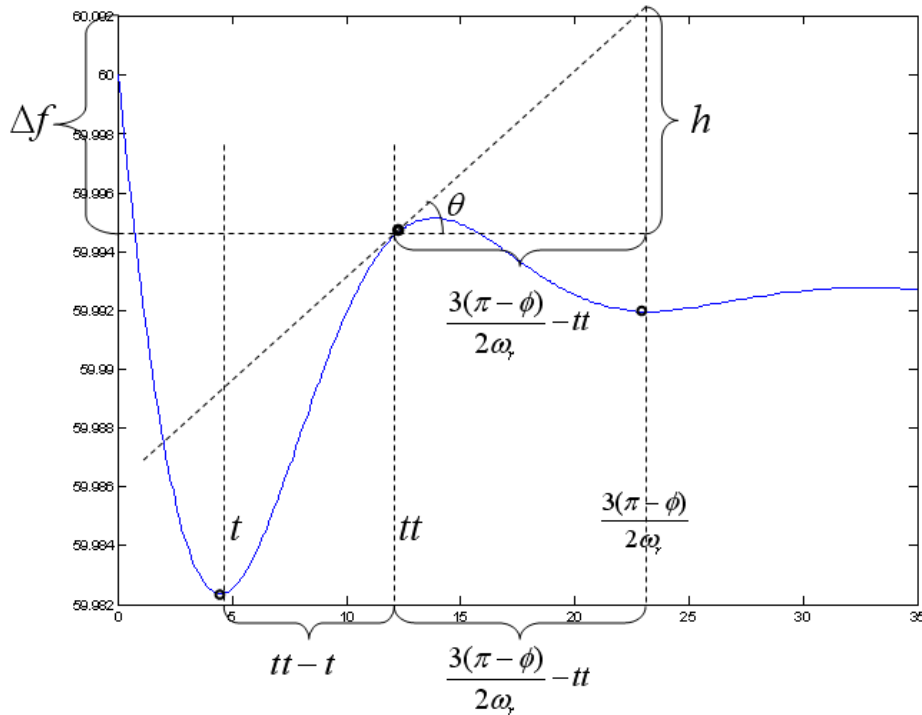


Figure 3.14 Parameters in Stopping Criteria for tt

This figure is based on an example with less damping where $F_H = 0.05$. Firstly, we calculate the nearest minimizers on both sides of the solution of tt . The one on the left is t , which we already worked out. The one on the right is close to the first minimizer of function $-\sin(\omega_r t + \phi)$ after tt . We can approximately find the first minimizer after tt by setting $\omega_r t + \phi = \frac{3}{2}\pi$, and we get $\frac{3(\pi - \phi)}{2\omega_r}$. Secondly, we compare the distance from tt

to the left minimizer and the right minimizer. Then we choose whichever is the larger in these two distances, $tt - t$ and $\frac{3(\pi - \phi)}{2\omega_r} - tt$. And then we multiply the larger distance by the slope of point tt . Like in figure 3.14, for example, $tt - t < \frac{3(\pi - \phi)}{2\omega_r} - tt$, so we multiply $\frac{3(\pi - \phi)}{2\omega_r} - tt$ by $\tan(\theta)$.

$$h = \max\left(tt - t, \frac{3(\pi - \phi)}{2\omega_r} - tt\right) \tan(\theta)$$

$$h = \left[\frac{3(\pi - \phi)}{2\omega_r} - tt \right] \tan(\theta)$$

And $\tan(\theta)$ is the derivative of the frequency function at point tt , which is $f'(tt)$. So,

$$h = \left[\frac{3(\pi - \phi)}{2\omega_r} - tt \right] f'(tt)$$

where,

$$f'(tt) = \frac{RP_{step} \alpha}{(DR + K_m) 2\pi} e^{-\zeta\omega_n tt} [\omega_r \cos(\omega_r tt + \phi) - \zeta\omega_n \sin(\omega_r tt + \phi)]$$

Therefore, we have the stopping criteria the same as 6.2.1: stop until $h < 0.01\Delta f$, where,

$$h = \left[\frac{3(\pi - \phi)}{2\omega_r} - tt \right] f'(tt)$$

$$\Delta f = \|f(tt) - 60\|$$

After checking the stopping criteria of iteration, we can get the first maximizer tt after t and the frequency value at this point.

(4) Find $t1$

As discussed in the previous sections, to find the first intersections of the frequency response curve and the horizontal frequency threshold curve, we need to solve the equation

$$\frac{RP_{step}}{(DR + K_m) 2\pi} [1 + \alpha e^{-\zeta\omega_n t} \sin(\omega_r t + \phi)] + 60 = f_{given}$$

The smallest solution is t_1 , and the second smaller solution is t_2 . We will also use Newton-Raphson method and Armijo step size rule to solve the equation. Figure 3.15 below is the flow chart to get t_1 .

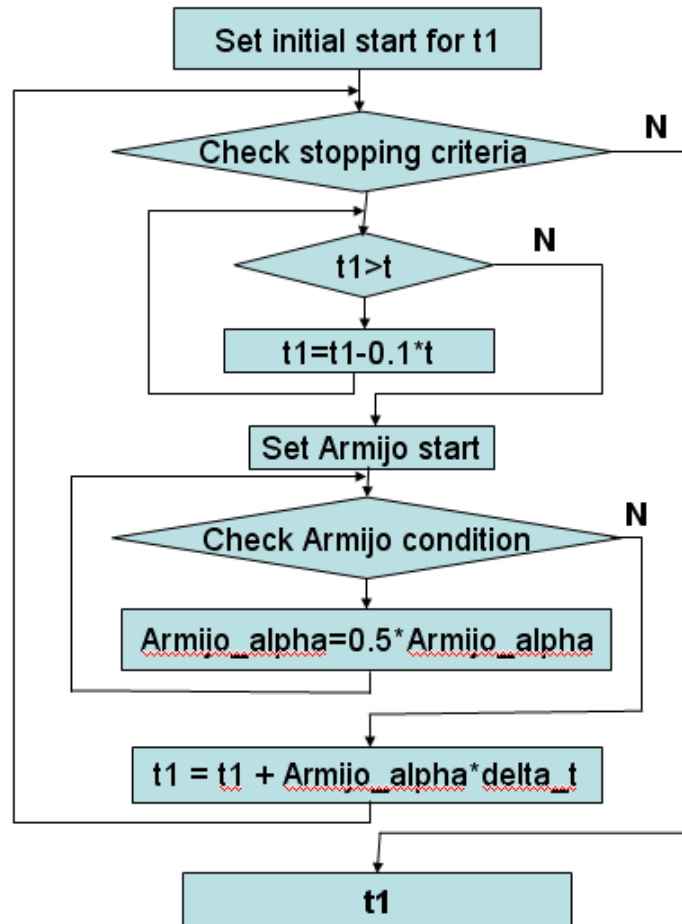


Figure 3.15 Flow Chart of Calculating t_1

First, we need to set an initial start for t_1 . From figure 3.15, we know that t_1 is between zero and the first local minimizer t . So we set the initial start from one-tenth of t left from t .

$$t_1 = t - 0.1t$$

Then we check the stopping criteria for the Newton-Raphson iteration: if t_1 is so close to the solution that the distance from t_1 to the given frequency threshold line is smaller than the absolute value of the frequency drop at this point, we can stop iterating and call t_1 a solution.

Figure 3.16 shows the parameters of the stopping criteria. Δf is the absolute value of the frequency drop of $t1$, which is $\Delta f = \|f(t1) - 60\|$. h is the distance from point $t1$ to the given frequency threshold line. The iteration will continue until $h < 0.01\Delta f$.

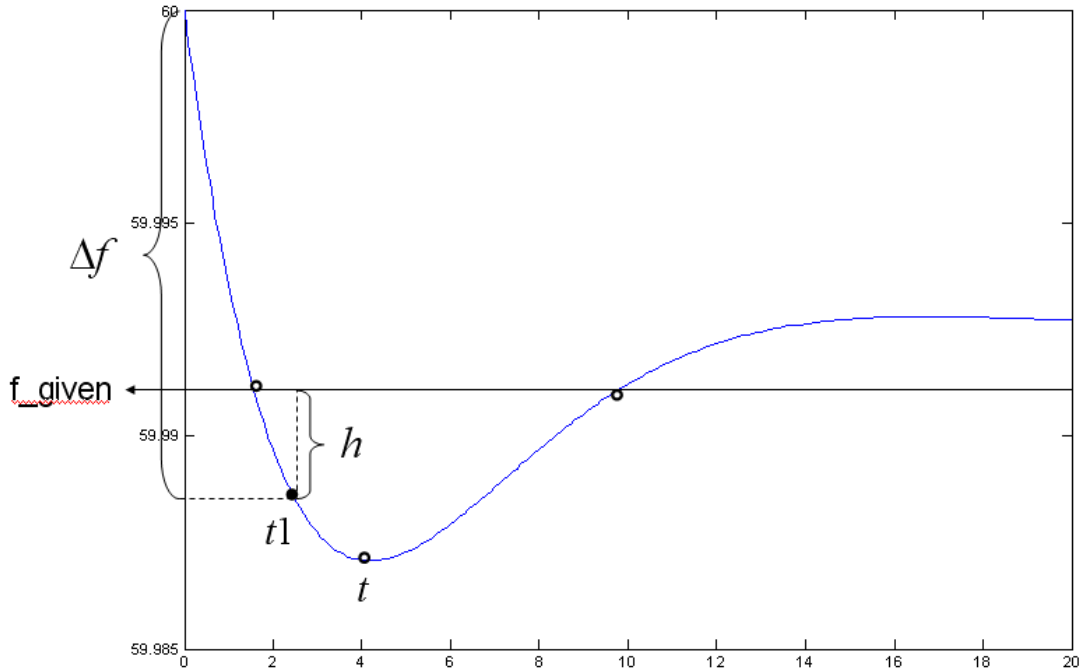


Figure 3.16 Parameters in Stopping Criteria for $t1$

Before we go on with the Newton-Raphson method, we check to make sure $t1$ does not run to the right side of t . Because if the system seeks the solution at the right side of t , it will end up at $t2$, the second intersection of the two functions. To avoid this, the system will first check if $t1$ is at the right side of t . If $t1 > t$, $t1$ will be moved one-tenth t to the left,

$$t1 = t1 - 0.1t \quad \text{for } t1 \text{ is moved to the left side of } t$$

Then we continue the Newton-Raphson method like in 6.2.1, and we will find $t1$, the time when the frequency first drops under the frequency threshold.

(5) Find t2

The steps and methods to find t2 are very similar to that of find t1, shown in figure 3.17.

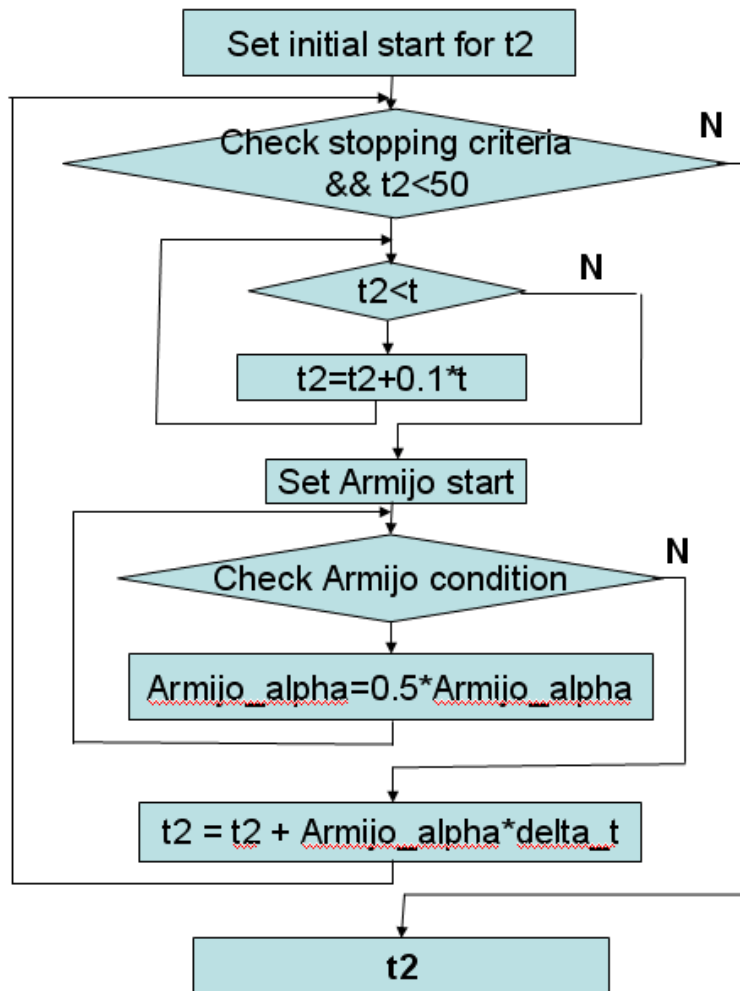


Figure 3.17 Flow Chart of Calculating t2

Similarly to keeping the estimates of t1 to the left of t, the value of the iteration of t2 should be kept at the right side of t. If $t2 < t$, the iteration t2 will be moved one-tenth to the right of t.

$$t2 = t2 + 0.1t$$

This will make sure that the solution does not end up at t1. When the system is checking the stopping criteria of the iteration, there is another limit on t before it updates the iteration. In a situation where $f_given > \max_f$, there will be no more intersections other than t1. So we have to set a limit to the iteration of t2 to prevent the system from keeping seeking t2 in vain. If the number of iterations is larger than 50, but

is still not close the solution then the system will stop searching and decide that t_2 is infinite. Then same steps are followed to find the second intersection t_2 . And $time_delay$ can be calculated by $t_2 - t_1$.

3.2.2.4 Results of the frequency outage checker

Two examples are listed below to show the results of running the code. In the first example, we use the typical parameters from the previous section;

- $R = 0.05$
- $H = 4.0s$
- $K_m = 0.95$
- $FH = 0.3$
- $TR = 8.0s$
- $D = 1.0$

and input:

- $P_{step} = -0.4$
- $input_f = 59.9883$

After running the code, we have the result:

- $t = 4.2045$
- $min_f = 59.9871$
- $t_1 = 2.5339$
- $t_2 = 6.8077$
- $time_delay = 4.2737$

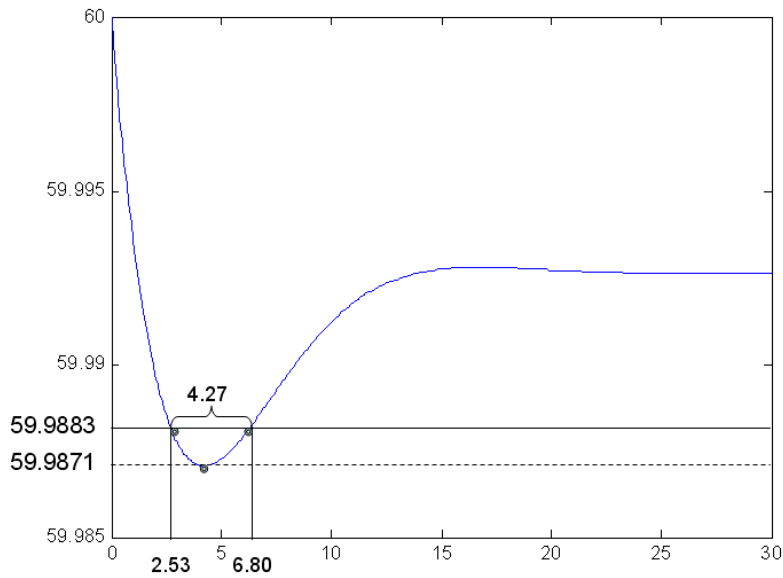


Figure 3.18 Result of running the code (1)

We can see in figure 3.18, it took 4.2045 seconds for the frequency to first drop to the minimum value of 59.9871 Hz, and the time duration when the frequency is below 59.9883 Hz is 4.2737 s.

Here is a second example with less damping where $FH = 0.1$. The other parameters and inputs are the same.

- $R = 0.05$
- $H = 4.0s$
- $K_m = 0.95$
- $FH = 0.1$
- $TR = 8.0s$
- $D = 1.0$
- $P_{step} = -0.4$
- $input_f = 59.9883$

After running the code, we have the result:

- $t = 4.4055$
- $\text{min}_f = 59.9835$
- $t1 = 1.9174$
- $t2 = 9.5382$
- $\text{time_delay} = 7.6208$

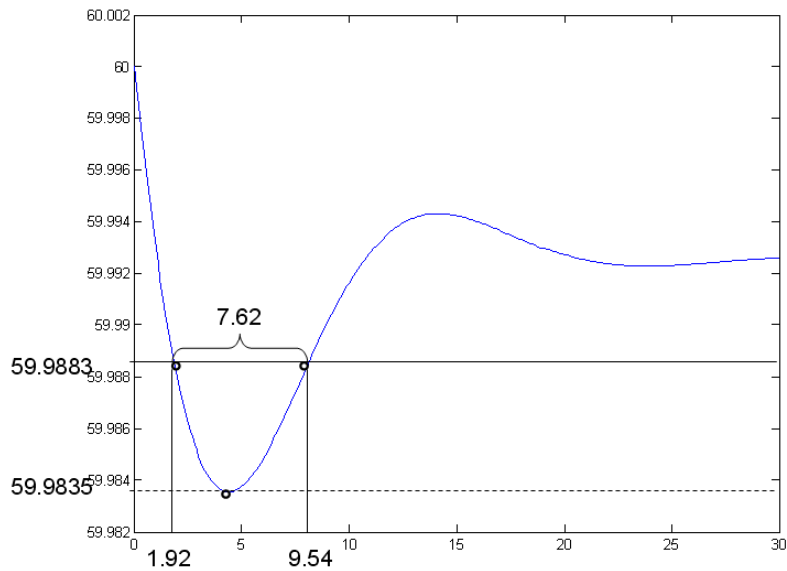


Figure 3.19 Result of running the code (2)

We can see in figure 3.19, it took 4.4055 seconds for the frequency to first drop to the minimum value of 59.9835 Hz, and the time duration when the frequency is below 59.9883 Hz is 7.6208 s.

4 Conclusion

In this project, UT has had the primary responsibility for developing a cascading outage analysis tool that has its own graphical user interface (GUI). This work has concentrated on developing the simulation algorithm for the cascading outage checkers and the GUI, both in a windows-based environment.

We developed version 1.0 of the Cascading Outage Analyzer (COA) software as PC-based Windows software. The Cascading Outage Analyzer (COA) uses MS Access as a database. In order to reuse a previously implemented power flow module and provide flexibility to switch to or add different power flow modules and determine a specific module to run at run-time, three design patterns, the Adapter, Strategy, and Factory Method design patterns, have been adopted.

Following the initial disturbance, the Cascading Outage Analyzer software has three outage checking algorithms namely the 'Frequency Checker', 'Line Overload Checker' and 'Under Voltage Checker'. Among the three checkers, the line overload and bus under-voltage checkers use the power flow algorithms and users can select between two variations of the load flow program: Full AC power flow and Decoupled power flow, which are provided by Commonwealth Associates Inc. On the other hand, we use the System Frequency Response (SFR) model as a frequency change model in the frequency checker algorithm.

The developed COA software is verified with a 9 bus system and this system is presented in 'Power System Control and Stability' [8] by Anderson and Fouad. The scenario simulation of 9 bus system for understanding the developed software will be presented in the Appendix of this report.

In future, the developed Cascading Outage Analyzer (COA) will be enhanced with new version and the following issues will be upgraded in the next version.

- Enhance Input DB system
- Upgrade simulation engines and algorithms
- Implement simulation output files (for example, *.CSV)

Appendix A: User's Manual

Cascading Outage Analyzer V1.0 Jan 2009 System Manual

- Instructions for the Cascading Outage Analyzer User Interface -

Contents

Introduction to the system manual	53
Installation and start	53
System requirements	53
Installing COA	53
Starting COA	53
User interface	54
User interface elements	54
Menu Bar	54
Toolbar	55
System Information Bar	55
Workspace	56
Basic functions	57
Open Power System Data	57
Exit	58
Power System Data	59
Buses	59
Generators	59
Lines	60
Load	61
Set Up COA-sys	62
Disturbance	62
Cascading Outage Checker	63
Frequency Checker	64
Simulate COA-sys	65
Run Checker	65
Simulation Report	66
Help	68
About COA	68
Example	69
Opening a power system network	69
Viewing the power system network data	71
First Disturbance: Calculations and results	72
Second Disturbance: Calculations and results	77
User's Manual Index	83

Introduction to the system manual

Installation and start

System requirements

Minimum hardware requirements

- PC or notebook
- CPU: x86 compatible
- RAM: 256 MB
- Hard disk Space: 16 MB

Software requirements

- MS Windows XP 32 bit
- MS Access 2003
- MS .NET Framework (1.1)
- PFlow (Commonwealth Associates Inc.)

Installing COA

To install the Cascading Outage Analyzer (v 1.0) the following steps are required:

- The folder **COA_v1.0** from the installation disk must be copied to the hard disk C:\
- A folder named *Dataset* with the following destination must be created
C:\project\Dataset
- A folder named *source* with the following destination must be created
C:\project\implementation\source
- The files **COChecker.mdb** and **Datainput.mdb** from the installation disk must be copied to the *Dataset* folder specified above
- The file **temp.mdb** from the installation disk must be copied to the *source* folder specified above

Starting COA

Clicking on the *Cascading Outage Analyzer.sln* file located in the folder COA_v1.0 opens the Microsoft Visual Studio application. To start the COA the user must then press **CTRL+F5** or click the **PLAY** button (see Fig.A1).

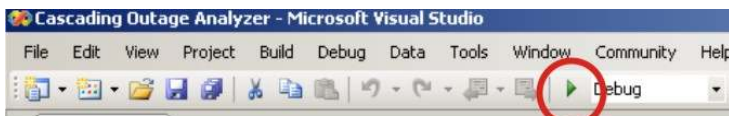


Figure A.1: PLAY button on Microsoft Visual Studio toolbar to start the COA.

User interface

After opening the COA application the user views the configuration of the COA system:

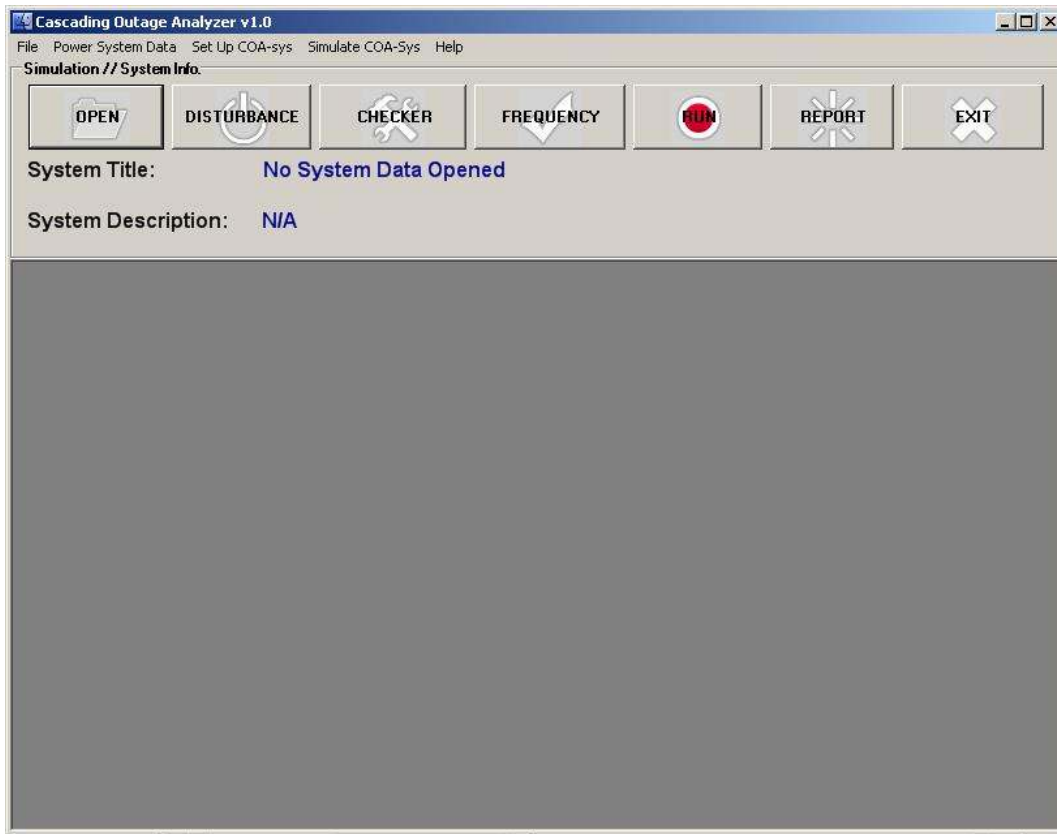


Figure A.2: COA User Interface

User interface elements

The user interface elements include the following:

- Menu Bar
- Toolbar
- System Information Bar
- Workspace

Menu Bar

COA's menu bar is an important element because it accesses all functions of the COA application.



Figure A.3: COA's Menu Bar.

The menu bar contents are:

- File
The File menu contains basic functions of the COA which include opening power system data files and exiting the application.
- Power System Data
The Power System Data menu is used to view the data of the basic components of the power system, which include buses, generators, loads and lines.
- Set Up COA-sys
The Set Up COA-sys menu is used to set the simulation options, such as choosing the disturbance or the cascading outage checkers.
- Simulate COA-Sys
The Simulate COA-Sys menu contains the run and report functions.
- Help
The Help menu contains general information about the application .

Toolbar

The icons of the toolbar allow access to all common functions of the program, which can also be selected from menus.



Figure A.4: COA's Toolbar

The elements of the toolbar are:

- Open
- Disturbance
- Checker
- Frequency
- Run
- Report
- Exit

System Information Bar

The System Information Bar contains the system title and the basic description of the loaded power system.



Figure A.5: COA's System Information Bar

Workspace

The Workspace takes up most of the window. This area is used to view the input and output data. The input data include the power system information data handled by Access DB and the simulation options. The output data are contained in the Simulation Report. The message boxes are also displayed in the Workspace area.

Basic functions

The basic functions of the COA application are opening a power system and exiting the application. Both these functions are contained in the File menu of the COA's Menu bar.



Figure A.6: COA's File menu.

Open Power System Data

To load the input data of an existing power system, the user must click **File - Open Power System Data** or click the **OPEN** icon on the toolbar. This action opens the following dialog box:

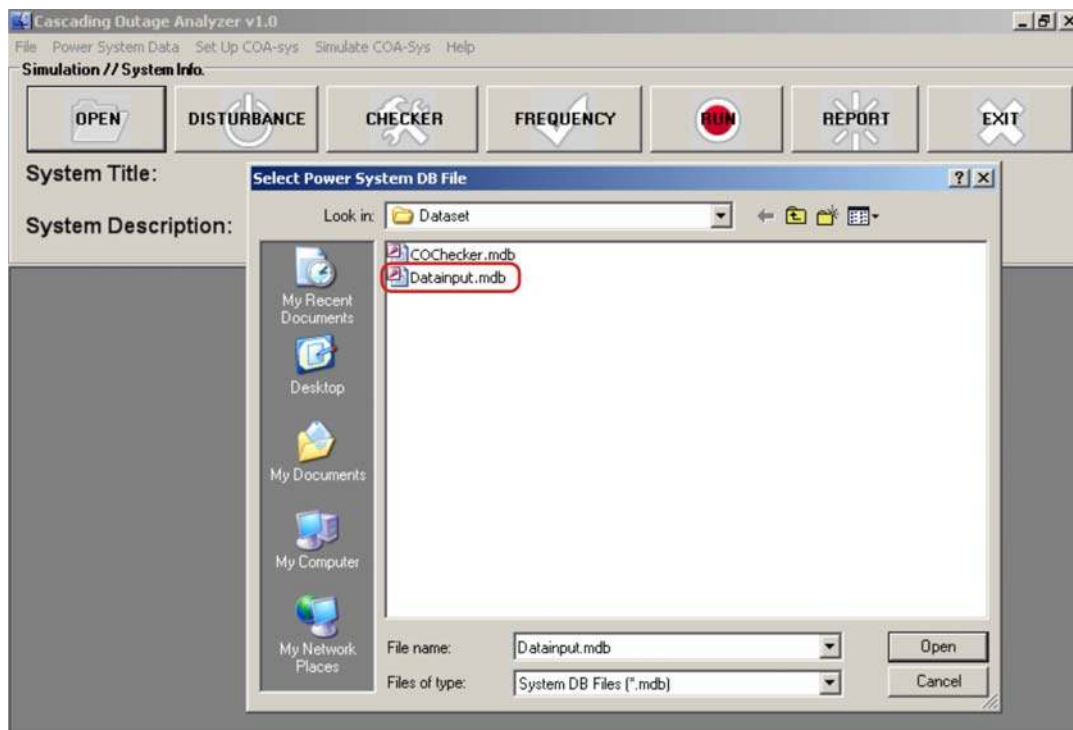


Figure A.7: Selecting Power System input data dialog box.

By selecting *Datainput.mdb* and clicking **Open** at the above dialog box the user views the following message, which declares the successful loading of the input data on the COA application.

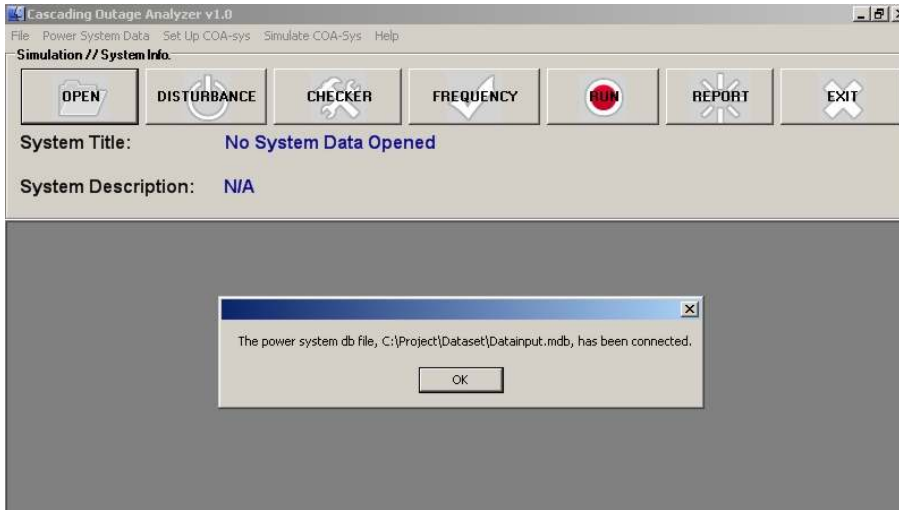


Figure A.8: Successful upload of the system input data message.

By clicking **OK** in the above message box the System Title and System Description of the loaded power system appear on the System Information Bar (see Fig.A.5).

Exit

To exit the COA application the user can either select **File - Exit** or click on the **EXIT** icon on the toolbar.

Power System Data

The user can view the data of the basic components of the power system with the Power System Data menu of the Main menu.

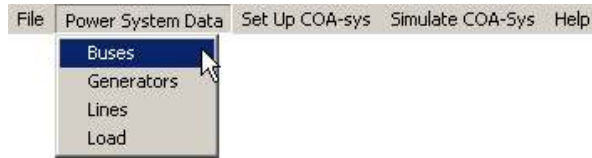


Figure A.9: Power System Data menu.

Buses

The user can view the power system buses data by clicking **Power System Data - Buses** from the Main menu and then **View Bus Data - + - Table** on the *Buses Window* in the Workspace. The user can exit the *Buses Window* by clicking on the **Exit** button next to the View Bus Data button.

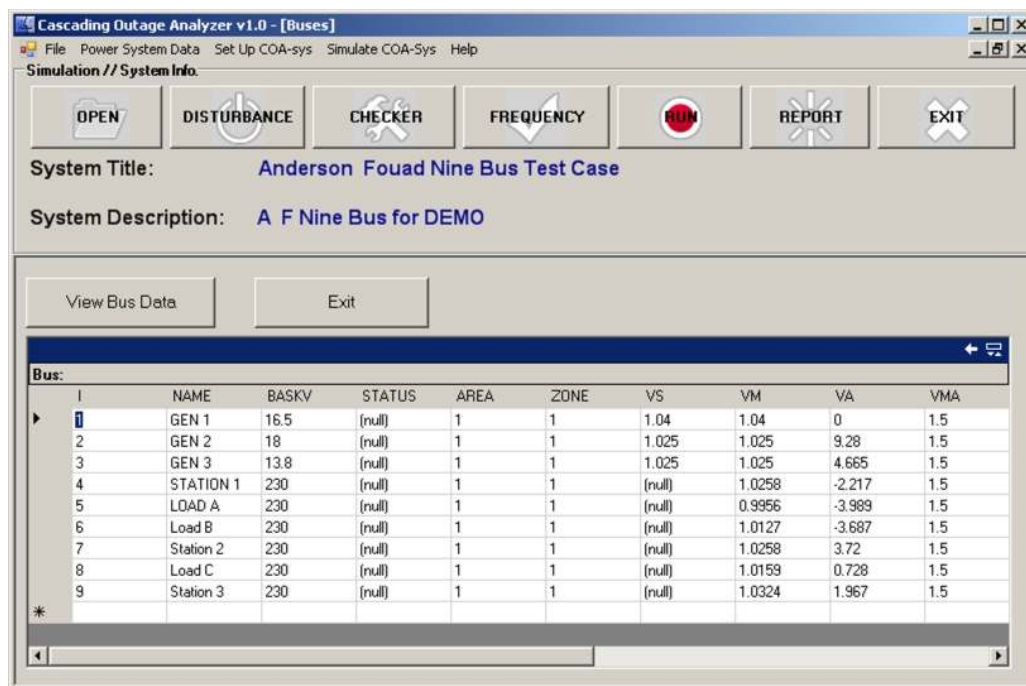


Figure A.10: Power system buses data (Buses Window).

Generators

The user can view the power system generators data by clicking **Power System Data - Generators** from the Main menu and then **View GEN Data** on the *GeneratorForm Window* in the Workspace. The user can exit the *GeneratorForm Window* by clicking on the

Exit button next to the View GEN Data button.

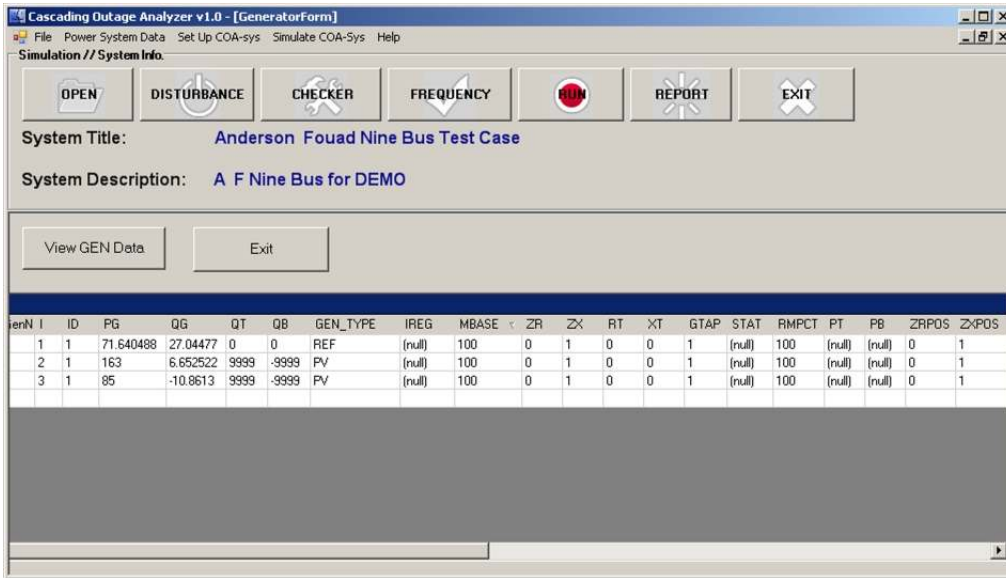


Figure A.11: Power system generator data (GeneratorForm Window).

Lines

The user can view the power system lines data by clicking **Power System Data - Lines** from the Main menu and then **View LINE Data** on the *LineForm Window* in the Workspace. The user can exit the *LineForm Window* by clicking on the **Exit** button next to the View LINE Data button.

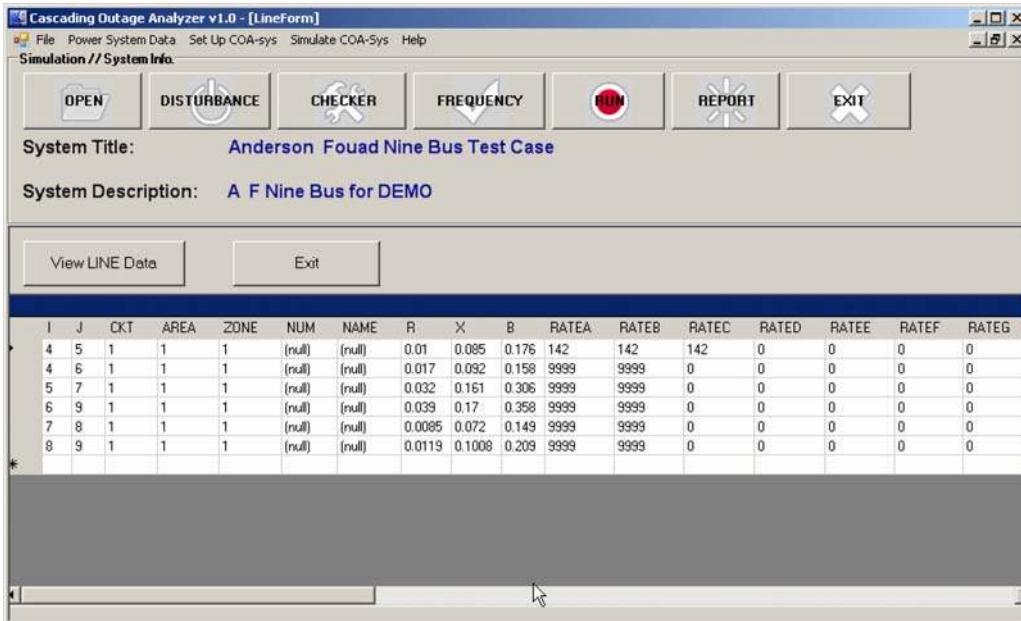


Figure A.12: Power system line data (LineForm Window).

Load

The user can view the power system load data by clicking **Power System Data - Load** from the Main menu and then **View Load Data** on the *LoadForm Window* in the Workspace. The user can exit the *LoadForm Window* by clicking on the **Exit** button next to the View Load Data button.

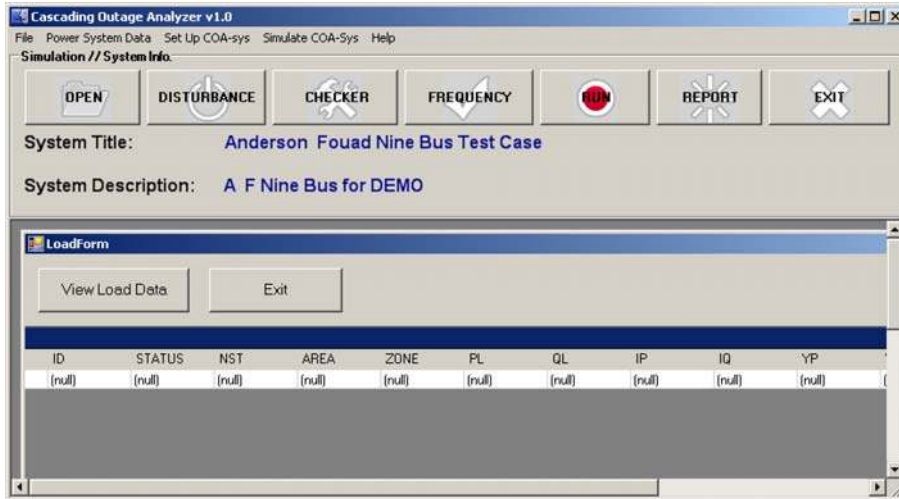


Figure A.13: Power system load data (LoadForm Window).

Set Up COA-sys

From the Set Up COA-sys menu of the Main menu the user can set-up the simulation scenarios and define the options for the used outage checkers.

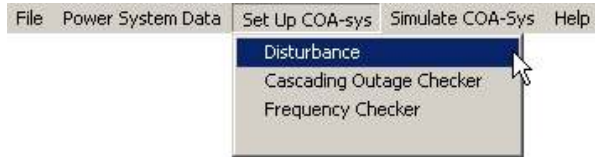


Figure A.14: Set Up COA-sys menu.

Disturbance

By clicking **Set Up COA-sys - Disturbance** from the Main menu or the **DISTURBANCE** icon on the toolbar the user can view the *Disturbance Editor Window* in the Workspace area. This Window displays the information regarding the selected initial disturbance in the power system. Only one disturbance at a time can be handled and only of the Line Outage form (this disturbance applies for a line or any branch device outage).

The **Reload** button at the bottom of this Window can be used to load the current data stored in the *Disturbance* table in the corresponding database (make modifications without saving them), while the **Update** button actually updates the data of the *Disturbance* table in the corresponding database (make modifications and save them). The user can exit the *Disturbance Editor Window* by clicking on the **Close** button on this Window.

When the user modifies the entries in the *Disturbance Editor Window* the user must also make the same modifications in the *Disturbance* table in the database located at *C:\project\Dataset\Datainput.mdb*.

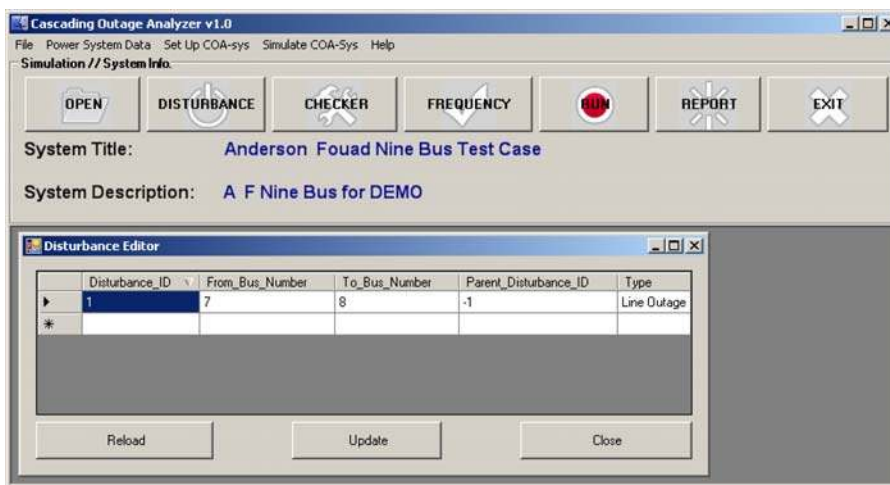


Figure A.15: Disturbance Editor Window displaying the initial disturbance.

Cascading Outage Checker

By clicking **Set Up COA-sys - Cascading Outage Checker** from the Main menu or the **CHECKER** icon on the toolbar the user can view the *Overload and Voltage Editor Window* in the Workspace area. This Window displays the types of the outage checkers that will be used and their usage priority.

There are three types of outage checkers:

- Line Overload
- Under Voltage
- Frequency Checker

and their order of usage can be determined by varying the entries in the *Time_Span* column (smallest number in *Time_Span* corresponds to the outage checker that will be used first).

The **Reload** button at the bottom of this Window can be used to load the current data stored in the *Outage Checker* table in the corresponding database (make modifications without saving them), while the **Update** button actually updates the data of the *Outage Checker* table in the corresponding database (make modifications and save them). The user can exit the *Overload and Voltage Editor Window* by clicking on the **Close** button on this Window.

When the user modifies the entries in the *Overload and Voltage Editor Window* the user must also make the same modifications in the *Outage Checker* table in the database located at *C:\project\Dataset\Datainput.mdb*.

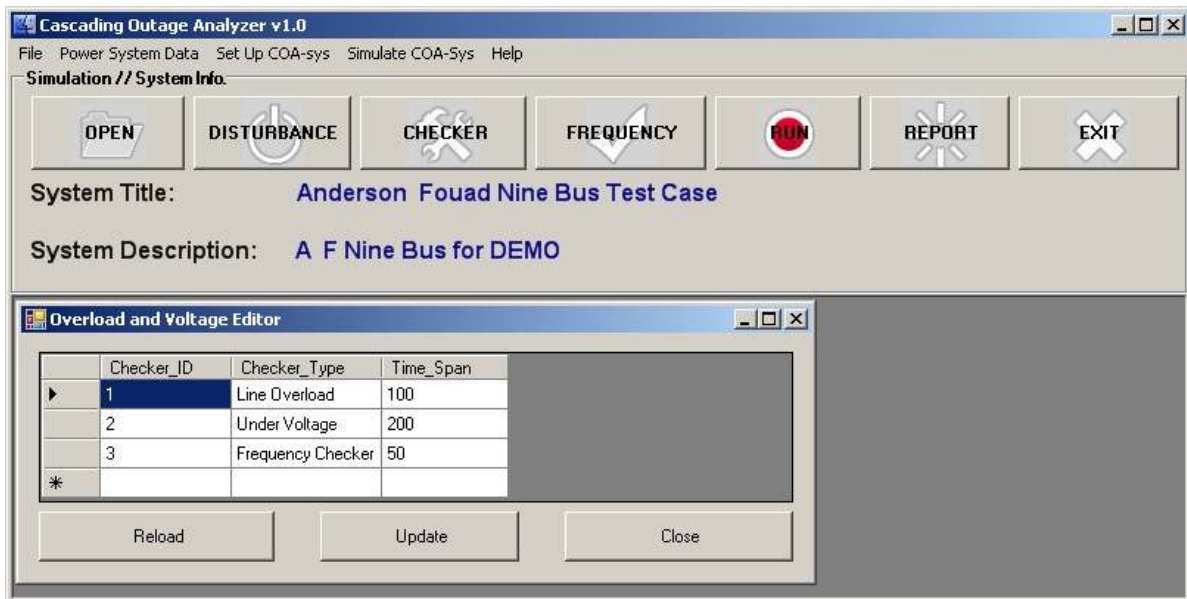


Figure A.16: Overload and Voltage Editor Window displaying the Outage Checkers used.

Frequency Checker

By clicking **Set Up COA-sys - Frequency Checker** from the Main menu or the **FREQUENCY** icon on the toolbar the user can view the *Frequency Editor Window* in the Workspace area. This Window displays the characteristics of the frequency outage checkers that will be used.

The columns of this table provide the following information:

- Bus: The bus number of the power system
- BusType: The type of the bus (1: Generation bus, 2: Load bus)
- Threshold: Input value for the frequency threshold
- Delay: Input value for setting the time delay of the frequency relay.

The **Reload** button at the bottom of this Window can be used to load the current data stored in the *FOChecker* table in the corresponding database (make modifications without saving them), while the **Update** button actually updates the data of the *FOChecker* table in the corresponding database (make modifications and save them). The user can exit the *Frequency Editor Window* by clicking on the **Close** button on this Window.

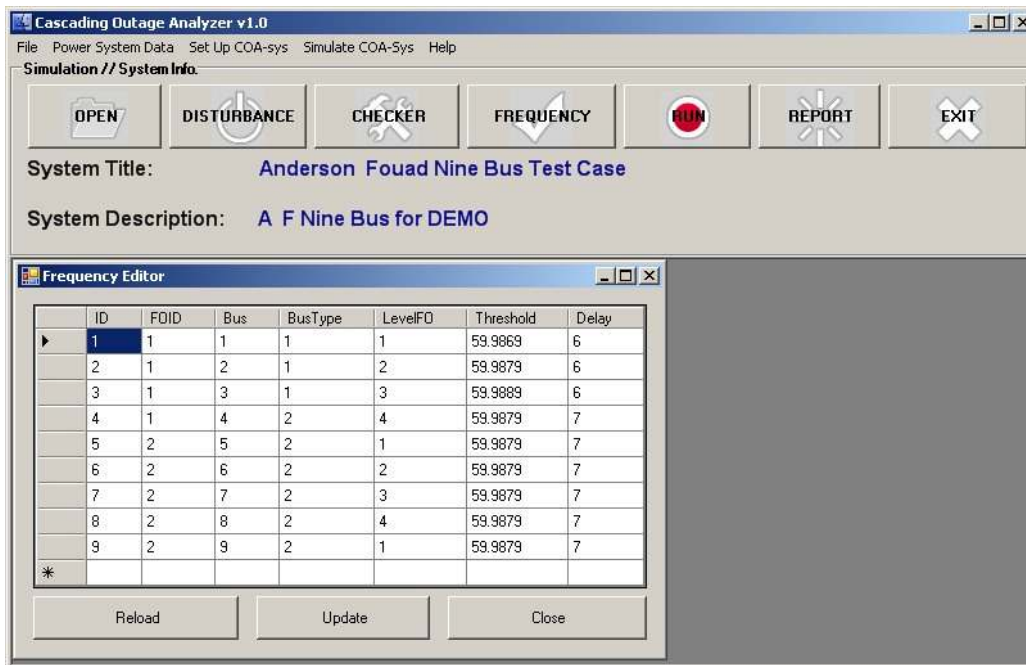


Figure A.17: Frequency Editor Window.

Simulate COA-sys

After setting the options and the parameters of the simulation the user can simulate a disturbance and view the results of the simulation from the Simulate COA-Sys menu.

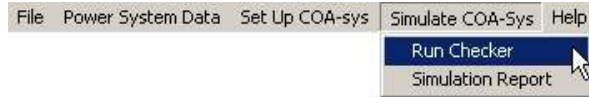


Figure A.18: Simulate COA-Sys menu.

Run Checker

By clicking **Simulate COA-Sys - Run Checker** from the Main menu or the **RUN** icon on the toolbar the user views the *Select Power Flow Algorithm Window* in the Workspace area. This Window prompts the user to choose the desired power flow method to be used in the simulation:

- AC Power Flow
- Decoupled Power Flow

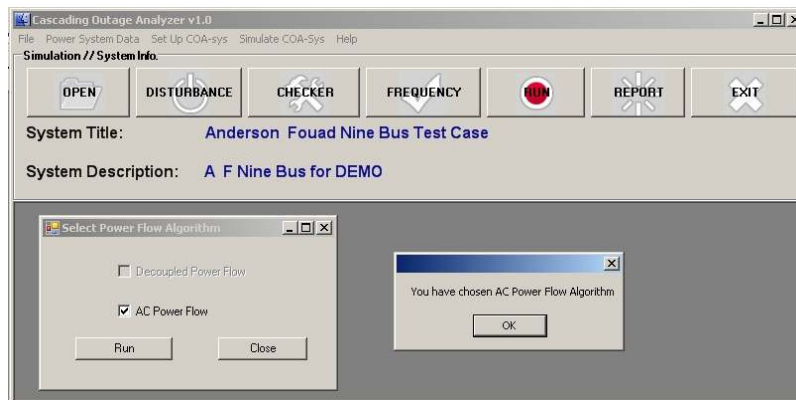


Figure A.19: Selecting the desired Power Flow Algorithm Window.

After ticking the corresponding box in the Window a message box appears in the Workspace area declaring the chosen Power Flow Algorithm. After first clicking **OK** in the message box and then clicking **Run** in the *Select Power Flow Algorithm Window* the simulation starts.

Several windows open in the Workspace area before the simulation stops (see Fig. A.20). The user should wait until the end of the simulation, which is declared by a message box (see Fig. A.21) displaying either:

- Result: 1 - Successful run of the simulation scenario or
- Result: 0 - Failure to run the simulation scenario.

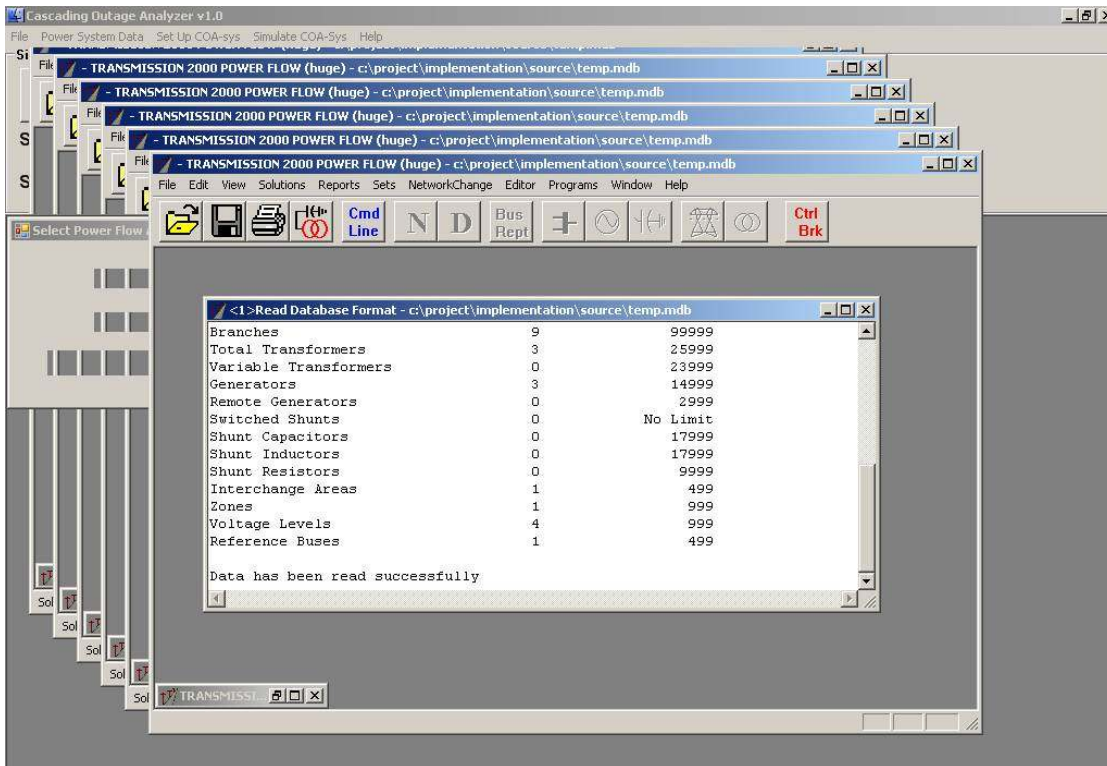


Figure A.20: The running process.

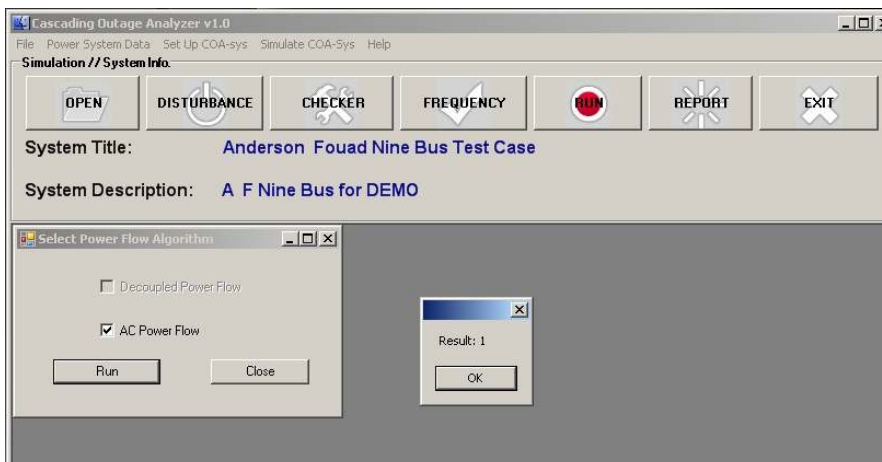


Figure A.21: End of simulation.

After the end of the simulation process the user should click **OK** in the message box and **Close** in the *Select Power Flow Algorithm Window*.

Simulation Report

By clicking **Simulate COA-Sys - Simulation Report** from the Main menu or the **REPORT** icon on the toolbar the user views the *Cascading Outage Report Window* in the Workspace area. This window presents the resulting outages including the initial

disturbance in tree view (Cascading Outage Tree View tab).

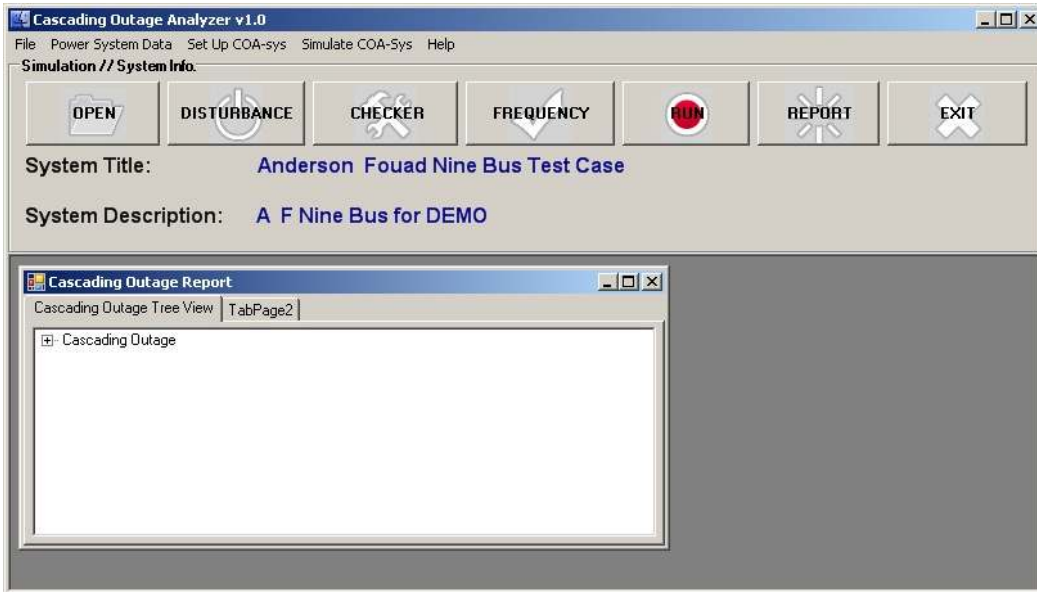


Figure A.22: Cascading Outage Report Window.

The user must click successively the + sign in order to view the results of the simulation. The results are displayed in chronological order, thus the first line under Cascading Outage is the initial disturbance.

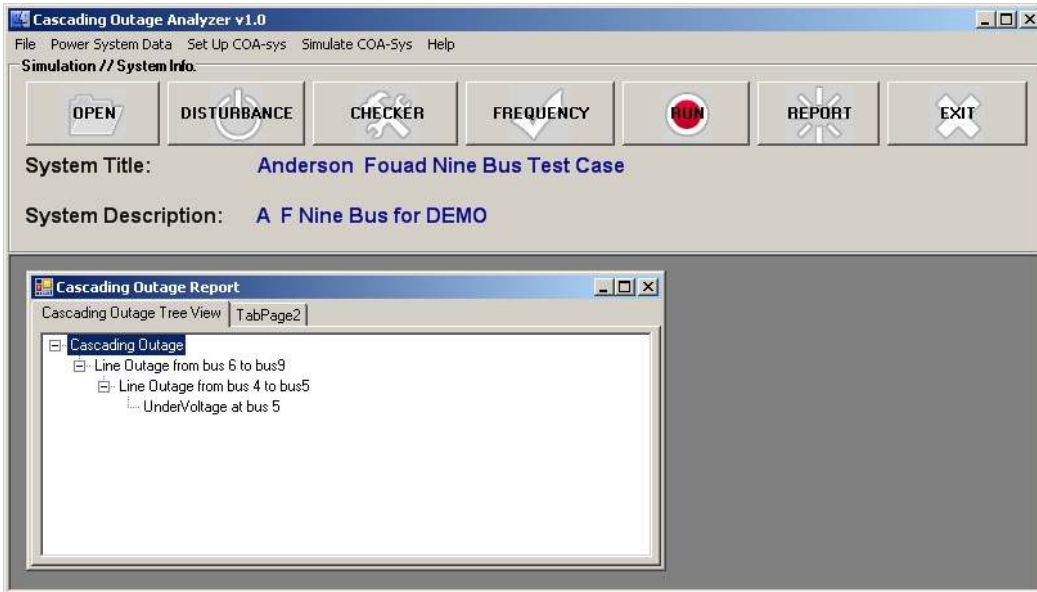


Figure A.23: Cascading outage results displayed in tree view format.

Help

About COA

The user can view the current version number and general information about the COA application in the *HelpInfo Window* by clicking **Help - About Cascading Outage Analyzer (COA)**.

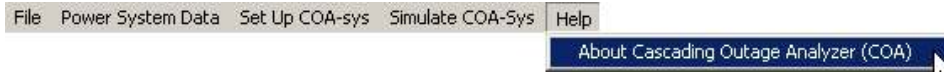


Figure A.24: COA's Help menu.

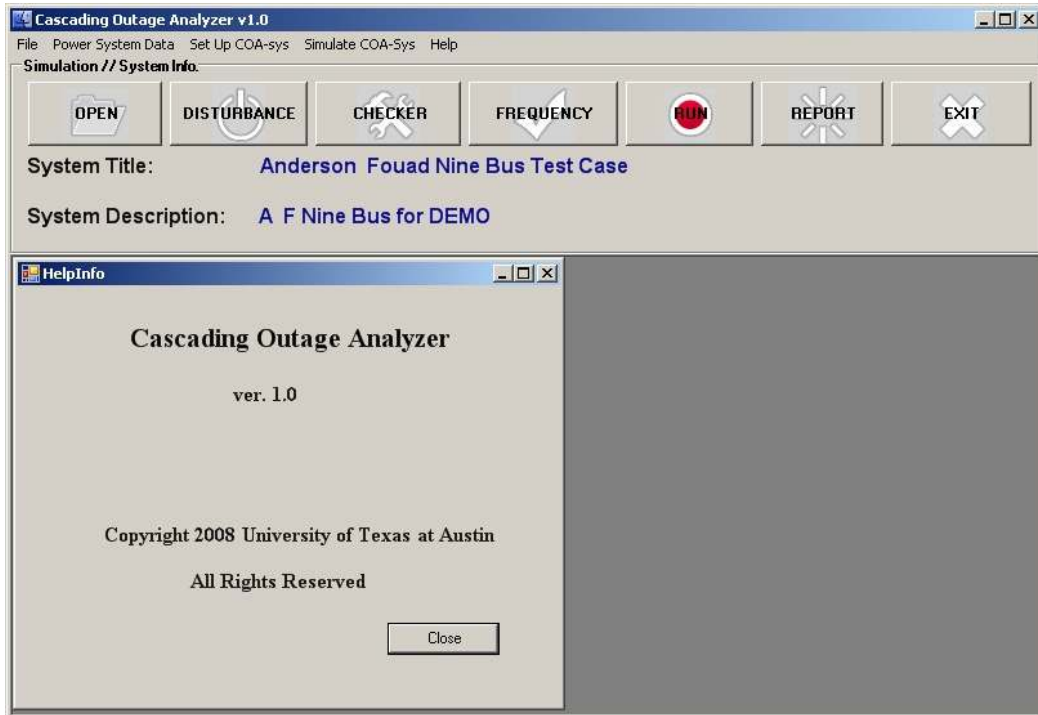


Figure A.25: HelpInfo Window with general information about the COA application.

Example

The function of the COA application will be illustrated with an example.

For this example the following power system is considered and two simulation scenarios are presented.

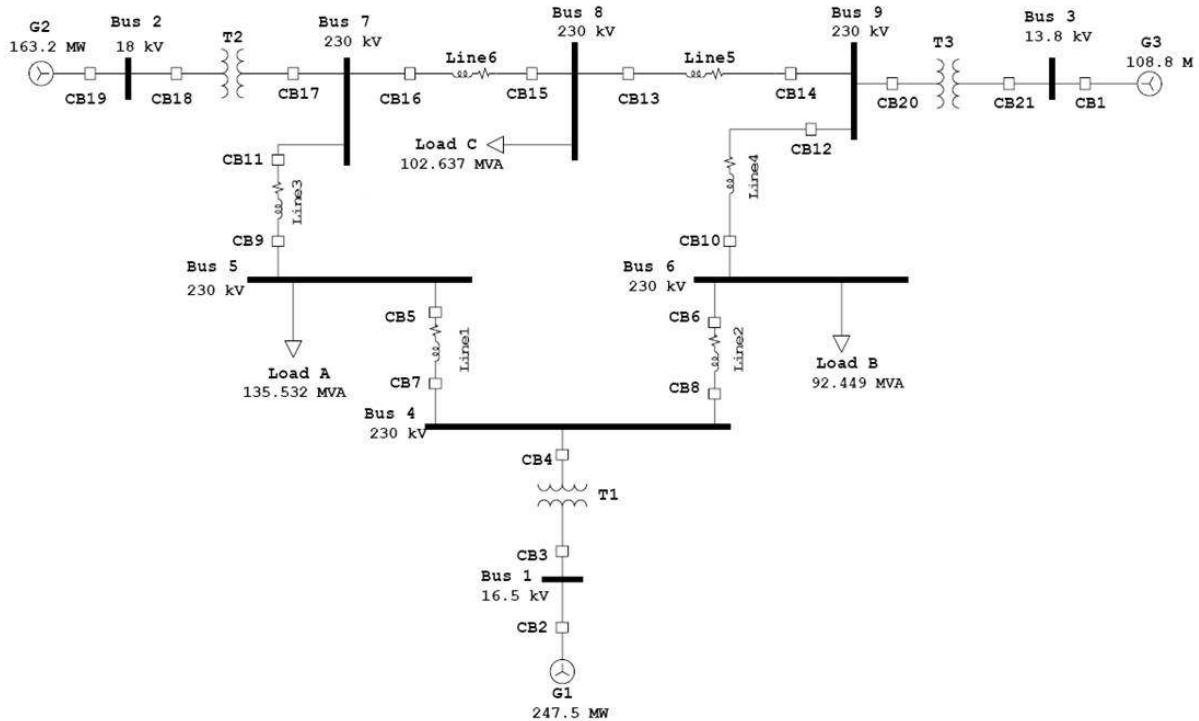


Figure A.26: One line diagram of the Example power system network [8].

Opening a power system network

In the COA application previously implemented power flow software is used to determine sequential equilibrium states for checking if further outages will occur following an initial disturbance. The implemented outage checkers utilize an independent power flow module, PFlow, which has been developed by Commonwealth Associates Inc. and is licensed to The University of Texas.

For this reason the components of the example power system depicted in Fig. A.26 and their associated data are stored in a database, in compliance with the reused software. The database for the example power system *Datainput.mdb* is contained in the application installation disk. To utilize the COA application this database must be copied to the location *C:\project\Dataset\Datainput.mdb*.

If the user wants to test simulation scenarios on a new power system the format of the given

Datinput.mdb database must be maintained. For this reason the developers of the COA application recommend that the given database is copied and then modified accordingly. Although the database contains many tables the user has to modify only the following tables for the power flow calculation:

- BUS: contains the bus data of the network,
- GENERATOR: contains the generator data of the network,
- LINE: contains the line data of the network,
- LOAD: contains the load data of the network.

However, regardless of the power system network the database used must be named *Datinput.mdb* and copied to the location *C:\project\Dataset\Datinput.mdb*.

To view the configuration of the COA application the user must click on the *Cascading Outage Analyzer.sln* file located in the folder *COA_v1.0* and then press **CTRL+F5** or click the **PLAY** button. To load the input data of the example power system, the user must click **File - Open Power System Data** or click the **OPEN** icon on the toolbar (see Fig. A.27).

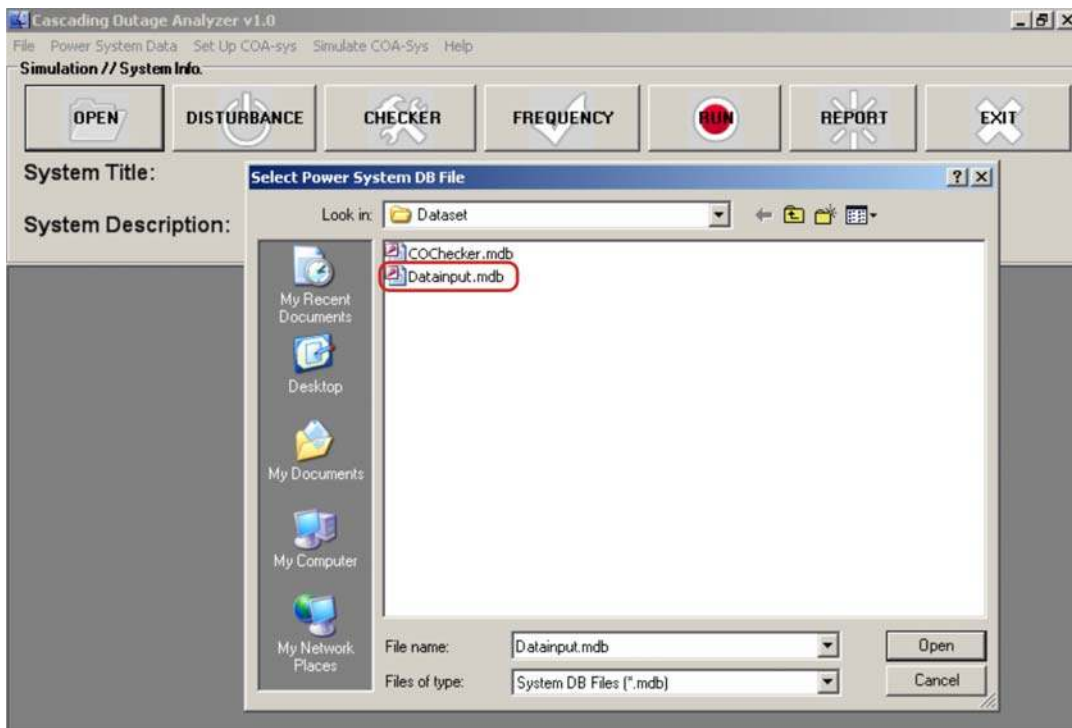


Figure A.27: Opening the example power system.

By selecting *Datinput.mdb* and clicking **Open** at the above dialog box the user views a message box declaring the successful loading of the input data on the COA application. After clicking **OK** the System Title and System Description of the loaded example power system appear on the System Information Bar (see Fig.A.28).

System Title:	Anderson Fouad Nine Bus Test Case
System Description:	A F Nine Bus

Figure A.28: Example power system description shown at the COA's System Information Bar

Viewing the power system network data

The example power system is a nine-bus power system with three generators, three transformers, three loads and six lines. The user can view the data of this network using the Power System Data menu of the Main menu of the COA application.

For example, to view the example power system buses data the user must click **Power System Data - Buses** from the Main menu and then **View Bus Data - + - Table** on the *Buses Window* in the Workspace.

The screenshot shows the 'Cascading Outage Analyzer v1.0 - [Buses]' window. The 'System Info' section displays the system title 'Anderson Fouad Nine Bus Test Case' and description 'A F Nine Bus for DEMO'. Below this, there are buttons for 'View Bus Data' and 'Exit'. The main area contains a table with the following data:

Bus:	I	NAME	BASKV	STATUS	AREA	ZONE	VS	VM	VA	VMA
	1	GEN 1	16.5	(null)	1	1	1.04	1.04	0	1.5
	2	GEN 2	18	(null)	1	1	1.025	1.025	9.28	1.5
	3	GEN 3	13.8	(null)	1	1	1.025	1.025	4.665	1.5
	4	STATION 1	230	(null)	1	1	(null)	1.0258	-2.217	1.5
	5	LOAD A	230	(null)	1	1	(null)	0.9956	-3.989	1.5
	6	Load B	230	(null)	1	1	(null)	1.0127	-3.687	1.5
	7	Station 2	230	(null)	1	1	(null)	1.0258	3.72	1.5
	8	Load C	230	(null)	1	1	(null)	1.0159	0.728	1.5
	9	Station 3	230	(null)	1	1	(null)	1.0324	1.967	1.5

Figure A.29: Power system buses data for the nine bus example power system.

To view the example power system lines data the user must first exit the Buses Window above and then click **Power System Data - Lines** from the Main menu and then **View LINE Data** on the *LineForm Window* in the Workspace. The user can exit the *LineForm Window* by clicking on the **Exit** button next to the View LINE Data button.

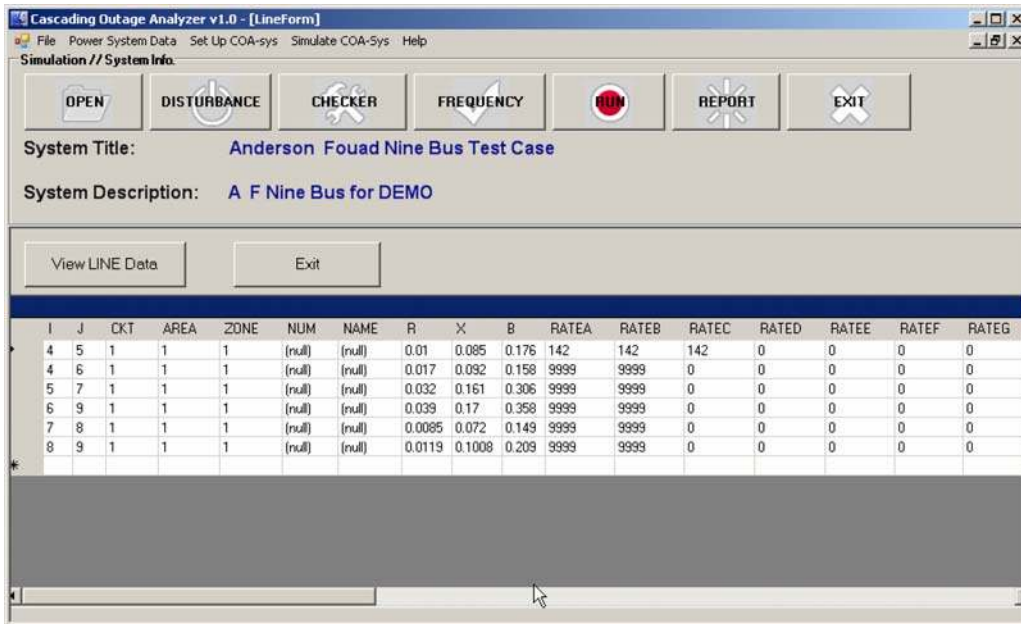


Figure A.30: Power system line data for the nine bus example power system.

The described procedure is similar for viewing the example power system generator or load data.

First Disturbance: Calculations and results

The first initial disturbance to be applied to the example power system is the line outage of the Line 6 between Buses 7 and 8, which is equivalent to opening the circuit breakers CB 15 and CB 16, as shown in Fig. A.31.

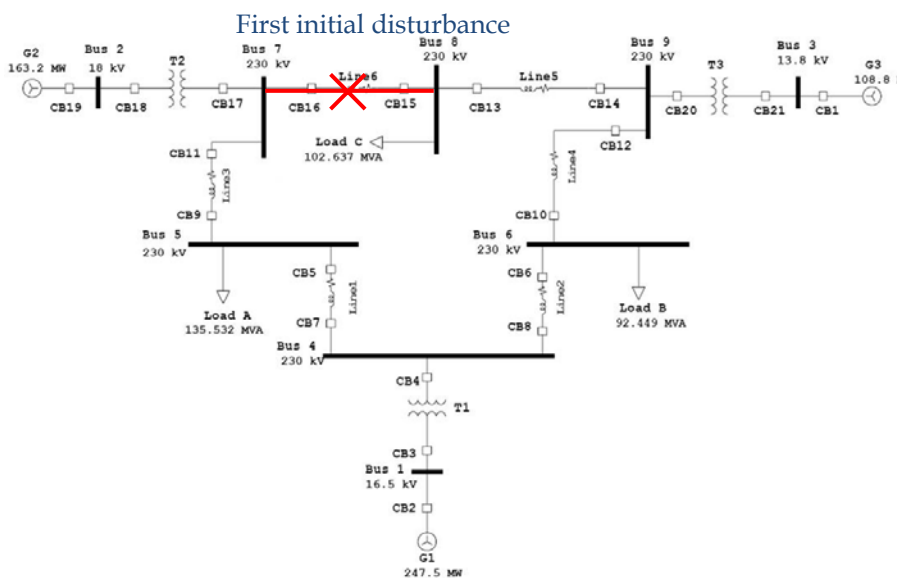


Figure A.31: First initial disturbance applied to the example power system.

By clicking **Set Up COA-sys - Disturbance** from the Main menu or the **DISTURBANCE** icon on the toolbar the user can view the *Disturbance Editor Window* in the Workspace area. This Window displays the information regarding the initial disturbance in the power system, as shown in Fig. A.32.

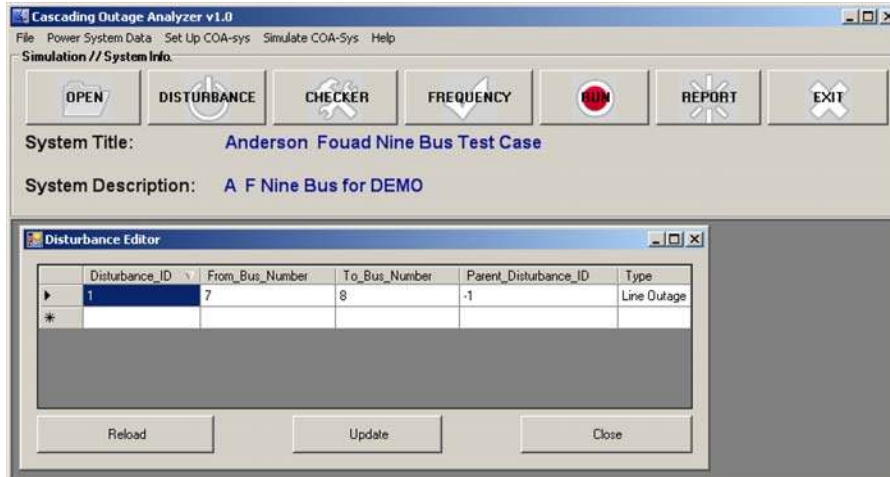


Figure A.32: Disturbance Window displaying the first disturbance for the example power system.

By clicking **Set Up COA-sys - Cascading Outage Checker** from the Main menu or the **CHECKER** icon on the toolbar the user can view the *Overload and Voltage Editor Window* in the Workspace area. For the first initial disturbance all the available outage checkers will be used, in the following order:

- Frequency Checker
- Line Overload
- Under Voltage

as is denoted by the respective value in the *Time_Span* column and shown in Fig. A.33.

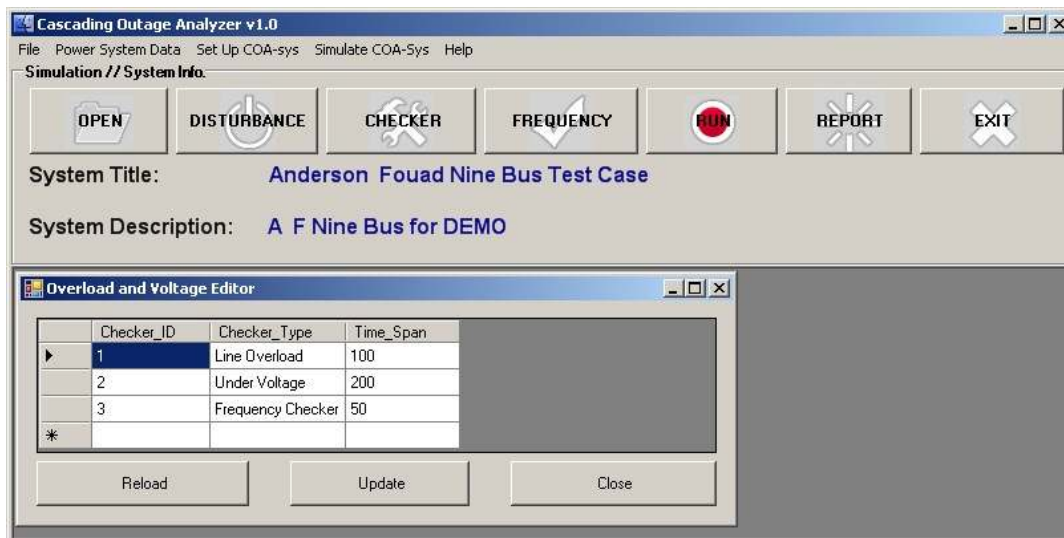


Figure A.33: Window displaying the outage checkers used in the first initial disturbance.

By clicking **Set Up COA-sys - Frequency Checker** from the Main menu or the **FREQUENCY** icon on the toolbar the user can view the *Frequency Editor Window* in the Workspace area. The window displaying the characteristics of the frequency outage checkers that are used in the first initial disturbance applied to the example power system is shown in Fig. A.34. These characteristics include the bus number and type, the frequency threshold and the time delay for the frequency relays. As can be seen from Fig. A.34 the frequency checkers have under-frequency relays. The frequency checkers at load buses have greater values for the time delay of their under-frequency relays than the ones used at generator buses. Moreover, the frequency checker at bus 3 has the highest under-frequency threshold.

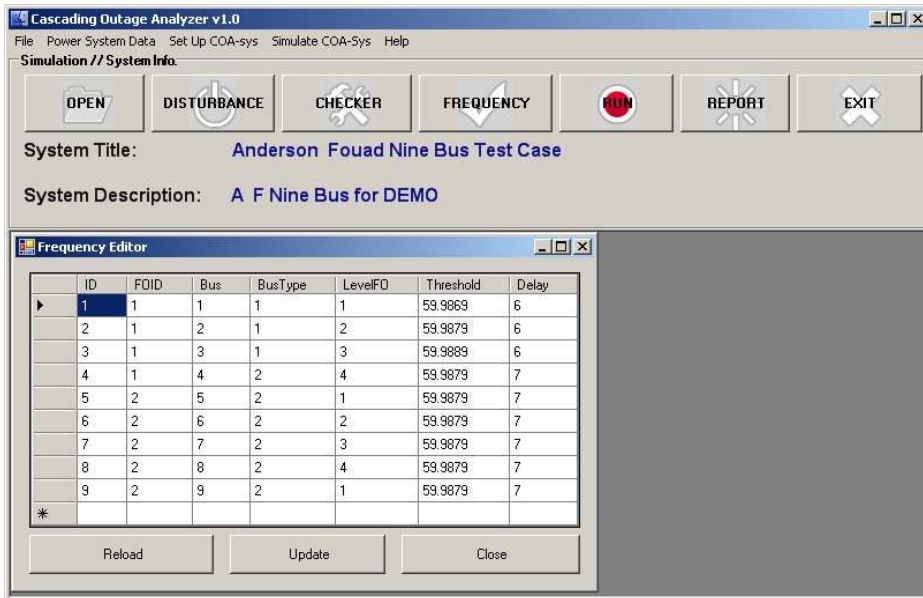


Figure A.34: The characteristics of the frequency checkers used in the first initial disturbance.

Having set the options and the parameters of the simulation the user can now simulate the initial disturbance and view the results of the simulation.

By clicking **Simulate COA-Sys - Run Checker** from the Main menu or the **RUN** icon on the toolbar the user views the *Select Power Flow Algorithm Window* in the Workspace area. For the first initial disturbance applied to the example power system the selected power flow method is the AC Power Flow, as shown in Fig. A.35.

After first clicking **OK** in the message box and then clicking **Run** in the *Select Power Flow Algorithm Window* the simulation starts. At the end of the simulation a message box displaying 'Result: 1' appears in the Workspace area declaring the successful end of the simulation.

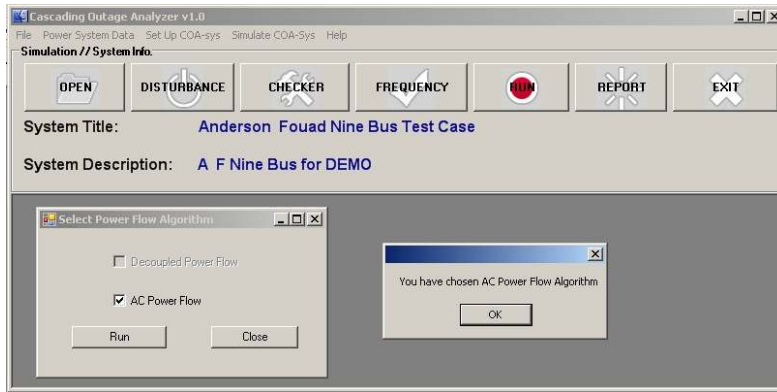


Figure A.35: Power Flow Algorithm Window for the first initial disturbance.

The user can view the results of the first simulation by clicking **Simulate COA-Sys - Simulation Report** from the Main menu or the **REPORT** icon on the toolbar. The *Cascading Outage Report Window* appears in the Workspace area displaying the resulting outages including the initial disturbance in tree view format, as shown in Fig. A.36.

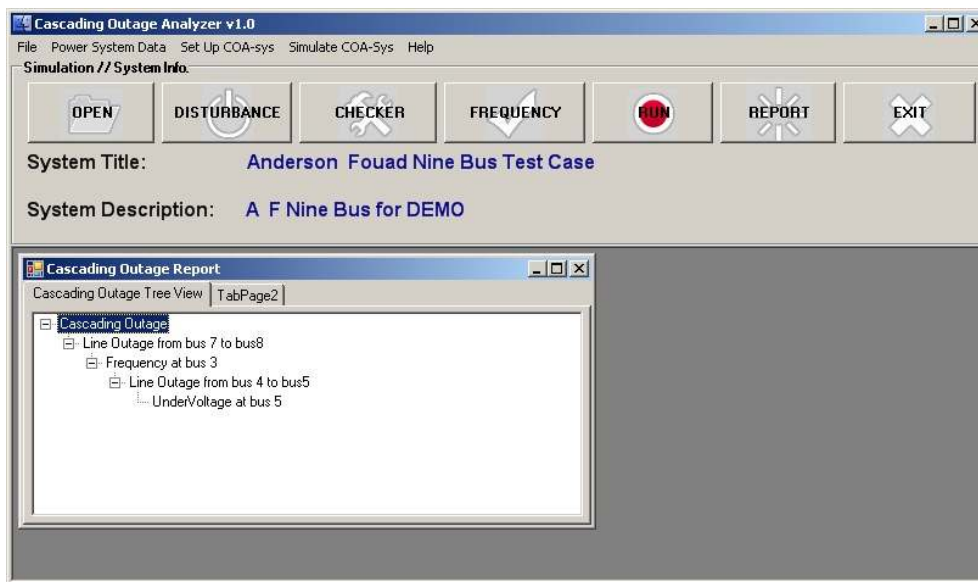


Figure A.36: Simulation Report for the first initial disturbance.

The results of the simulation are presented in chronological order and must be interpreted as follows:

- *Line Outage from bus 7 to bus 8*
This is the initial disturbance applied to the example power system.
- *Frequency at bus 3*
This is the first cascading outage resulting from the initial disturbance. The information displayed means that the settings of the frequency checker at the (generator) Bus 3 are violated causing the opening of the circuit breaker CB 3 and the

outage of generator G3.

— *Line Outage from bus 4 to bus 5*

This is the second cascading outage resulting from the initial disturbance. The information displayed means that the settings of the line overload checker at the line between Buses 4 and 5 (Line 1) are violated causing the opening of circuit breakers CB 5 and CB 7.

— *Under Voltage at bus 5*

This is the third cascading outage resulting from the initial disturbance. The information displayed means that the settings of the under-voltage checker at Bus 5 are violated causing the opening of the circuit breaker CB 9 and the loss of Load A.

The above results are in compliance with the chosen simulation options regarding the order of the outage checkers used. First the settings of the frequency checkers are checked for violation, then the settings of the line overload checkers and finally the settings of the under-voltage checkers. Moreover, the frequency checker whose settings are violated is the one with the highest under-frequency threshold, resulting in the outage of generator G3. The tripping of the overload checkers at Line 1 is also easy to interpret, since this line has the smallest thermal limit. Finally, the outage of Line 1 causes excessive power flow on Line 3 for the supply of Load A, which results in the under-voltage settings at Bus 5 to be violated. The simulation of the events resulting from the first initial disturbance is presented graphically in Fig. A.37.

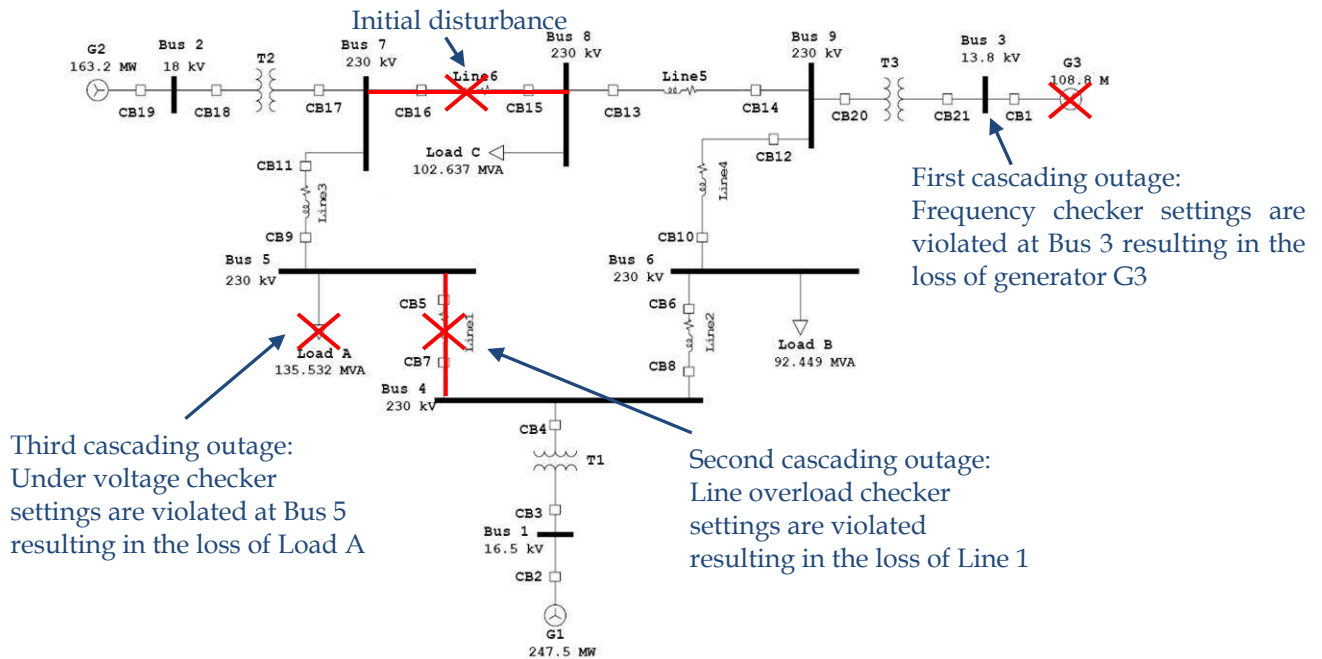
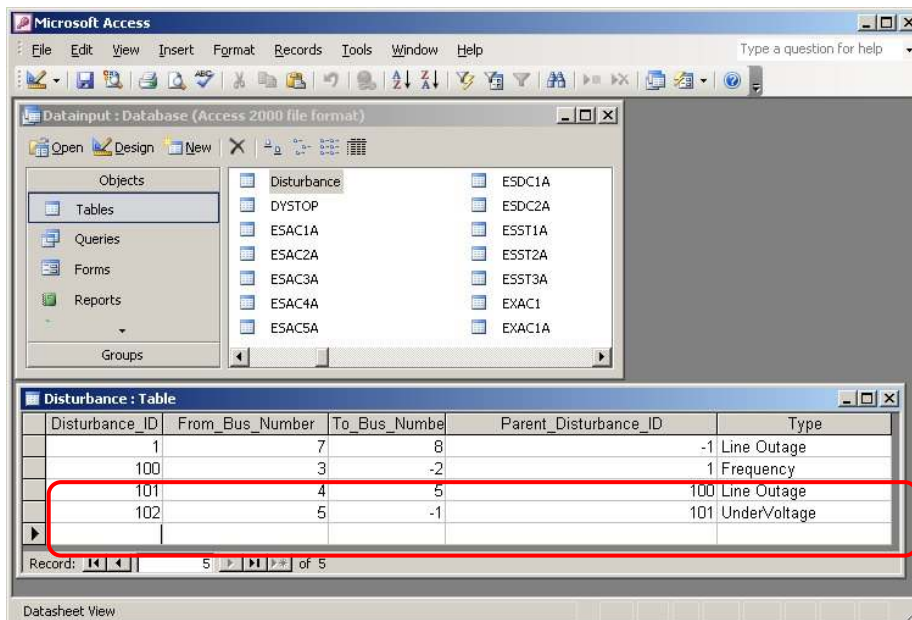


Figure A.37: Graphical representation of the first simulation scenario.

Second Disturbance: Calculations and results

Before the user can simulate a new scenario (even if no changes will be made to the simulation parameters and settings) the following actions must be taken:

- The entries of the table **Disturbance** in the database *Datainput.mdb* located at *C:\project\Dataset\Datainput.mdb* with Disturbance ID 100 and higher must be deleted (all the entries but the first, see Fig. A.38). These entries correspond to the simulation results of the previous simulation which are recorded in this table.
- The table **FOChecker** in the database *COChecker.mdb* located at *C:\project\Dataset\COChecker.mdb* must be deleted.
- The table **FOChecker_org** in the database *COChecker.mdb* located at *C:\project\Dataset\COChecker.mdb* must be copied to the same location and renamed to table **FOChecker**.
- The entries of the table **CTDOutput** in the database *COChecker.mdb* located at *C:\project\Dataset\COChecker.mdb* must be deleted (see Fig. A.39). The simulation results of the frequency checkers are recorded in this table. The entries in the *BusNum* column correspond to the buses of the power system. The entries in the *STD* column display the time-delay settings for each frequency checker while the entries in the *CTD_Under* column show the calculated time duration of the under-frequency below the frequency threshold value. (The frequency checkers can also have over-frequency relays in which case the calculated duration of the over-frequency at any frequency checker will be displayed in the column *CTD_Over*).



Disturbance_ID	From_Bus_Number	To_Bus_Number	Parent_Disturbance_ID	Type
1	7	8	-1	Line Outage
100	3	-2	1	Frequency
101	4	5	100	Line Outage
102	5	-1	101	UnderVoltage

Figure A.38: Lines to be cleared in the Disturbance table before running a new simulation.

The screenshot shows the Microsoft Access interface with the 'CTDOutput : Table' open in Datasheet View. A red box highlights the first nine rows of the table, which correspond to the data in the table below.

ID	BusType	BusNum	Threshold_freq	STD	CTD_Under	CTD_Over
1	1 3		59.9889	6	6.54064270103755	0
2	1 2		59.9879	6		0
3	1 1		59.9869	6		0
4	1 2		59.9879	6	4.54024988750314	0
5	1 1		59.9869	6	2.3071033623081	0
6	1 2		59.9879	6	4.54024988750314	0
7	1 1		59.9869	6	2.3071033623081	0
8	1 2		59.9879	6	4.54024988750314	0
9	1 1		59.9869	6	2.3071033623081	0

Figure A.39: Lines to be cleared in the CTDOutput table before running a new simulation.

After performing the above actions the second scenario can be started. The second initial disturbance to be applied to the example power system simulates the outage of the transformer T3 between Buses 3 and 9, which is equivalent to opening the circuit breakers CB 20 and CB 21, as shown in Fig. A.40.

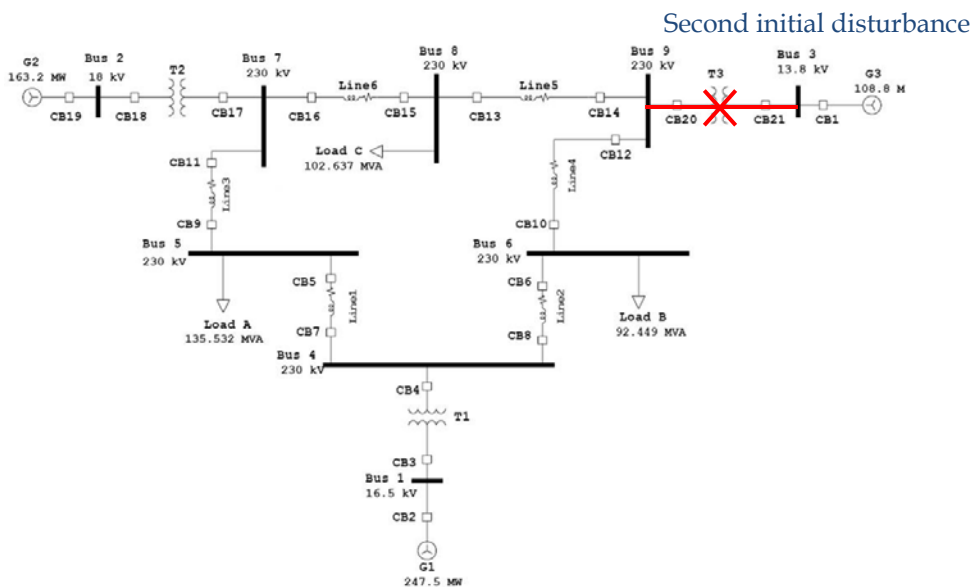


Figure A.40: Second initial disturbance applied to the example power system.

By clicking **Set Up COA-sys - Disturbance** from the Main menu or the **DISTURBANCE** icon on the toolbar the user can view the *Disturbance Editor Window* in the Workspace area. In this Window the user will view the information regarding the previous initial disturbance in the power system. To implement the new scenario the user has to modify the entries in the *From_Bus_Number* and *To_Bus_Number* columns, according to Fig. A.41, by typing in the new values and clicking the **Update** button. It is reminded to the user that the same modifications must be made in the Disturbance table in the database located at *C:\project\Dataset\Datainput.mdb*.

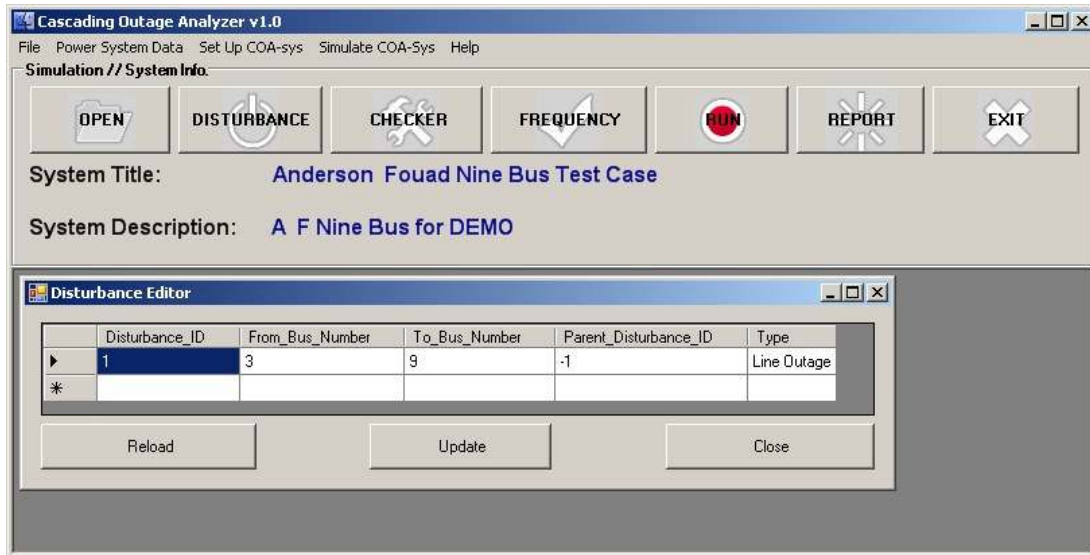


Figure A.41: Window displaying the second initial disturbance for the example power system.

By clicking **Set Up COA-sys - Cascading Outage Checker** from the Main menu or the **CHECKER** icon on the toolbar the user can view the *Overload and Voltage Editor Window* in the Workspace area. For the second initial disturbance two outage checkers will be used, in the following order:

- Line Overload
- Under Voltage.

To implement the new scenario the user has to make the necessary modifications to the entries in the *Overload and Voltage Editor Window*, according to Fig. A.42, by deleting the third row and by clicking the **Update** button. It is reminded to the user that the same modifications must be made in the Outage Checker table in the database located at *C:\project\Dataset\Datainput.mdb*.

Since the frequency checker will not be used there is no necessary action regarding the parameters of this type of checker (**Set Up COA-sys - Frequency Checker** from the Main menu or **FREQUENCY** icon on the toolbar).

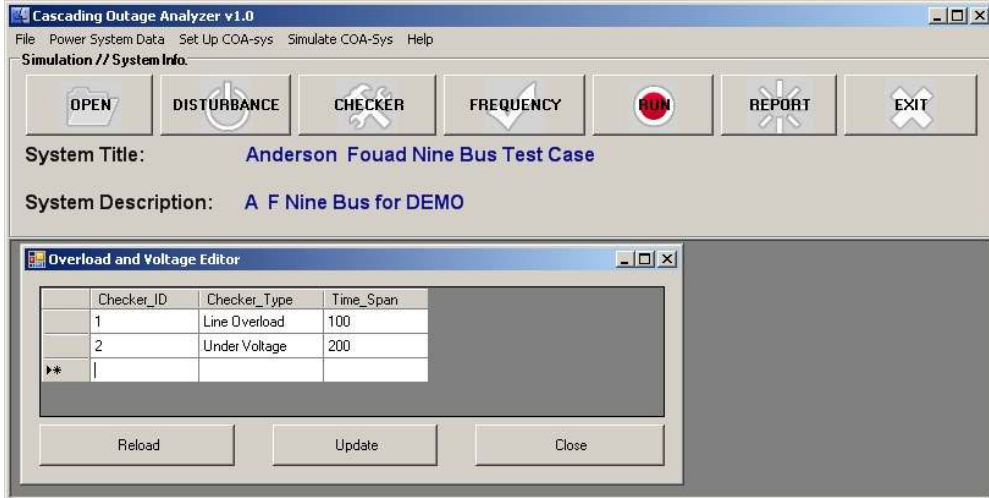


Figure A.43: Window displaying the outage checkers used in the second initial disturbance.

By clicking **Simulate COA-Sys - Run Checker** from the Main menu or the **RUN** icon on the toolbar the user views the *Select Power Flow Algorithm Window* in the Workspace area. For the second initial disturbance applied to the example power system the selected power flow method is the Decoupled Power Flow, as shown in Fig. A.44.

After first clicking **OK** in the message box and then clicking **Run** in the *Select Power Flow Algorithm Window* the simulation starts. At the end of the simulation a message box displaying 'Result: 1' appears in the Workspace area declaring the successful end of the simulation.

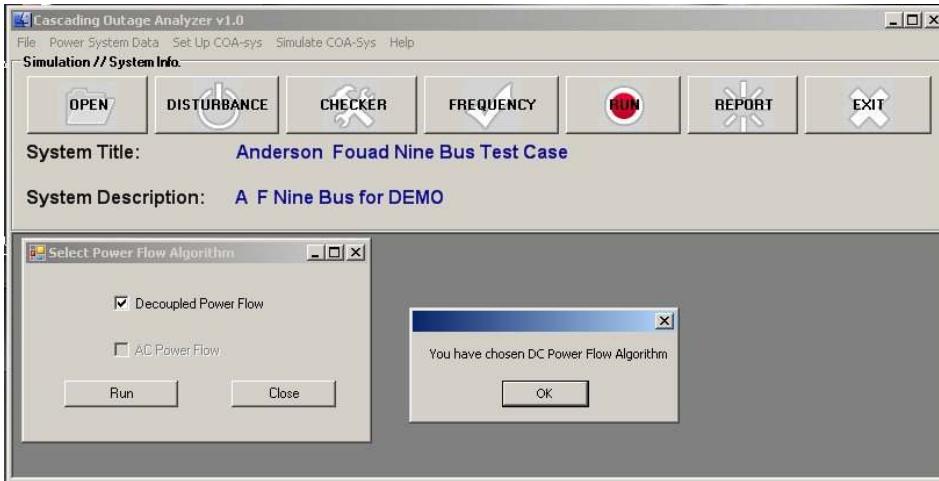


Figure A.44: Power Flow Algorithm Window for the second initial disturbance.

The user can view the results of the first simulation by clicking **Simulate COA-Sys - Simulation Report** from the Main menu or the **REPORT** icon on the toolbar. The *Cascading Outage Report Window* appears in the Workspace area displaying the resulting outages including the initial disturbance in tree view format, as shown in Fig. A.45.

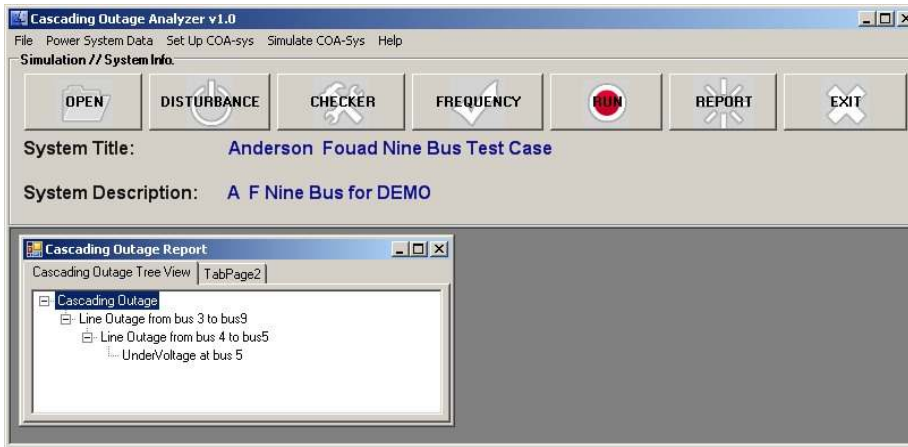


Figure A.45: Simulation Report for the second initial disturbance.

The results of the simulation are presented in chronological order and must be interpreted as follows:

- *Line Outage from bus 3 to bus 9*
This is the initial disturbance applied to the example power system, representing the outage of a transformer.
- *Line Outage from bus 4 to bus 5*
This is the first cascading outage resulting from the initial disturbance, which is the outage of Line 1.
- *Under Voltage at bus 5*
This is the second cascading outage resulting from the initial disturbance, which results in the loss of Load A.

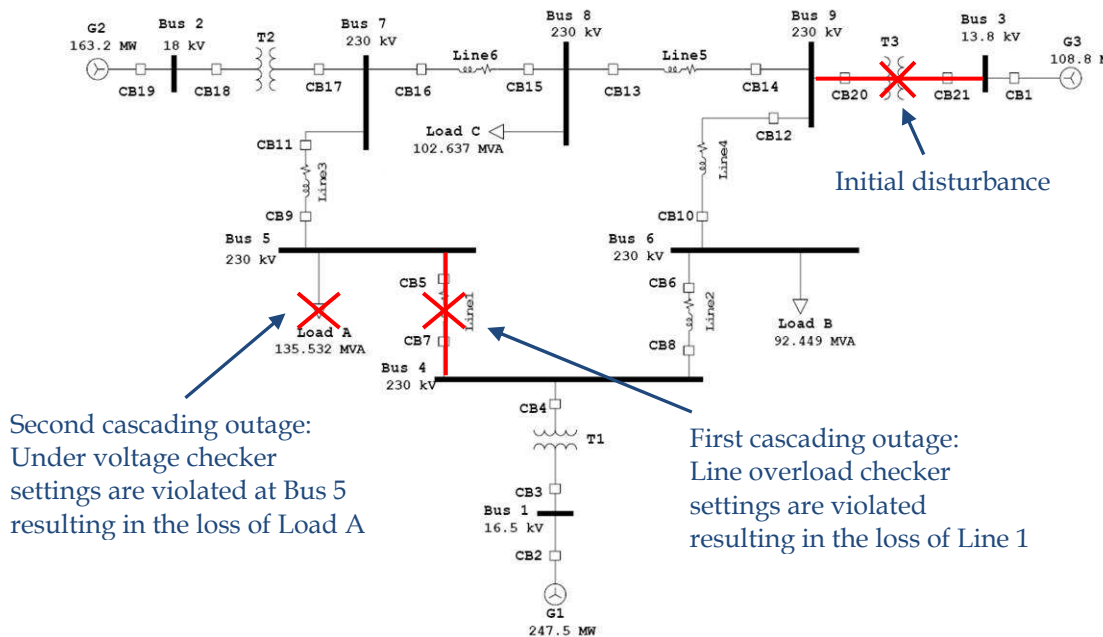


Figure A.46: Graphical representation of the second simulation scenario.

Again, the above results are in compliance with the chosen simulation options regarding the order of the outage checkers used. First the settings of the line overload checkers are checked for violation and then the settings of the under-voltage checkers. The tripping of the overload checkers at Line 1 is due to the fact that this line has the smallest thermal limit. The outage of Line 1 causes again excessive power flow on Line 3 for the supply of Load A, which results in the under-voltage settings at Bus 5 to be violated, and in the loss of Load A. The simulation of the events resulting from the second initial disturbance is presented graphically in Fig. A.46.

User's Manual Index

bus data		menu bar	56
<i>exit</i>	61	options	
<i>view</i>	61	<i>disturbance</i>	64
COA application		<i>frequency checker</i>	66
<i>exit</i>	60	<i>outage checkers</i>	65
<i>installation</i>	55	<i>power flow algorithm</i>	67
<i>start</i>	55	outage checker	
<i>user interface</i>	56	<i>edit</i>	65
data		<i>exit</i>	65
<i>buses</i>	61	<i>view</i>	65
<i>generators</i>	61	power system	
<i>lines</i>	62	<i>data</i>	61
<i>loads</i>	63	<i>new</i>	71
disturbance		<i>open</i>	59
<i>edit</i>	64	report	
<i>exit</i>	64	<i>interpretation (first simulation)</i>	77
<i>view</i>	64	<i>interpretation (second simulation)</i> ..	83
frequency checker		requirements	
<i>edit</i>	66	<i>hardware</i>	55
<i>exit</i>	66	<i>software</i>	55
<i>view</i>	66	simulation	
generator data		<i>end</i>	68
<i>exit</i>	62	<i>new</i>	79
<i>view</i>	61	<i>options</i>	64
line data		<i>report (tree view)</i>	68
<i>exit</i>	62	<i>run</i>	67
<i>view</i>	62	system information bar	57
load data		toolbar	57
<i>exit</i>	63	workspace area	58
<i>view</i>	63		

References

- [1] R. Baldick and M. H. Joung, “Progress Report on DOE Research Project DE-AI02-05ER256: “Reducing the Vulnerability of Electric Power Grids to Terrorist Attacks, December 2006
- [2] Commonwealth Associates, Inc. “White Paper: A Scenario Describing the August 14, 2003 Blackout,” September 9, 2003
- [3] E. Gamma, R. Helm, R. Johnson, J. Vlissides “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley Longman Publishing Co., Inc., 1995
- [4] J. Keogh, “Visual Basic 2005: your visual blueprint for writing dynamic applications”, Wiley Publication Inc., 2006
- [5] A. Stellman and J. Greene, “Head First C#: A Brain-Friendly Guide”, O’Reilly Media Inc., 2008
- [6] Commonwealth Associates, Inc. “Transmission 2000 Power Flow Program, Power flow Version 5.10 Package”, April 2008
- [7] P. M. Anderson and M. Mirheydar “A Low-Order System Frequency Response Model”, IEEE Transactions on Power Systems, Vol. 5, No. 3, August 1990
- [8] P. M. Anderson and A. A. Fouad, “Power System Control and Stability”, Vol. 1, The IOWA state University Press, Ames, USA, 1977