NTNU – Trondheim
Norwegian University of
Science and Technology

Efficient Power Flow Algorithms for Risk Management in Electrical Power Systems

Gaute Roska

Master of Energy and Environmental Engineering
Submission date: June 2015
Supervisor: Olav B Fosso, ELKRAFT
Co-supervisor: Robert Fossmark Pedersen, Goodtech
Trond Tollefsen, Goodtech

Norwegian University of Science and Technology
Department of Electric Power Engineering

Problem Description

Efficient Power Flow Algorithms for Risk Management in Electrical Power Systems

In a specialization project in the autumn 2014, techniques for efficient load flow calculations have been investigated. A number of alternative techniques have been tested either by using available tools and partly by own programming in MATLAB.

This work should be continued using more realistic test systems and by this develop efficient strategies for identifying cases needing detailed evaluation. Especially the strategies and conclusions of the different versions of decoupled load flow should be further investigated. An alternative strategy to do studies on complete network structures and reduce number of cases, can be to establish equivalents of the parts of the systems that are not affected by the outages and by this reduce size to allow a larger number of cases to be evaluated. Such efficient equivalents can be based on power transfer distribution factors and will be an exact representation of the area as long as the topology and injection profile within the area is unchanged.

Special emphasis should be put on comparison of methods and verification of results.

Supervisor

Prof. Olav Bjarte Fosso

Preface

My Master of Science in Electrical Power Engineering has come to an end, June of 2015. This thesis will be my final work at Norwegian University of Science and Technology.

Programming has been an important part of the thesis, and it has been a time-consuming process. However, I feel my programming skills have increased during the work with this thesis.

I wish to thank my supervisor Professor Olav Bjarte Fosso at the Department of Electrical Power Engineering. He has given me valuable guidance and motivated me.

I would also like to thank my co-supervisor, Robert Fossmark Pedersen. He has offered me help with the MATLAB-code and he has also motivated me during the work process. I have learnt a lot by study his work at this field of research.

I also wish to thank my family. Their support has been important to me during the challenging, but instructive years of study at NTNU.

- Gaute Roska -

Abstract

The society depends on a reliable electrical power system. The socio-economic cost will be high if a wide-ranging disruption of the supply occurs over an extended period of time. Thus programs that help system operators to identify possible risk in electrical power systems are needed.

The fast decoupled power flow, primal version, is found to have better convergence characteristic than the dual one, for power systems with a low X/R-ratio.

Linear sensitivity factors have been utilized in several of the algorithms. These factors are based upon DC and can be used to find post-contingency flows very fast, hence useful for decision support.

Contingency analysis is important in this report. The purpose of contingency analysis is to test systematically for outage of components like branches and generators. Most of the implemented code is contingency analysis algorithms.

The efficient bounding method has been implemented. This method encircles the outage component and thus reduces the number of calculations needed. The number of expansion steps needed from the outage, are dependent on an angle spread criterion concerning the boundary nodes. EBM is found to be faster than normal DC power flow. The concentric relaxation method is a simpler variant of the EBM. It utilizes a pre-determined number of layers to enclose an outage. The EBM and the CRM are best used when only the largest flow violations are of interest. Both the EBM and CRM are DC algorithms.

A post-compensation technique has been utilized for some of the algorithms, both AC and DC. This technique makes it possible to avoid time consuming factorization of matrices, thus allow for more calculations during a limited time frame. The post-compensation technique gives a more detailed output, and is faster than the EBM and the CRM. Even the AC-version used less time than the EBM when testing a large power system.

Indirect ranking of contingencies using a performance index has been implemented in the DC contingency analysis algorithms. A modification is done to the method so that only overloaded branches contribute to the index; hence it prevents the phenomena known as "masking error".

Efficient power flow algorithms are useful when time frame is limited. However, it is important to know the different algorithms' limitations. DC algorithms, for instance, should not be used for systems with a low X/R-ratio.

Samandrag

Samfunnet er avhengig av eit påliteleg kraftsystem. Den sosioøkonomiske kostnaden vil vere høg dersom ein omfattande driftsstans av levert kraft inntreff over ein lengre tidsperiode. Derfor er program, som hjelper systemoperatørane med å identifisere mulig risiko i elektriske kraftsystem, nødvendige.

Rask fråkopla kraft flyt, primal versjon, viser seg å ha betre konvergenskarakteristikk enn dual versjonen for kraftsystem med eit lågt X/R-forhold.

Lineære sensitivitetsfaktorar har blitt nytta i fleire av algoritmane. Desse faktorane er basert på DC, og kan brukast til å finne oppdatert kraft flyt veldig raskt, og er derfor nyttige som brukarstøtte.

Eventualitetsanalyse er viktig i denne rapporten. Føremålet med eventualitetsanalyse, er å teste systematisk for utfall av komponentar som kraftliner og generatorar. Mesteparten av den implementerte koden er eventualitetsanalyse algoritmar.

Den "effektive grense metoden" (EGM) har blitt implementert. Denne metoden sirklar inn komponenten som har driftstans, og reduserer dermed talet berekningar som trengst . Talet ekspansjonssteg som trengst frå utfallskomponenten, er avhengig av eit vinkelspreiing kriterium som omfattar grensenodar. EGM er funne å vere raskare enn normal likestrøm kraftflyt. Den "konsentriske avslappings metoden" (KAM) er ein enklare variant av EGM . Den nyttar eit førehandsbestemt tal lag for å sirkle inn eit utfall. EGM og KAM er best brukt når berre dei største overbelastningane er av interesse . Både EGM og KAM er DC-algoritmar.

Ein etterkompensasjons teknikk har vorte brukt i nokre av algoritmane, både for AC og DC . Denne teknikken gjer det mulig å unngå tidkrevjande faktorisering av matriser, og gjer det på den måten mulig med fleire berekningar i løpet av ein avgrensa tidsperiode . Etterkompensasjons teknikken gir ei meir detaljert utskrift, og er raskare enn EGM og KAM. Sjølv AC-versjonen brukte mindre tid enn EGM når eit stort kraftsystem vart testa.

Indirekte rangering av eventualitetar ved hjelp av ein ytingsindeks, har blitt implementert i DC Eventualitetsanalyse algoritmar . Ein modifikasjon er gjort i framgangsmåten, slik at berre belasta kraftlinjer bidrar til indeksverdien, såleis forhindrar det fenomenet kjent som "maske feil".

Effektive kraft flyt algoritmar er nyttige når tilgjengeleg tid er avgrensa. Likevel er det viktig å kjenne til dei forskjellige algoritmane sine avgrensingar. DC-algoritmar, til dømes, bør ikkje nyttast til system med eit lågt X/R-forhold.

List of Contents

List of Figures

List of Tables

Nomenclature

FDPF	=	fast decoupled power flow
NRPF	=	Newton-Raphson power flow
DCPF	=	direct current power flow
ACPF	=	alternate current power flow
PI	=	performance index
EBM	=	efficient bounding method
CRM	=	concentric relaxation method
PTDF	=	power transfer distribution factors
LODF	=	line outage distribution factors
SF	=	sensitivity factors
PF	=	power flow
MB	=	Megabytes
PU	=	per unit
SDCCCA	=	single DC Contingency Compensation Algorithm
SACCCA	=	single AC Contingency Compensation Algorithm
DDCCCA	=	double DC contingency compensation algorithm
GOA	=	generator outage algorithm
branch, line	=	both used for power line
node, bus	=	both used for connection point in power system
(1 to 5)	=	from bus 1 to bus 5
s, ms	=	seconds, milliseconds

1 Introduction

.

Electric power systems are perhaps one of the most complex systems operated by humans. The generation of electric energy has to equal the consumption at any time and the voltage and frequency level should be kept constant during the operation. The load demand fluctuates all the time and control actions makes sure the generation equals the consumption Expansion of a power system is time-consuming and expensive. The process involves building, planning, negotiation with landowners and environment considerations. However, due to population growth and economic expansion, consumption of electricity is growing. Hence, the power system is operated closer to its limits [1].

In order to prevent blackouts and wide scale load disconnections, power systems need to be operated with a sufficient security margin. Hence the transmission grid should be able to handle the change in flow patterns due to outage of a line. In addition, there should be enough reserve generation capacity to replace the failure of generating units, and [2].

The probability of two or more independent failures or faults taking place simultaneously is often assumed too low to be considered credible. This is a standard assumption when it comes to operating a power system. Therefore a system should be able to withstand the loss of any single component. A power system which is run like this is "N-1 secure" and if it is able to withstand the failure/loss of two components, it is "N-2 secure" [2].

Electric energy is fairly cheap and almost always available and the modern society depends on a reliable supply of electric energy. Chaos would likely be the outcome, should the supply of electric energy fail for an extended period of time [1].

Power system operators are to run the power system in the most economical manner. Hence, the electrical energy should be bought or produced at the lowest possible cost, and the amount of transactions should be maximized in order to utilize the capacity. This cause more power to be transmitted over long distances and the grid's stress level increases. Thus balancing the greed for profit and the fear of blackouts is the essence of operating a power system. A power system should on all occasions be operated in a way which ensures that no credible contingency could trigger cascading outages or another type of instability. Lightning strikes on a transmission grid system and other unpredictable faults and failures are unavoidable. [2].

It is important to have appropriate software which can monitor a power system's robustness and identify possible risk, because the society depends on a reliable electrical power system. PROMAPS[1] is an example of software that optimizes the operation, and analyse fault combinations. It can be used both in real time or offline as a planning tool. [23].

Chapter 2 contains some theory about power system reliability, linear sensitivity factors and some other important methods and techniques.

Chapter 3 gives a brief explanation about important mistakes done in the project report.

Chapter 4 explains the MATLAB-code for some important algorithms. The purpose of this chapter is not to explain every detail and variable, but rather important features and some of the algorithms' output.

Chapter 5 contains information about the different power systems which have been used for testing the algorithms. There is on small system, a medium and a large one.

In chapter 6 the results are presented and there is a concise discussion about the main findings, while Chapter 7 gives an overall discussion.

Chapter 8 concludes the results, and chapter 9 gives a suggestion for further work for this field of research.

The derivation of the linear sensitivity factors is carried out in Appendix A, while Appendix B contains the MATLAB-code.

[1] PROMAPS - Probability and reliability methods applied to power systems

2 Theory

2.1 System Security

The theory of "system security" is generally based upon Wollenberg [18].

It is important to maintain system security when operating a power system. It is stated that: "System security involves practices designed to keep the system operating when components fail" [18]. By maintaining proper amounts of spinning reserve, the remaining units in the system can make up the deficit without too low a frequency drop or disconnecting of load. A transmission line may, for instance, be damaged by a storm and taken out by automatic relaying. Then with proper actions like re-dispatching of generation, flow limits for transmission lines are not exceeded because the remaining ones can share the increased loading and still remain within limits.

All equipment in a power system can be disconnected from the network. The reasons for disconnections are generally divided into two categories: Scheduled outages and forced outages:

Scheduled outages are typically done to perform maintenance or replacement of the equipment, and it is scheduled by operators to minimize the impact on the reliability of the system.

Forced outages are those that happen at random and may be due to internal component failures or outside influences such a lightning, wind storms, ice build-up, etc.

Because of the unpredictable nature of forced outages, the system must be operated continuously in a way that prevents dangerous condition if any credible outage is to occur. Since power system equipment is designed to be operated within certain limits, most of the equipment is protected by automatic devices. Such devices are able to switch out equipment if limits are violated. If a forced outage occurs and limits are violated on other components, the event may be followed by a series of further actions, for instance switching other equipment out of services. The entire system may collapse if this process of cascading failures continues. This is usually referred to as a **system black out** [2].

N-1 refers to a system with n components, and n-1 is its state with one component out. The n-1 criterion states that no single outage will result in other components experiencing flow or voltage limit violations. Most large power systems install equipment to allow operations personnel to monitor and operate the system in a reliable way.

System security can be broken down into three major functions that are carried out in an operations control central:

1**. System monitoring**

2. **Contingency analysis**

3. **Security-constrained optimal power flow**

System monitoring provides the operators of the power system with pertinent up-to-date information on the conditions on the power system. It is the most important function of the three. System of measurements and data transmission is called energy management systems, EMS. These systems have evolved to schemes that can monitor voltages, currents, power flows, and the status of circuit breakers and switches in every substation in a power system transmission network. In addition, other critical information such as frequency, generator unit outputs, and transformer tap positions can also be telemetered.

Power system operators have to deal with a power system in one of four modes:

- **Normal**
- **Alert**
- **Emergency**
- **Restoration**

Normal usually means that there are no alarms being presented and contingency analysis is not reporting contingencies that would cause overloads or voltage violations.

Alert means that either an alarm has been presented to the operator or the contingency analysis programs have presented the possibilities of a contingency problem.

Emergency would indicate serious alarm messages that the operators must act on immediately and threaten to cause major shutdowns of power system equipment or even parts of the system.

Restoration comes if the system loses equipment or part of the system or even most of it shut down or blacked out. Equipment must be investigated to see if it can be brought back on line and then switched back into the system. Loads that were dropped are brought back on line, sometimes in small blocks. Restoration can take many hours especially if large generators are involved.

Contingency analysis is the second major security function. The result of this type of analysis allows system to be operated defensively. Many of the problems that occur on a power system can cause serious trouble within such a quick time period that the operator cannot take action fast enough once process is started. This is often the case with cascading failures. Because of this aspect of system operation, modern operations computers are equipped with contingency analysis programs the model possible system troubles before they arise. These programs are based on a model of the power system and are used to study outage events and alarm the operators to any potential overloads or out-of-limit voltages.

Optimal power flow is the third major security function. In this function, a contingency analysis is combined with an optimal dispatch of generation, so when a security analysis is run, no contingencies result in violations. It is appropriate to divide the power system into four operating objectives:

Normal state dispatch: This is the state that the power system is in prior to any contingency. It is optimal with respect to economic operation, but it may not be secure.

Post-contingency: This is the objective after a contingency has occurred. This can for example be overvoltage or transformer beyond its flow limit.

Secure dispatch: This is the objective with no contingency outages is to correct the operating parameters to account for security violations.

Secure post-contingencies: The objective is to re-mediate the contingency as applied to the base-operating condition with corrections.

2.2 Factors Affecting Power System Security

The theory of "factors affecting power system security" is generally based upon Wollenberg [18].

As a consequence of many widespread blackouts in interconnected power systems, the priorities for operation of modern power systems have evolved to the following: "Operate the system in such a way that power is delivered reliably. Within the constraints placed on the system operation by reliability considerations, the system will be operated most economically" [18].

It is impossible to build a power system with so much redundancy that failures never cause load to be dropped on a system. Therefore systems are designed in a way that the probability of dropping load is acceptably small. Thus, most power systems are designed to have sufficient redundancy to withstand all major failures events.

The operators' task is to try maximizing the reliability of the system within the design and economic limitations continuously. A power system is usually never operated with all equipment connected due to maintenance or failures. Transmission-line failures cause changes in the flows and voltages on the transmission equipment remaining connected to the system. Therefore, the analysis of transmission requires methods to predict these flows and voltages so as to be sure they are within their respective limits. Generation failures can also cause flows and voltages to change in the transmission system, with the addition of dynamic problems involving system frequency and generator output. When generation is lost, much of the made up power will come from tie lines, and this can mean line flow limit or bus voltage limit violations. The system must monitor two things to be sure generator outages do not cause problems when one is lost: Check spinning reserve at all times to be sure it is adequate and model generator outages and their effect on transmission flows and voltages.

When a transmission line or transformer fails and is disconnected, the flow on that line goes to zero and all flows nearby will be affected. The result can be a line flow or limit or bus voltage limit violation. The operators therefore usually want to check as many of them as possible, as frequently as possible.

An even more difficult analysis is to check all pairs of possible simultaneous outages, which is denoted (N-2). Thus, all pairs of generators, and all pairs of transmission lines as well as pairs of single generator outages plus a possible single transmission-line outage at the same

time would have to be analysed. This (N-2) analysis is much more difficult because of the extremely large number of cases to model. The usual practice is to only study a few of the (N-2) cases that are known by experience to be the most serious cases.

The most difficult methodological problem to cope with in contingency analysis is the speed of solution of the model used. The most difficult logical problem is the selection of "all credible outages". If each outage case studied were to solve in 1 s and several thousand outages were of concern, it would take close to 1 h before all cases could be reported. This would be useful if the system conditions did not change over that period of time. However, power systems are constantly undergoing changes and the operators usually need to know if the present operation of the system is safe, without waiting too long for the answer. Contingency analysis execution times of less than 1 min for several thousand outages cases are typical of computer and analytical technology as of 2014 [18].

An approximate model of the power system can be used to gain speed of solution in a contingency analysis procedure. Various DC power flow models provide adequate capability for many systems. For other systems, voltage is a concern and full AC power flow analysis is required.

2.3 An overview of security analysis

The theory of "an overview of security analysis" is generally based upon Wollenberg [18].

A security analysis study that is run must be executed very quickly in order to be of any use to operators. There are three basic ways to accomplish this [18]:

1. **Study the power system with approximate but very fast algorithms.**
2. **Select only the important cases for detailed analysis**.
3. **Use a computer system made up of multiple processors or vectors processors to gain speed.**

The first method has been in use for many years and goes under various names such as "linear sensitivity methods" or "DC power flow methods". This approach is useful if one only desires an approximate analysis of the effect of each outage. These methods are also called linear sensitivity factors. It has all the attributed to the DC power flow, only branch MW flows are calculated and these are only within about 5% accuracy.

If it is necessary to know a power system's MVA flows and bus voltage magnitudes after a contingency outage, then some form of complete AC power flow must be used. This becomes a problem when thousands of cases must be checked during a short time frame. It is impossible to execute thousands of complete AC power flows when time is limited, but most of the cases result in power flow results that do not have flow or voltage limit violations. **Contingency selection** or **contingency screening** eliminates most of the non-violation cases and only run complete power flow on the "critical" cases.

There are ways of running thousands of contingency power flows if special computing facilities are utilized. This involves many processors running separate cases in parallel or the use of vector processors.

2.4 Power Flow Algorithms

The power flow algorithms were explained thoroughly in the project report [24]. Only a brief presentation will be given of DCPF and FDPF, based on [5]. These are essential power flow algorithms which are implemented in this report.

2.4.1 Direct Current Power Flow

DCPF is a non-iterative, linear power flow algorithm. The voltages are assumed flat at 1 PU, phase angles are assumed small enough to neglect and the power system are modelled as loss-free. The **B'**-matrix is built in a similar way as the bus admittance matrix, but the resistance elements are neglected and the imaginary operator "j" is removed.

$$\boldsymbol{B'\delta} = \mathbf{P} \tag{2.1}$$

$$\boldsymbol{\delta} = \boldsymbol{B'^{-1}P} \tag{2.2}$$

The active power flow on a transmission line is estimated by the angle difference divided by the lines' reactance. The DCPF is reliable in high voltage systems with large X/R

$$P_{ij} = \frac{\delta_i - \delta_j}{X_{ij}} \tag{2.3}$$

2.4.2 Fast Decoupled Power Flow

For high-voltage power systems, the change in active power is mostly connected to the power angles while reactive power is most connected to voltage magnitude. This physical decoupling phenomenon can be written as:

$$\begin{bmatrix}\Delta \boldsymbol{P} \\ \Delta \boldsymbol{Q}\end{bmatrix} = \begin{bmatrix}\boldsymbol{J_1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{J_4}\end{bmatrix}\begin{bmatrix}\Delta \boldsymbol{\delta} \\ \Delta |\boldsymbol{V}|\end{bmatrix} \tag{2.4}$$

Multiplying the corresponding elements in the matrix above gives the two equations:

$$\Delta \boldsymbol{P} = \boldsymbol{J_1}\Delta \boldsymbol{\delta} = \left[\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{\delta}}\right]\Delta \boldsymbol{\delta} \tag{2.5}$$

$$\Delta \boldsymbol{Q} = \boldsymbol{J_4}|\boldsymbol{V}| = \left[\frac{\partial \boldsymbol{Q}}{\partial |\boldsymbol{V}|}\right]\Delta |\boldsymbol{V}| \tag{2.6}$$

By certain simplifications, for example voltage magnitudes are assumed close to 1 PU, the phase angle - and voltage magnitude vector can be written as:

$$\Delta \boldsymbol{\delta} = -[\boldsymbol{B'}]^{-\boldsymbol{1}}\frac{\Delta \boldsymbol{P}}{|\boldsymbol{V}|} \tag{2.7}$$

$$|\boldsymbol{V}| = -[\boldsymbol{B''}]^{-\boldsymbol{1}}\frac{\Delta \boldsymbol{Q}}{|\boldsymbol{V}|} \tag{2.8}$$

$\boldsymbol{B'}$ and $\boldsymbol{B''}$ are constant elements found by some altering of the bus admittance matrix. These two matrices need only to be triangularized and inverted once in the iteration process.

A detailed treatment of the FDPF can be found in Stott [6], Amerongen [7] and Monticelli [8].

2.5 Concentric Relaxation Method

The concentric relaxation method is based upon the assumption that an outage has only a limited geographical impact. The power system is divided into an affected part and an unaffected one. The buses at the end of the interrupted line are marked as layer zero. Further, buses that are one transmission line or transformer from layer zero are labelled layer one. This process can be done until all the buses of the network are included in layers. Then a number of layers is chosen and all buses within the chosen layer and lower are solved as a power

flow. The rest of the buses in the higher layers are kept as reference buses with constant voltages and phase angles [17].

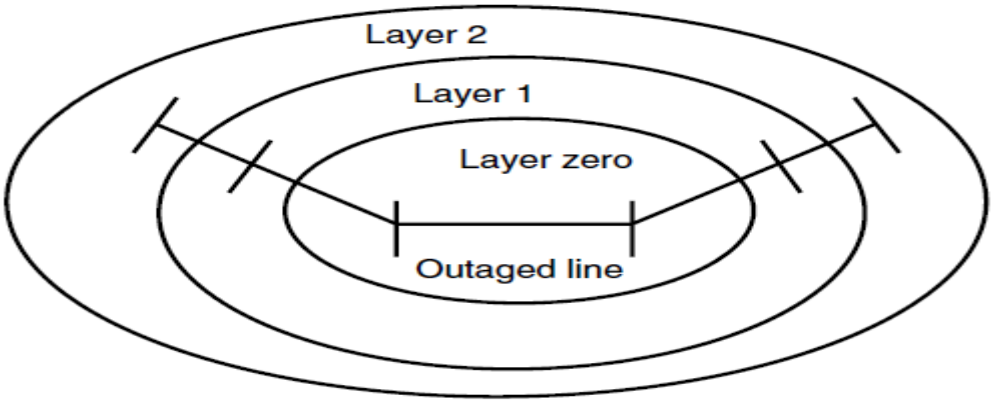

Figure 1: Concentric Relaxation Method [17]

The concentric relaxation method will require more layers for circuits whose influence is felt further from the outage.

2.6 Efficient Bounding Method

The efficient bounding method is more advanced than the CRM. This method uses an adjustable region around the outage. The aim of this method is to as few calculations as possible to determine the impact of an outage. The power system is divided into three subsystems [17]:

- N1: The subsystem immediately surrounding the outaged line.
- N2: The external subsystem that we shall not solve in detail.
- N3: The set of boundary buses that separate N1 & N2.

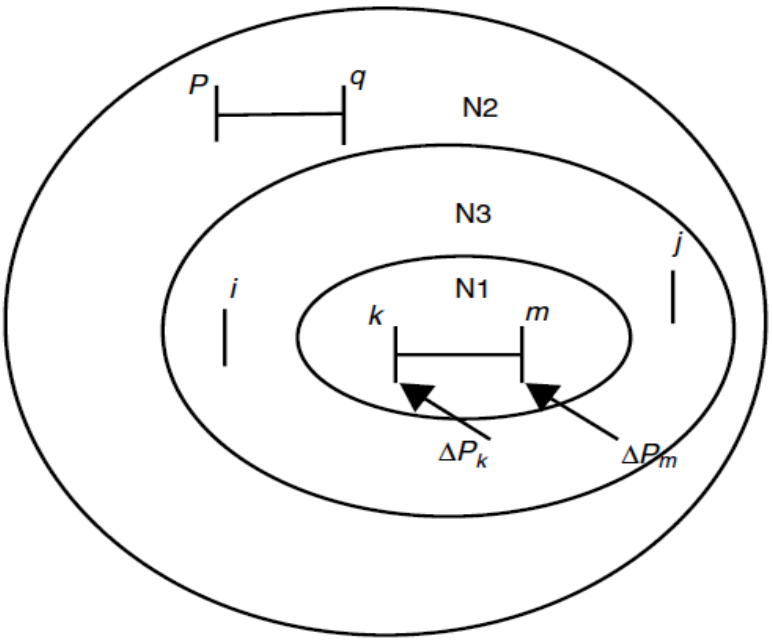

Figure 2: The efficient bounding method [17]

The subsystem N1 contains at minimum, the nodes k and m. It can be formed by expanding the network around the outage for a number of tiers, normally 3 or 4, or by a topological search for a closed path around. The subsystem N3 consist of the boundary nodes that separate subsystem N1 from nodes in subsystem N2. The effect of the branch outage is modelled by a pair of appropriately scaled injections: ΔP at node k and $-\Delta P$ at node m. The algebraic sum of all incremental flows across the set of boundary nodes is equal to zero since the subsystems N2 and N3 are passive and there are no paths to ground [9].

The linear sensitivity factors are essential in the EBM and the analysis is performed using the sparse LU-factors of the real, symmetric matrix $\boldsymbol{B}'$.

The linear incremental power flow model is similar to DCPF but it focus on the incremental changes in the system. For a system of n buses it can be expressed as:

$$\boldsymbol{B}'\Delta\theta = \Delta P \tag{2.9}$$

$\boldsymbol{B}'$: n x n susceptance matrix.

$\Delta\theta$: n-vector of changes in bus angle.

ΔP: n-vector of changes in real power injections.

The equation above is best used in systems with high X/R ratios. The impact of each contingency on MW flows can be found by solving for $\Delta\theta$, and calculating the resulting changes in branch flows ΔP_{km} from the equation:

$$\Delta P_{km} = (\Delta\theta_k - \Delta\theta_m)/x_{km} \tag{2.10}$$

$\Delta\theta_k, \Delta\theta_m$: Changes in bus voltage angels at nodes *k* and *m*.

x_{km}: Reactance of branch *km*.

The post-contingency branch flow is computed as:

$$P_{km} = P_{km}^0 + \Delta P_{km} \tag{2.11}$$

P_{km}^0: The pre-contingency branch MW flow.

P_{km} can then be compared with the branch flow limit to determine whether it is a violation.

The derivation of the method for a single-branch outage is shown below. This derivation can be extended to any kind of topological change no matter how complicated.

Suppose a transmission line in N2 carries a flow of f_{pq}^0 , then there is a maximum amount that the flow on pq are permitted to shift. Thus, it can increase to an upper limit or a lower limit.

$$\Delta f_{pq}^{max} = min\left[(f_{pq}^{+} - f_{pq}^0), (f_{pq}^0 - f_{pq}^{-})\right] \tag{2.12}$$

This can be transformed into a maximum change in phase angle difference:

$$f_{pq} = \frac{(\theta_p - \theta_q)}{x_{pq}} \tag{2.13}$$

$$\Delta f_{pq} = \frac{(\Delta\theta_p - \Delta\theta_q)}{x_{pq}} \tag{2.14}$$

$$(\Delta\theta_p - \Delta\theta_q)^{max} = \Delta f_{pq}^{max} x_{pq} \tag{2.15}$$

Brandwajn [9] develops a theorem concerning the maximum change in the phase angle difference across pq:

$$\left|\Delta\theta_p - \Delta\theta_q\right| < \left|\Delta\theta_i - \Delta\theta_j\right| \tag{2.16}$$

Where i and j are any pair of buses in N3, the boundary region. $\Delta\theta_i$ is the largest $\Delta\theta$, and $\Delta\theta_j$ the smallest $\Delta\theta$. $\left|\Delta\theta_i - \Delta\theta_j\right|$ provides an upper limit to the maximum change in angular spread across any circuit in N2. By combining the two former equations above, a new expression is formed:

$$\Delta f_{pq}^{max} x_{pq} \ < \left|\Delta\theta_i - \Delta\theta_j\right| \tag{2.17}$$

All circuits in N2 are safe from overload if the value of $\left|\Delta\boldsymbol{\theta_i} - \Delta\boldsymbol{\theta_j}\right|$ is less than the smallest value of $\Delta\boldsymbol{f_{pq}^{max}x_{pq}}$ over all pairs $\boldsymbol{pq}$, where $\boldsymbol{pq}$ corresponds to the buses at the ends of circuits in N2.

If this condition fails, then N1 must be expanded and a new value of $\left|\Delta\theta_i - \Delta\theta_j\right|$ in N3. Then the test of possible overloads must be rerun in the new N2 subsystem. If the condition holds, on the other hand, the expansion process is finished. Thus, only the branches enclosed by the boundary in N1 must be tested for overloads.

For each contingency, the method screens the entire network for potential branch flow limit violations and computes the branch flows for a small subset of branches whose limits are endangered. Ideally, only a small portion of the network has to be solved to establish the set of endangered branches.

If there are only a few branches in N2 which violates the condition, the flow of these branches should be calculated and no further expansion of the boundary is needed.

The proof of the angle spread criterion can found in [9].

2.7 Sparse Adjacency Matrix

In order to traverse the nodes from a fault, the sparse adjacency matrix, A, can be utilized. An adjacency matrix is a means of representing which nodes of a graph are adjacent to which other nodes [3].

By adding ones on the diagonal, the subset can be extended one tier from $nodes^n$:

$$nodes^{n+1} = nodes^n(A + I) \tag{2.18}$$

The picture below illustrates the principle of the adjacency matrix:

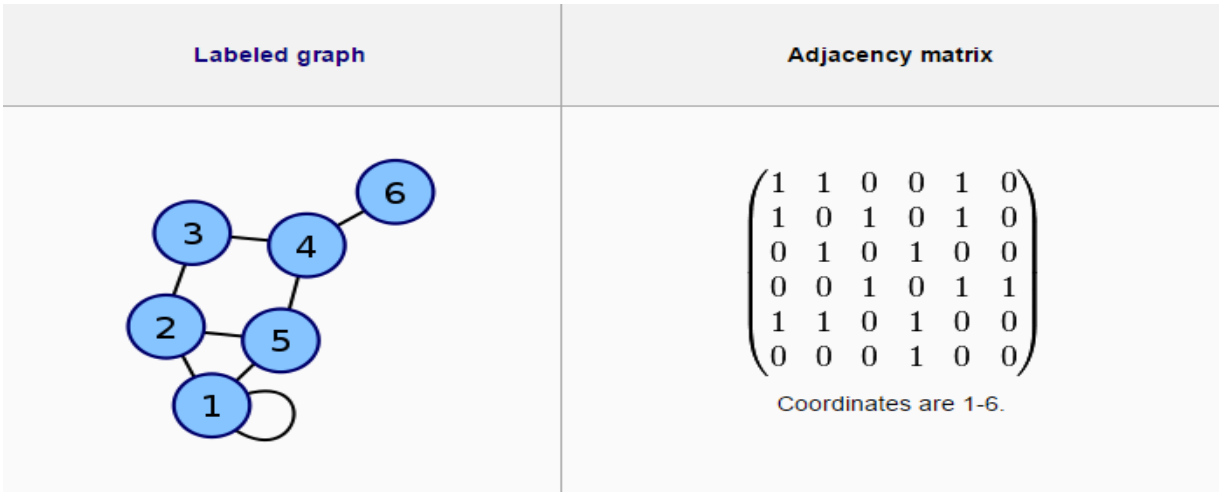

Figure 3: Adjacency Matrix Illustrated [3]

Here node 6 is only connected to node 4, thus the coordinate pair (6,4) & (4,6) models this connection. Node 1 is connected to itself, hence the diagonal element. The matrix is symmetric.

The theory presented above plays a major part in the implementation of the EBM in MATLAB. In the code, the elements of $nodes^n$ are stored to distinguish between boundary - and centre nodes respectively.

2.8 Contingency Selection - Performance Index

The PI is also used in this report to rank contingencies. More information about indirect ranking methods can be found in [16].

The PI is a scalar value which indicates how much a particular outage affect the system. The PI for flows can be given as [18]:

$$PI\_flow_i = \sum_{\substack{all\ branches \\ l,l\neq i}} \left(\frac{P_{flow\ l,i}}{P_l^{max}}\right)^{2n} \tag{2.19}$$

i : 1...N_{lines}

$P_{flow\ l,i}$: The flow on line l with line i out.

P_l^{max}: The maximum flow capacity of line l.

n: Determine the sensitivity of the ranking.

If n is a large number, the PI will be a small number when all flows are within limit, and it will be large if one or more lines are overloaded. When the PIs have been calculated they can be sorted and the contingencies with biggest PIs can be studied further with ACPF. When all the flows are within their limits, the PI will be a small number and when one or more lines are overloaded the PI will rise quadratic if n = 1. One phenomenon with PI, is that several lines that are close to the max rating can give a PI similar to several ones that are just above their limit. This is often referred to as "masking".

In this report an self-made variant of PI is used, which only counts the PI-value of a line if, an only if, an overload is registered. This has been done in order to avoid the problematic considering "masking".

2.9 Adaptive Localization

Adaptive Localization is another method that take advantage of the impact of a fault will cause greater changes in the neighbourhood of the outage. This method is best used starting at a solved base case scenario before the outage, thus avoiding a flat start. When the bus voltage magnitudes and the phase angles are kept constant as in the base case, there will only be significant mismatch error at the two buses at the end of an outaged line. The adaptive localization method only adjusts buses that have significant mismatch. In a typical scenario the algorithm start iterating at the two buses of the outaged line, then more buses are included until the algorithm is about to converge. Then the number of buses that needs adjustment will decrease fast until no mismatches exceeds the predetermined convergence tolerance limit. Further information about this algorithm can be found in [13].

This algorithm has not been implemented in this report, but the concept of not using a flat start, but rather the base case information, haven been used with promising results.

2.10 Linear Sensitivity Factors

Linear sensitivity factors are derived from DCPF. These factors are useful when many contingencies have to be evaluated when there is limited time available. The factors show the approximate change in line flows for changes in generation on the network configuration [18]. There are three categories of them:

- Power Transfer Distribution Factors [PTDFs]
- Line Outage Distribution Factors [LODFs]
- Sensitivity Factors, δ [SFs]

A more detailed derivation can be found in Appendix A.

2.10.1 Power Transfer Distribution Factors

The PTDF factor represent the sensitivity of the flow on line l to a shift of power from i to j.

The PTDFs are derived as:

$$PTDF_{i,j,l} = \frac{\Delta f_l}{\Delta P} \tag{2.20}$$

l : Line index.

i : Bus where power is injected.

j : Bus where power is taken out.

Δf_l : Change in megawatt power flow on line l when a power transfer of ΔP is made between i and j.

ΔP : Power transferred from bus i to bus j.

The outage of a given generator can be simulated with the assumption that all the generation lost will be made up by the swing bus. Suppose the generator before the outage generated P_i^0 MW and all the power is lost. ΔP would then be represented as:

$$\Delta P = -P_i^0 \tag{2.21}$$

The new power flow on each line in the network could be calculated using a pre-calculated set of PTDFs:

$$\hat{f}_l = f_l^0 + PTDF_{i,ref,l}\Delta P \tag{2.22}$$

for l = 1...L where

$\hat{f}_l$: Flow on line l after the generator on bus i fails.

f_l^0: Flow before the failure.

In equation [*] "ref" is substituted for "j" to indicate that the shift is from bus "i" to the reference bus. The "outage flow", $\hat{f}_l$, on each line can be compared to its limit and those exceeding their limit flagged for alarming.

The effects of simultaneous changes on several generating buses can be calculated using superposition. For instance if loss of generator i were to be compensated by governor action on machines throughout the interconnected system. One frequently used method assumes that the remaining generators pick up in proportion to their maximum MW rating. The proportion of generation pickup from unit $j(j \neq i)$ will then be:

$$\gamma_{ij} = \frac{P_j^{max}}{\sum_k P_k^{max}} \quad k \neq i \tag{2.23}$$

P_k^{max}: Maximum MW rating for generator k.

γ_{ij}: Proportionality factor for pickup on generating unit j when unit i fails.

The new flow then becomes:

$$\hat{f}_l = f_l^0 + PTDF_{i,ref,l}\Delta P_i - \sum_{j\neq i}\left[PTDF_{ref,j,l}\gamma_{ji}\Delta P_i\right] \tag{2.24}$$

If any of the generators have reached its maximum production limit, then a more detailed generation pickup algorithm will be required.

2.10.2 Line Outage Distribution Factors

The LODF are used when testing for overloads when transmission circuits are lost:

$$LODF_{l,k} = \frac{\Delta f_l}{f_k^0} \tag{2.25}$$

$LODF_{l,k}$: Line outage distribution factor when monitoring line l after an outage on line k.

Δf : Change in MW flow on line l.

f_k^0 : Original flow on line k before it was outaged.

The LODF can be pre-calculated, and if the power flows on the two lines l and k are known in advance, then the flow on line l, with line k out, can be determined by using LODF:

$$\hat{f}_l = f_l^0 + LODF_{l,k} f_k^0 \tag{2.26}$$

f_l^0, f_k^0 : Pre-outage flows on line l and k.

$\hat{f}_l$: Flow on line l with line k out.

LODF can be used to test for overloads very fast in a contingency algorithm.

2.10.3 Sensitivity Factors

The sensitivity factor δ is the ratio of change in phase angle θ, anywhere in the system, to the original power P_{nm} flowing over a line nm before its outage.

$$\delta_{i,nm} = \frac{\Delta\theta_i}{P_{nm}} \tag{2.27}$$

The factor δ is closely connected to LODF. The derivation of the factor can be found in appendix. The sensitivity factor δ is central in implementing the code of EBM.

2.11 Post-Compensation

Since the post-compensation technique is an important part of this report, a brief derivation will be included here. Alsac [15] gives a more thorough derivation of the compensation technique.

The problem to be solved can be written:

$$(\boldsymbol{Y} + \Delta\boldsymbol{Y}) \cdot \boldsymbol{V} = \mathbf{I} \tag{2.28}$$

A change in the admittance matrix is added and the purpose is to find the new solution of **I**. The admittance matrix **Y** is a sparse (n x n) network matrix. Equation (2.28) can be written as:

$$(\boldsymbol{Y} + \boldsymbol{M} \cdot \boldsymbol{\delta y} \cdot \boldsymbol{M^T}) \cdot \boldsymbol{V} = \boldsymbol{I} \tag{2.29}$$

$\boldsymbol{\delta y}$ is an (m x m) matrix which contains the modifications to **Y**, and **M** is an (n x m) connection matrix.

For the branch oriented compensation an admittance change Δy_l for a branch between nodes i and k is added to Y_{ii} and Y_{kk} and subtracted from $Y_{ik}$and Y_{ki}. When m branches are modified simultaneously, $\boldsymbol{\delta y}$ becomes an (m x m) diagonal matrix of the admittance changes. $\boldsymbol{M}$ has m columns each with +1 and -1 in the relevant positions.

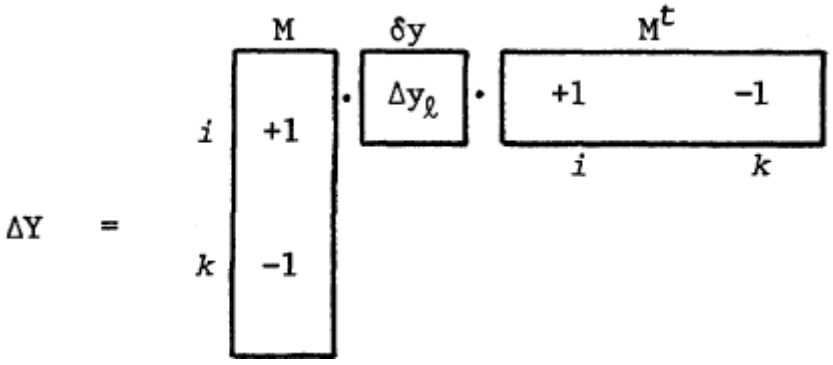

Figure 4: Branch Oriented Compensation [15].

In order to solve the (2.29) the IMML[2] has to be used.

$$\boldsymbol{V} = (\boldsymbol{Y^{-1}} - \boldsymbol{Y^{-1}} \cdot \boldsymbol{M} \cdot \boldsymbol{c} \cdot \boldsymbol{M^T Y^{-1}}) \cdot \boldsymbol{I} \tag{2.30}$$

[2] Inverse Matrix Modification Lemma

The (*m* x *m*) matrix **c** need to be defined and then it is beneficial to introduce another (*m* x *m*) matrix, **z**. This matrix is composed of the elements of $\boldsymbol{Y^{-1}}$ associated with the nodes affected by the modification. Each element of z contains four elements of $\boldsymbol{Y^{-1}}$in the branch-oriented approach.

$$\boldsymbol{z = M^T \cdot Y^{-1} \cdot M} \tag{2.31}$$

The matrix **c** can then be written:

$$\boldsymbol{c = (\delta y^{-1} + z)^{-1}} \tag{2.32}$$

When using post compensation the process can be summarized into the following steps:

1. Perform network solution with fast forward and backward substitution.

$$\boldsymbol{\widehat{V} = Y^{-1} \cdot I} \tag{2.33}$$

2. Calculate compensation vector.

$$\boldsymbol{X = Y^{-1} \cdot M} \tag{2.34}$$

$$\boldsymbol{\Delta V = X \cdot c \cdot M^T \cdot \widehat{V}} \tag{2.35}$$

X in (2.34) should be solved by LU factorization, without calculate the $\boldsymbol{Y^{-1}}$explicitly.

3. Perform Compensation.

$$\boldsymbol{V = \widehat{V} - \Delta V} \tag{2.36}$$

In this project post-compensation is also used for DCPF. Then the compensation procedure corresponds as described above, but instead of (2.28) the compensation process starts with:

$$\boldsymbol{(B' + \Delta B') \cdot \delta} = \mathbf{P} \tag{2.37}$$

3 Results From Project Report

The project report [24] contained an important mistake in the compensation code. The element **c** is calculated with the wrong operator. The operator " \ " was used instead of the correct " / " divide operator. This became a source of frustration, because when the code did not produce correct output, much time was spent at error checking. The author of the project report was blind to this fault. Thus an important lessoned learnt is to not only focus at the variables, but also remember to check the operators carefully. The two scripts `DCcompAlg.m` and `contTest1.m` from the project report have been corrected. The updated code can be found in Appendix B.

The comment about the splitting of M-matrix, in the project report, was wrong. This splitting process is correct, it was realised when inquiring into Stott [6] more thoroughly. Therefore the FDPF algorithm, which utilizes post-compensation, has been central in this report.

4 Explanation of the MATLAB-code

MATLAB (Matrix Laboratory) has been used in this report too [19]. The software is designed for matrix operations [20]. In power system analysis matrix operations is central. The MATLAB power system simulation package, MATPOWER is also central in this report. This package has been updated to version 5 recently [21]. Most of the algorithms in this report depend on various MATPOWER functions. Instead of spending time at building power system functions from scratch, the focus can be at implementing and testing efficient power flow algorithms like the EBM.

The implementation has mostly been done in scripts. Most of the functions are from MATPOWER. When it comes to use of memory of the computer it is more efficient to use several functions and return local variables. However, in this report scripts has been used to increase the readability of the code in the appendix and to avoid a lot of fragmented functions. It is easier to follow the structure of the MATLAB-code in the appendix in a consecutive order without many different functions to keep track of. Some output matrices have been used in order to check the results. This is neither optimal with respect to memory usage, but makes

it easy to check the results afterwards. Another benefit by writing the code in scripts, it that It is easy for other who read the report to test the code fast; simply download and activate the MATPOWER to the MATLAB folder. Then create identical scripts and copy the code from the appendix. The following sections will provide some explanations of the MATLAB-code and a brief discussion about the code.

4.1 The Efficient Bounding Method

The code `bounding.m` can be found in Appendix B.

Pedersen [23] tried to implement the bounding method. This has also been an objective for this report, because the EBM is central when it comes to model reduction of a power system during an outage, and it is based on an angle spread criterion which is mathematically proved. The characteristic of using an adjustable region around the outage is also fascinating. The code of this report has taking the algorithm a step further, by adding some new features and by correcting an important logical mistake in the previous code. The most profound feature is that the algorithm does not just indentify the danger area, it also check for flow limits and sort them based on PIs. It is also possible to rise the angle spread limit and thus avoid expansion layers.

One important mistake found in the previous code cause the angles to be updated wrong due to the following implementation of the two inverse columns of the **B**'-matrix:

```
C_bf = Q * (U \(L \(P * N_bf([pv;pq]))));

C_bt = Q * (U \(L \(P * N_bt([pv;pq]))));
```

Here the **`C_bf`** and **`C_bt`** will be indexed with the PV-nodes first and then PQ-nodes, but this indexing is not used when the PTDF are calculated. As an example with the 24-bus system, the difference with this two types of indexing will be:

- Normal consecutive indexing:

  1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24

- Indexing with combined list of PV-nodes first and PQ-nodes afterwards (**`[pv;pq]`**):

1-2-7-14-15-16-18-21-22-23-3-4-5-6-8-9-10-11-12-17-19-20-24

Node 13 is the slack bus and is therefore not present in the second list above. The reason why this will produce wrong output, can be given by the following example:

Suppose that the boundary nodes are **11-12-13-14-15**, then the sensitivity factors of the node **3-4-5-6-8** will be calculated instead. Hence, the algorithm will constantly be calculating the wrong boundary angles. This will make the testing of the angle spread criterion worthless for the entire algorithm. To avoid this mistake, the implementation must be changed accordingly:

```
C_bf([pv;pq])) = Q * (U \(L \(P * N_bf([pv;pq]))));

C_bt([pv;pq])) = Q * (U \(L \(P * N_bt([pv;pq]))));
```

In the code bounding.m, all the inverse columns of **B**' are calculated before the contingency analysis starts. Then only half the operations are needed, because two inverse columns must be calculated for each contingency otherwise. Hence, pre-calculating will save **n** unnecessary inverse operations during the analysis, but in return it will need more storage space in the computer memory. However, the storage in memory will be just 43.3 MB and it only takes 2.4 seconds to calculate. The time required to calculate the inverse columns when they are needed, are much greater than pre-calculating for the big Polish power system.

There are three adjustable parameters to control the EBM. The variable **start_steps** control who many start steps the algorithm shall use before checking for the angle-spread criterion. This is found to not have a great impact at the time consumption. It can be set at 0. **max_steps** decide the maximum number of steps the EBM are allowed to do for each contingency. When the simpler variant CRM are to be tested, **start_steps** can be set to the same number as **max_steps** for the preferred number of layers in the analysis. **raiseLimit** control how much the minimum limit are to be raised. For some power systems this option should be used, because some branches can be close to their capacity limits. If the maximum angle spread value of the boundary nodes are to be less than the smallest value of branch-limits, this will in some cases include the entire system. Even some values can be negative due to negative reactance values. The idea behind raising this value, is that fewer steps are needed. When the limit is raised, the branches which is not part of the criterion check will be calculated explicitly. During the loop process, only the flow of the branches which are enclosed by the boundary, plus the branches which are left out because of the raise of the limit, will be calculated.

When new branch-flows are calculated, they will be tested against their limits. Only those branches that exceed their limits will contribute to the PI. The list of the PI will be sorted, so that the most important contingencies can be identified.

It is possible to check a detailed output from the code. The output **`limit`** is the limit, which the maximum angle spread of the boundary must be less, in order for the subsystem outside the boundary nodes to be safe for overloads. **`numbers_of_nodes_visisted`** displays the total number of nodes which enclosed by the boundary nodes, with the boundary nodes included. **`not_success`** will indicate if any of the contingencies need the entire system to be included in the boundary. **`resultPI`** keeps track of the PI and what contingency which cause it, and are sorted with the highest PI first. The output-variable **`angle_spread_log`** keeps track of how the maximum angle spread of the boundary changes for each step expansion.

4.2 Single AC Contingency Compensation Algorithm

The code `compensationAC.m` can be found in Appendix B. This algorithm builds on the work of Pedersen [23], but some new features have been implemented.

There are two control parameters. The first one **`tol`** is the tolerance limit for the FDPF. This is set to 1% as default because it seems as a reasonable trade-off between consumption of time and accuracy. This convergence limit represent the upper limit of the mismatches for any of the P - and Q bus values. Hence many of the buses will have values in the range between 0.001-0.01. The second control parameter is **`max_it`**. This parameter will control the maximum number of iterations allowed in the FDPF. Most contingencies tend to converge during 3-4 iterations. The maximum number is set to 10 by default. If the system has not converged within ten iterations, it will probably need many more iterations, if it converge at all.

One important aspect with the algorithm, is that a reference power flow is run before the contingency analysis starts. The updated phase-angles and voltage magnitudes are used throughout the contingency process. This will decrease the time substantially for big systems. For the biggest 3120-bus system, a flat start will at least need twenty iterations for most of the contingencies. Flat start is the state where voltage-magnitudes are set equal to 1 PU and phase-angels to 0°. Most of the contingencies will converge within four iterations when

running a reference flow. This will save thousands of unnecessary iterations and save a substantial amount of time for the biggest systems. Hence, the theory from the adaptive localization algorithm has been useful despite the algorithm itself has not been implemented. Since most of the phase angels and voltage magnitudes far away will change little from the base case, only the buses with the largest mismatches will be corrected. The effect of faster convergence is like an indirect model reduction of the system, which make it possible for a larger number of cases to be evaluated per time unit.

LODF are used to estimate the post-contingency flows fast. For the biggest 3120-bus system, the time to pre-calculate the LODF is 4.4 seconds and the storage in memory 104 MB. For a modern, powerful computer this will not be an issue and the time consumed will be less.

The algorithm will for each contingency indentify the biggest overload in MW-terms and the biggest or lowest voltage magnitude that are outside the preferred interval; for instance [0.90-1.10] or [0.95-1.05]. The output matrices **`PflowViolation`**, **`voltageViolationLow`** and **`voltageViolationHigh`** have all three columns. The first column shows the contingency which cause the violation, the second column shows the violation's location and the third column the value itself. The output vector **`iterations`** keeps track of the number of iterations per contingency.

4.3 Double DC Contingency Analysis

The double DC contingency analysis algorithm, doubleCompensationDC.m, can be found in Appendix B.

Before the DC double contingency start screening for the most severe outages, a reference flow is run by NRPF. This will provide a more accurate starting point for the DC-analysis. The code compensate correctly for parallel lines and if outage of a line will cause islanding, the code will skip the analysis. The following part of the second for loop is important:

```
if b2 <= b1
continue % Make sure unique combinations.
end
```

The code above will make sure that only unique combinations are tested, instead of testing for n^2 combinations which is more time consuming and unnecessary. The formula for theoretical unique combination is [23]:

$$n_{unique} = \frac{n(n+1)}{2} \tag{4.1}$$

n : Number of branches.

The true number of combinations will be less due to skipping of islanded nodes and the fact that combinations where **`b1 = b2`** are of no physical interest. This can be illustrated by an example from the 24-bus system. Here, outage of the branch between node 7 and node 8 will cause islanding of the generator at node 7. Thus, the total number of combinations tested are:

$$n_{combinations} = n_{unique} - n_{b1=b2} - n_{island} \tag{4.2}$$

$$n_{combinations} = \frac{38 \cdot 39}{2} - 38 - (38 - 1) = 666 \tag{4.3}$$

Testing of the code gives the exactly same number, 666, from the output variable **`count`**.

The variable **`resVa1`** is a matrix which will store all angles for single branch outages, while **`resVa2`** is only a vector. The PI is used to identify the most important contingency combinations.

The LUPQ-factors for the first branch outage, **`b1`**, must be stored because they are needed throughout the contingency process:

```
[L0, U0, P0, Q0] = lu(Bt);
```

Before the second for loop, new LUPQ must be pre-calculated. This will make the code a bit less effective and optimal. It should be possible to update the first LUPQ factors and compensate the line outage Instead of calculating them n times before the second for loop executes. This should be investigated in future work.

4.4 Generator Outage Contingency Analysis

A simple generator outage algorithm has been made to demonstrate another benefit use of PTDF. The algorithm utilizes PTDF to determine the post-outage flow under the assumption that the slack bus compensate the lost generation. This will increase the pressure on the

branches closest to the slack bus. It is possible to make a more complicate algorithm which can share the burden between more generators according to the theory, but this algorithm must also take into account that some of the generators may produce at maximum limit. This has not been implemented in this algorithm.

Only the inverse columns of **B** that are needed, are calculated. This is executed before the contingency analysis begins. It is found by testing, that for large systems, it takes less time to pre-calculate these factors and store them in memory, rather than calculate them for each contingency. This has been verified by testing of time consumption in MATLAB.

The algorithm iterates through all possible generator outages except for the generators present at the slack bus. The post-contingency flows are calculated, and if any overloads are discovered, it will be ranked with PIs. As with the other algorithms in this report which utilize PI-ranking, only overloaded branches will contribute to the PIs. This has been done in order to prevent the masking error of PIs.

5 The Power Systems

5.1 24-Bus System

Several power systems has been used to test the different algorithms. On small system, on medium and a large one. The systems have no need for conversion between intern and external indexing. The data is ordered in a consecutive way.

The small system used is **`case24_ieee_rts.m`**. This is a IEEE 24-bus reliability test-system. The power system has some parallel branches and it is N-1 secure in the base case scenario. It contains thermal data which are used when testing for overloads. The 24-bus system has been important when developing the algorithms to make sure they produce the expected output. This easier to test for a relative small system with good overview and small output. There is also a map of the power system, which displays how the nodes and branches are connected to each other. The voltage levels is 138 and 230 kV.

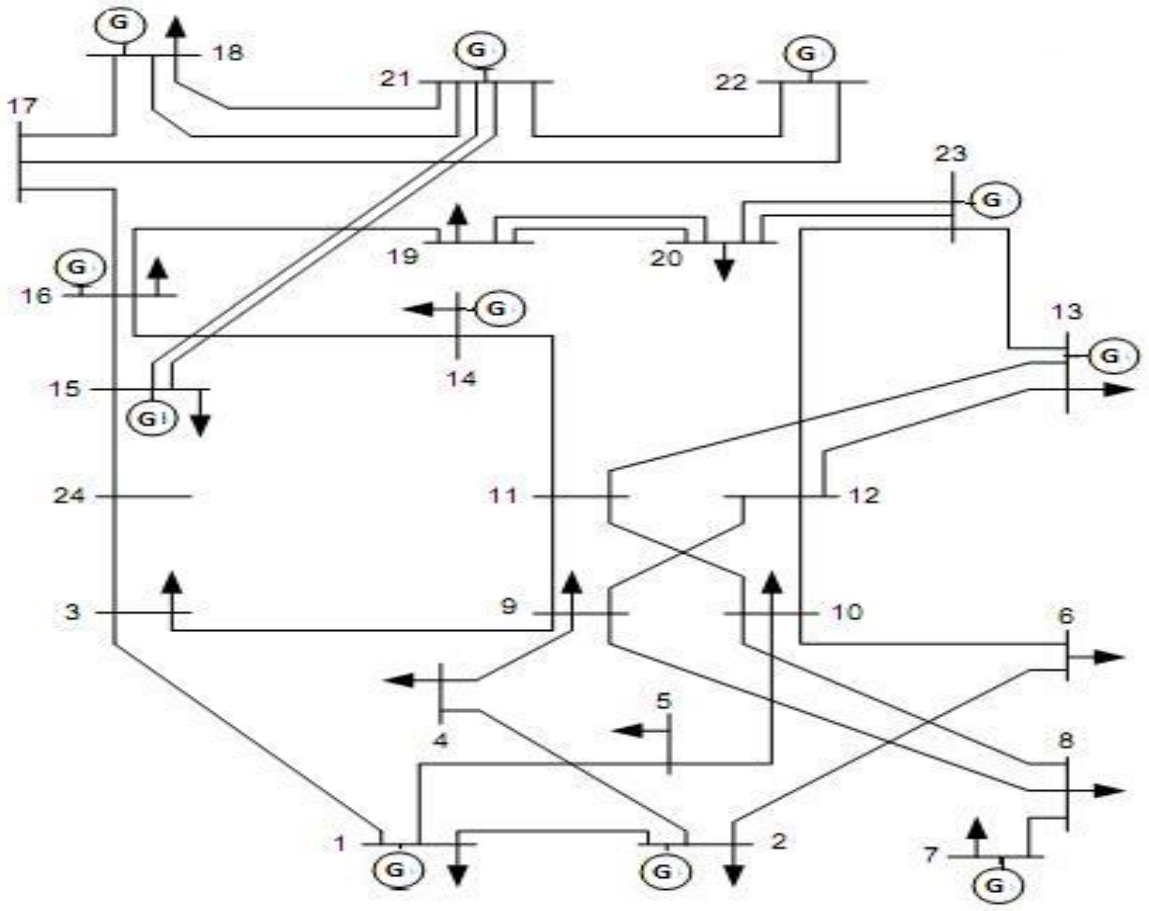

Figure 5: 24-Bus System [26]

5.2 118-Bus System

The medium sized **`case118.m`** system is also used. This is a 118-bus system which also has a map available. There are two different voltage levels; 138 and 345 kV. One drawback with this system, is the lack of thermal data. The algorithms make some simple approximations about these levels. All the branches which have an absolute value of power flow below 100 MW in the base case scenario, are allocated a thermal capacity of 250 MW. The remaining branches are allocated a thermal rating equal to 130 % of the largest absolute value of all branches. A picture of the power system is presented below.

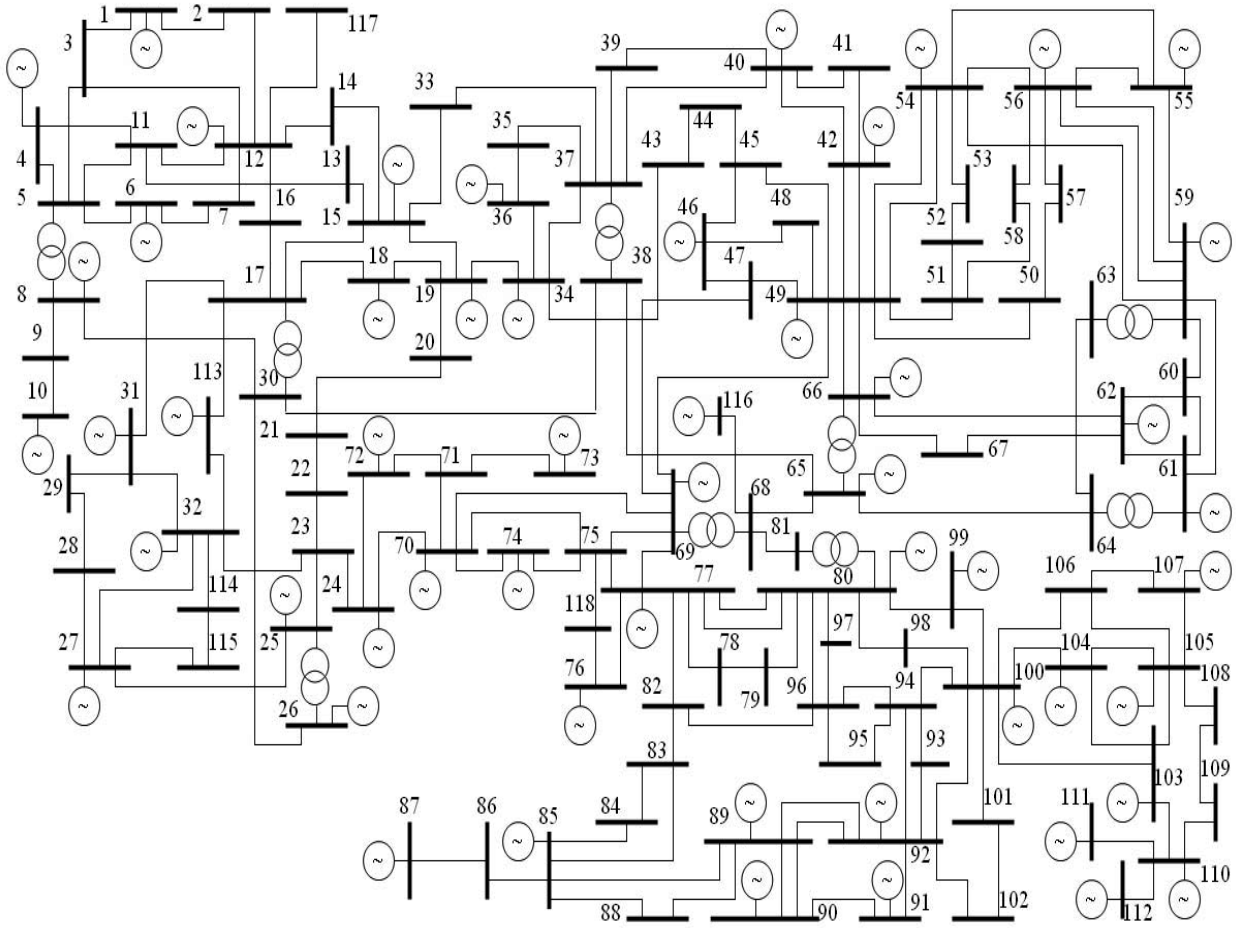

Figure 6: 118-Bus System [25]

5.3 2383-Bus System

The largest systems used, is the **`case2383wp.m`**. This is a 2383-bus system which models the winter 1999-2000 winter peak. It is a part of the 7500+ bus European UCTE system. In order to decrease the number of buses, the tie lines to foreign networks are replaced by artificial load and generator buses. The voltage levels are 400, 220 and 110 kV. Multiple generators at a bus have been aggregated in this system. This large system is useful when it comes to testing consumption of time of the algorithms. Thermal data are present for this system.. This power system has no map however. It is impossible to present an overview of the large system on a small A4-page.

Originally, a bigger polish system was used, **`case3120sp.m`**, which models the Polish system during the summer 2008 peak. Unfortunately, the lack of proper data was found to be a problem for this system. This was first pointed out by Pedersen [23]. Several hundred of the

buses have the resistance value of 8E-05 and the reactance value of 8.3E-4. These are very small values for both resistance and reactance, and the low value of the reactance cause converge problems with the EBM. It was also discovered that ten of the resistance values were negative! This make no sense in a physical perspective, as it means the voltage will increase over these resistances. Because of this lack of reality, and disappointing data quality, the smaller 2383-bus-system was chosen instead.

6 Results & Discussion

6.1 FDPF - Primal Versus Dual

There was some uncertainty regarding different X/R ratios and FDPF in the project report by this author [24]. The scaling of the resistance and reactance has been updated. The primal version is the BX version of the FDPF, which starts with P-θ iteration, whereas the dual version is the XB version, and starts with Q-V iteration. The abbreviation, BX, means that all the resistance elements in the second B-matrix is set to zero. The updated primal version, `fdlfPrimal.m`, can be found in Appendix B. The comparisons of the two versions can be found in the table below:

Table 1: FDPF - Primal versus Dual: X/R-ratio

Scaling		**Iterations**	
R	**X**	**Primal**	**Dual**
4.00	1.00	12	17
3.00	1.00	9	10
2.00	1.00	6	8
1.00	1.00	5	6
1.00	0.50	5	7
1.00	0.25	7	14
1.00	0.20	7	19
1.00	0.15	8	-

The result above has the same tendency as Amerongen [7], and Monticelli [8]. The primal version needs less iteration when the system's X/R-ratio is low. With neutral scaling, the two methods perform similar with 5 and 6 iterations respectively. For the least scaling, X = 0.15,

the dual version did not converge. This indicates that the primal version has best converge characteristic for distribution systems with low X/R-ratio.

6.2 Double DC Contingency Analysis

In Pedersen [23], under future work section, it is suggested to implement double DC contingency analysis utilizing the post-compensation technique. This has been done, and it is built as a screening algorithm, which identifies branch overloads based upon PIs. The PIs are calculated only for the branches that are identified as violation of flow limit. The advantage with this approach is that the "masking error" is avoided. However, the algorithm gives no information regarding voltage levels. The algorithm's purpose is to identify the most critical double contingencies when time is limited. This algorithm does not take into account that the reactance level might change when the flow of the line is changing [4].

6.2.1 Small System

The table below displays how accurate the algorithm is, when compared to NRPF. The initial phase angles which are used in the double DC compensation code are calculated from a NRPF. This is found to give better accuracy, rather than start with the angles from a DCPF. Branch number 7 (3 to 24) and branch number 24 (15 to 16) are out of service.

Table 2: Double DC: Accuracy - 24-Bus System

Accuracy - 24-Bus System		
Bus	Angle [°]	
	AC	DC
1	-14.1	-13.2
2	-13.9	-13.1
3	-21.0	-20.1
4	-15.4	-14.6
5	-14.9	-14.1
6	-16.2	-15.6
7	-11.4	-10.7
8	-15.1	-14.4
9	-12.2	-11.7
10	-12.4	-11.8

As can be seen from the table above, the angles from the DC double compensation code gives a good estimate for the angles compared to NRPF.

Table 3: Double DC: PI - 24-Bus System

Performance Indices - 24 Bus System		
Branch Interruption		PI
23	29	6.95
24	28	5.47
19	29	2.59
7	23	1.52
25	28	1.50
26	28	1.50
23	27	1.48
7	29	1.40
27	29	1.37
21	22	1.31

The table above shows the top ten combinations with highest PI. Some of the combinations have the same PI due to parallel lines. The two combinations with highest PIs; branch 23 (14 to 16) & 29 (16 to 19) and 24 (15 to 16) & 28 (16 to 17) are checked closer with a proper NRPF. These two branch combinations do not converge. This means that the algorithm has correctly identified two problematic combinations. It is possible to obtain a proper NRPF for the third highest PI of 2.59, with outage of 19 (11 to 14) & 29 (16 to 19). It appears that there are three branches which are close to their thermal short-term ratings. Branch 6 (3 to 9) has a thermal rating of 208 MW and is loaded 193 MW, branch 7 (3 to 24) has a thermal rating of 510 MW and is loaded 483 MW whereas branch 27 (15 to 24) is loaded with 509 MW and has a thermal capacity of 600 MW. Taking into account that the screening algorithm utilizes DC, and several of the branches of the 24-bus system have an X/R-ratio of just 4, the results seems promising.

6.2.2 Large System

The 2383-bus system has 500 branches which will cause isolation of a node. These combinations are skipped because the screening algorithm will not manage these cases. About ten of the branches are already overloaded in the base case scenario, with respect to short-

term thermal ratings. Thus, the screening algorithm will identify almost all combinations as potential overloads.

Time becomes a limiting factor when testing the algorithm at the large system. To illustrate this, the screening algorithm is run for just ten branches with all possible combinations of these. This will result in 23 905 unique combinations! The time needed to calculate and check for overloads, is approximately 75 s or 3.1 ms per combination. The 100 highest PI, range from 59 to infinity. Hence, a strategy based upon choosing the branches with the highest initial flow, and then test for all possible combinations, seems manageable. The contingencies with highest PIs can then be studied more thoroughly with NRPF or FDPF. This is also the purpose of a screening algorithm; to fast identify the most important contingencies. If all possible combinations are to be tested for a large system, it will take hours to complete the analysis!

One drawback with the 2383-bus system is the X/R-ratio. It is relative small for many of the branches. Several hundreds of the branches have a ratio of less than 2.8. This is not optimal for an algorithm which utilizes DC; hence the interpretation of the output must be critical.

6.3 Single AC Contingency Compensation Algorithm

6.3.1 Small System

The SACCCA is tested. It takes only 0.13 seconds to test for 37 branch outages. In order to test the accuracy of the code, the outage of branch 27 (15 to 24) is used. The accuracy can be studied in the table below.

Table 4: Single AC Contingency Compensation Algorithm - Accuracy

Single AC Contingency Compensation Algorithm - Accuracy						
Bus/Branch	Voltage-Angle [°]		Voltage [PU]		Active Power [MW]	
	NRPF	Algorithm	NRPF	Algorithm	NRPF	Algorithm
1	-14.04	-13.93	1.035	1.035	22.7	21.2
2	-13.87	-13.75	1.035	1.035	65.1	61.1
3	-20.97	-20.88	0.925	0.930	21.56	24.1
4	15.29	15.19	0.987	0.988	28.4	28.1
5	14.80	14.71	1.014	1.015	23.9	25.7
6	-16.13	-16.05	1.006	1.007	117.9	123.6
7	-11.34	-11.24	1.025	1.025	0	4.3
8	-15.01	-14.91	0.987	0.987	46.2	46.5
9	-12.16	-12.10	0.981	0.983	49.6	47.7
10	-12.28	-12.22	1.020	1.021	112.4	111.4

The accuracy seems good for the voltage angles and voltage magnitudes. When it comes to active power flow, which is calculated with LODFs, there is some more difference, but most of the values are within 10%. Branch 7 (3 to 24) is a bit special, because it is part of a radial structure (3-24-15), and the flow should be zero because bus 24 has no generation or load. However, the flow of 4.3 MW is rather small and can be neglected.

If the algorithm is set to detect voltage magnitudes outside the interval of [0.95-1.05], the algorithm will identify four cases. The result and comparison with NRPF for these cases can be found in the table below:

Table 5: Single AC Contingency Compensation Algorithm - Critical Values

Single AC Contingency Compensation Algorithm - Critical Values					
Branch Outage	Bus		Node	Voltage [PU]	
	From	To		Algorithm	NRPF
10	6	10	6	0.907	0.673
27	15	24	24	0.907	0.898
18	11	13	21	1.050	1.050
28	16	17	17	1.051	1.051

The algorithm has correctly identified important contingencies when testing against NRPF. However, for the outage of branch number 10, the actual voltage magnitude is much lower. This is probably due to hitting the reactive power limits, and the algorithm does not model

this. Despite this, the algorithm has correctly identified the contingency of branch 10 as problematic.

The table below displays the number of iterations needed per contingency:

Table 6: Single AC Compensation Algorithm - Iterations

Single AC Compensation Algorithm - Iterations			
Branch	**Bus**		**Iterations**
	From	**To**	
1	1	2	1
2	1	3	2
3	1	5	3
4	2	4	3
5	2	6	3
6	3	9	3
7	3	24	4
8	4	9	3
9	5	10	2
10	6	10	4
11	7	8	NaN
12	8	9	3

For most of contingencies only a few iterations are needed. The convergence tolerance is set to 1% and a reference flow is run before the analysis starts. The updated voltage magnitudes and voltage angles from the reference flow are used in the contingency analysis. For outage of branch 11, "not a number" is displayed. This is due to the fact that the outage of branch 10 will cause islanding of the node 7.

6.3.2 Large System

The SACCCA has also been tested at the large system. It has 2896 branches and 500 of them will be skipped by the algorithm due to the islanding issue. The time needed to create LODF-matrix, is just 2.36 s and it needs only 64 MB of computer memory. The SACCCA solves the system in just 25 s, or approximately 10.4 ms per contingency! It appears that this system is overloaded already in the base scenario; hence the list of PIs is large. The NRPF needs 476 s to complete the analysis, whereas the FDPF needs 207 s. This clearly indicates that the SACCCA can scan much faster through the system, compared to NRPF and FDPF. The

analysis is done for 2396 contingencies, thus the 500 branches which cause islanding is accounted for. The convergence tolerance was set to 10^{-5} for both NRPF and FDPF to assure good accuracy.

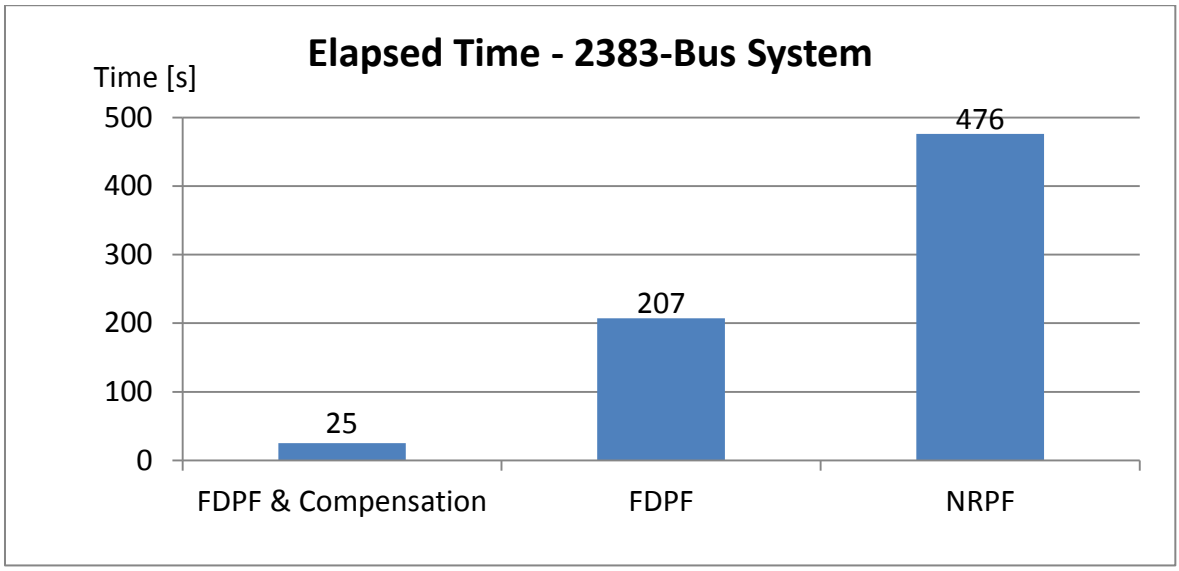

Figure 7: Elapsed Time - NRPF, FDPF and FDPF + Compensation

The graph above shows the huge difference in consumption of time. It must be emphasised that the convergence tolerance is much stricter for FDPF and NRPF. The two algorithms are also more advanced; they adjust for reactive power limits, and are able to produce a lot more detailed output. However, when time is a limited factor, the SACCCA shows promising results. There is a reduction of time of 95% compared to NRPF, and compared to FDPF, the reduction is 88%!

When it comes to accuracy, bus 169 is overloaded with 91 MW already in the base case scenario. The algorithm identifies bus 169 to be the most overloaded branch in MW-terms for most of the contingencies.

When it comes to overvoltage, just two contingencies have a voltage magnitude above 1.10 PU. For voltage level below 0.90 PU, most of the contingencies report bus 1905 as a problem, with a value of 0.89 PU. This occurs because the base case scenario has this under-voltage, hence the system has both problem with overloading and under-voltage even before the contingency analysis starts. This makes the analysis a bit less interesting for this system.

The graph below illustrates the benefit of the post-compensation technique, when testing the largest 3120-bus system. For standard FDPF, the two columns to the left represent tasks which are needed per contingency; B-matrices must be built and LUPQ-factors must be calculated.

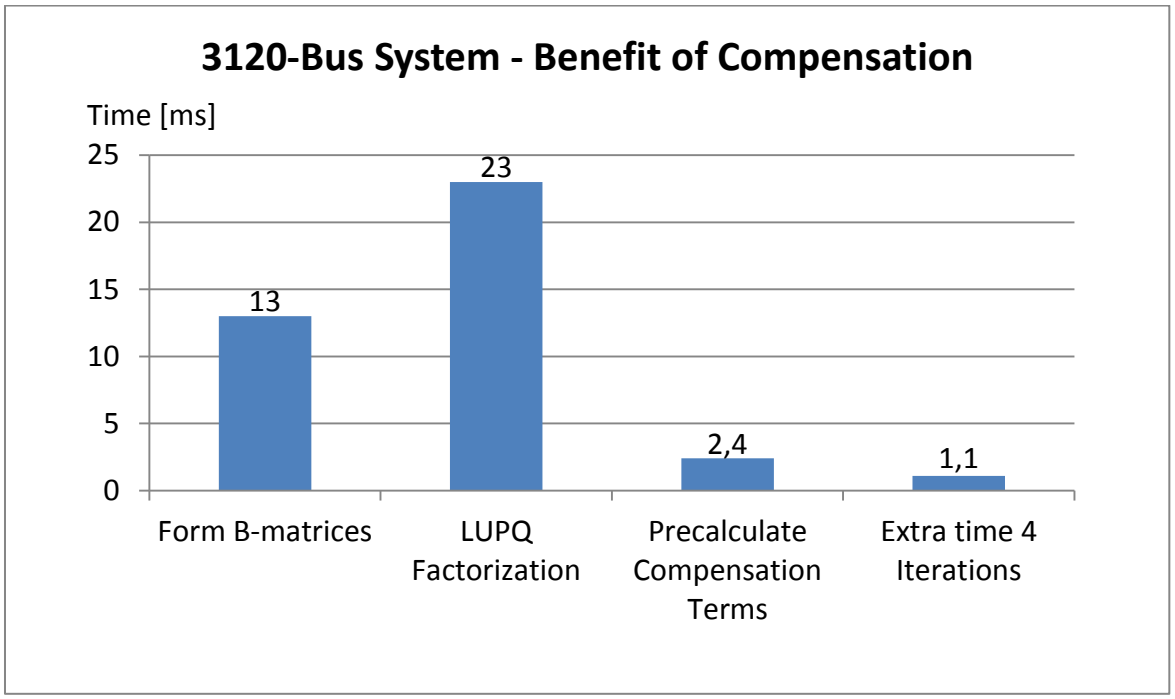

Figure 8: 3120-Bus System - Benefit of Compensation

The two small columns to the right, in the graph above, represent the extra time needed for the compensation procedure; elements must be pre-calculated and compensation factors must be calculated during the iteration process. The extra time needed to calculate compensation terms per contingency, is little compared to the time needed to form B-matrices and LUPQ-factorization.

6.4 Efficient Bounding Method

6.4.1 Small System

The EBM has been tested for the small power system. Outage of branch 3 (1 to 5) has been chosen to test the accuracy of the algorithm. The result for some nodes can be found in the table below.

Table 7: EBM: 24-Bus System - Accuracy

EBM: 24-Bus System - Accuracy		
Bus	**Angle [°]**	
	Normal DC	**Bounding**
1	-3.019	-3.020
2	-3.472	-3.472
3	-3.891	-3.891
4	-7.530	-7.530
5	-13.330	-13.330
6	-11.821	-11.821
7	-6.699	-6.556
8	-10.745	-10.745
9	-6.454	-6.455
10	-9.738	-9.738

The results in the table above seems accurate, thus the implementation of the linear sensitivity factor δ must be correct. It must also be stated that only the voltage angles of the nodes which are considered affected, are updated by the EBM. Node 7, for instance, has not been updated.

The table below contains the list of nodes that are traversed, when outage of branch 3 (1 to 5) is still the case. 12 of the 24 buses in the system is checked, thus 50%.

Table 8: EBM: 24-Bus System - Nodes Visited

EBM: 24-Bus System - Nodes Visited											
1	2	3	4	5	6	8	9	10	11	12	24

The graph below shows the converge characteristic of this outage. For the criterion of the EBM to be fulfilled, the angle-spread of the boundary nodes must be less than the smallest value of $x \cdot f_{max}$. The algorithm needs only 2 steps to converge successfully.

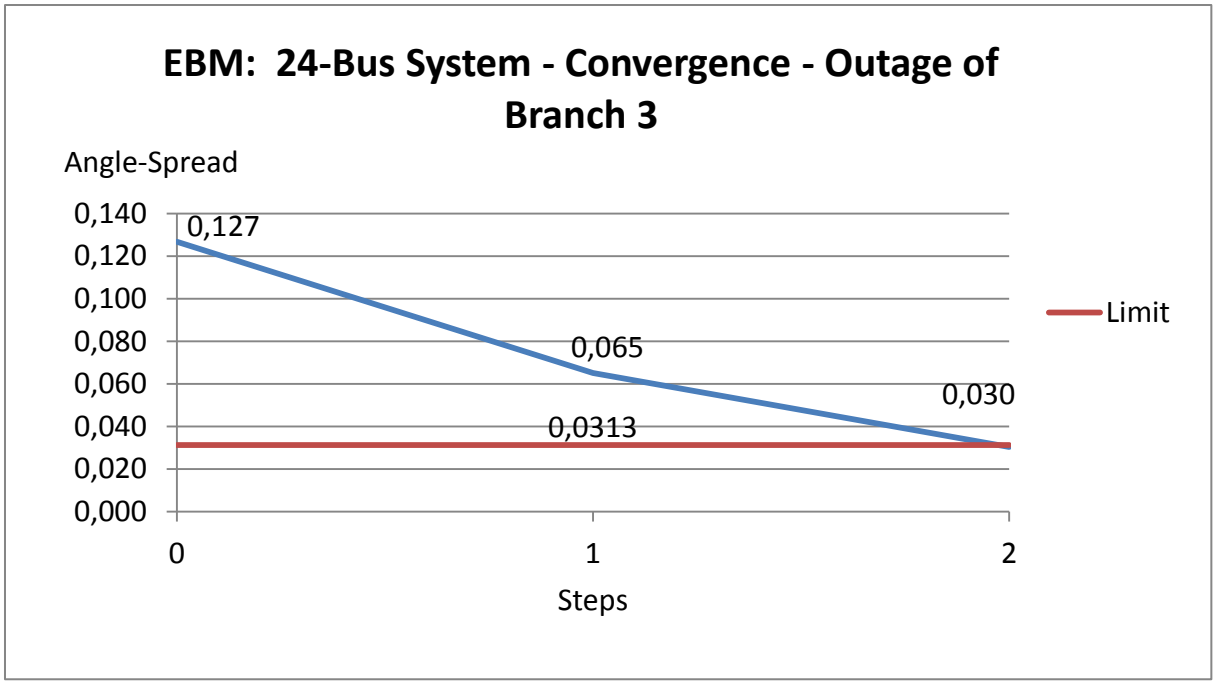

Figure 9: EBM: 24-Bus System - Convergence - Outage of Branch 3

The graph below illustrates how many of the nodes which must be traversed. This is a very small system, thus the EBM is not intended for such power systems. However, it is of interest to see how the algorithm performs. Some of the contingencies do not need expansion at all, from the initial boundary enclosing the branch outage. These contingencies only have two nodes visited: 1, 2, 9 and 33-37. Contingency 11 will cause islanding, hence the value 0.

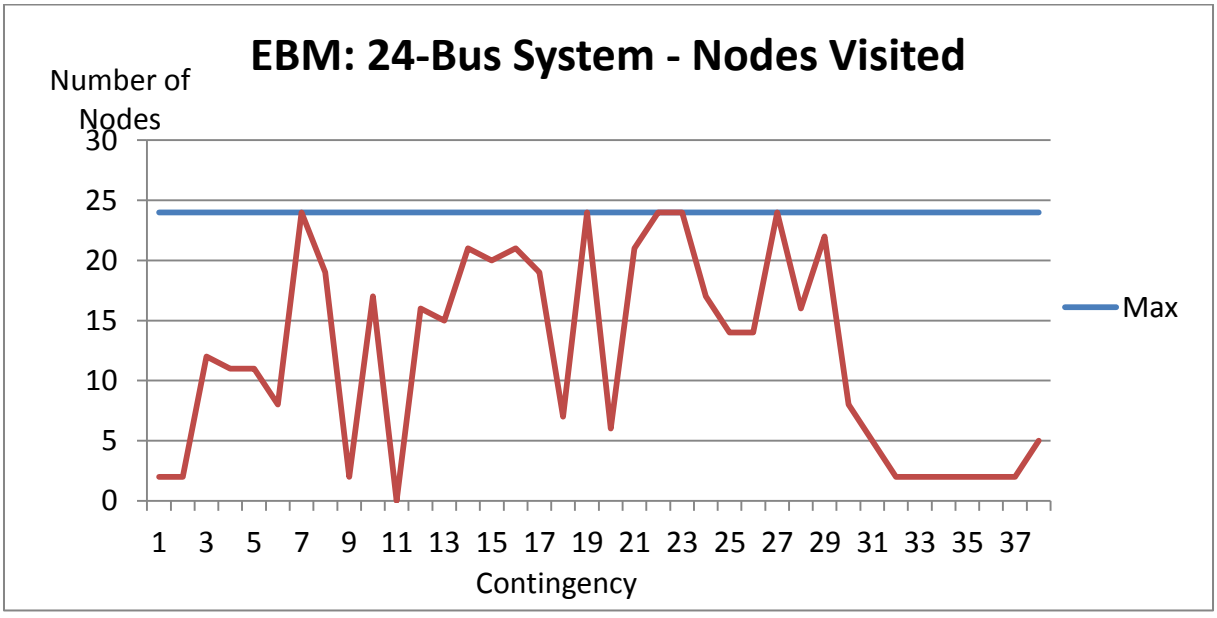

Figure 10: EBM: 24-Bus System - Nodes Visited

6.4.2 Medium System

The EBM are implemented with a setting which makes it possible to raise the angle-spread limit. If some of the branches are close to their respective limits, many steps will be needed to get a very small angle spread limit of the boundary nodes. It will be more beneficial to tag these branches as "danger branches". Then these branches' flows can be calculated, in addition to the nodes enclosed by the boundary. This way, the angle-spread limit can be raised. From a theoretical point of view, this should lower the number of steps needed, and thus increase the speed of the algorithm. This is actually the case, as can be seen from the table below.

Table 9: EBM: 118-Bus System - Effect of Raising the Angle Spread Limit

EBM: 118-Bus System - Effect of Raising the Angle Spread Limit			
Branch Outage	**Limit [rad]**	**Limit + 10 [rad]**	**Limit + 20 [rad]**
	0.022	0.047	0.081
	Number of Visited Nodes		
1	2	2	2
2	13	5	5
3	29	6	6
4	21	16	8
5	29	12	2
6	4	2	2
7	12	12	12
8	91	73	40
9	NaN	NaN	NaN
10	5	2	2
Time [ms]	505	394	331

When the limit is raised, the number of visited nodes decrease and the time consumption also decrease. By raising the limit by 20 branches, the need time will decreases by almost 35 %. This indicates that raising the limit can be useful for certain power systems.

6.4.3 Large System

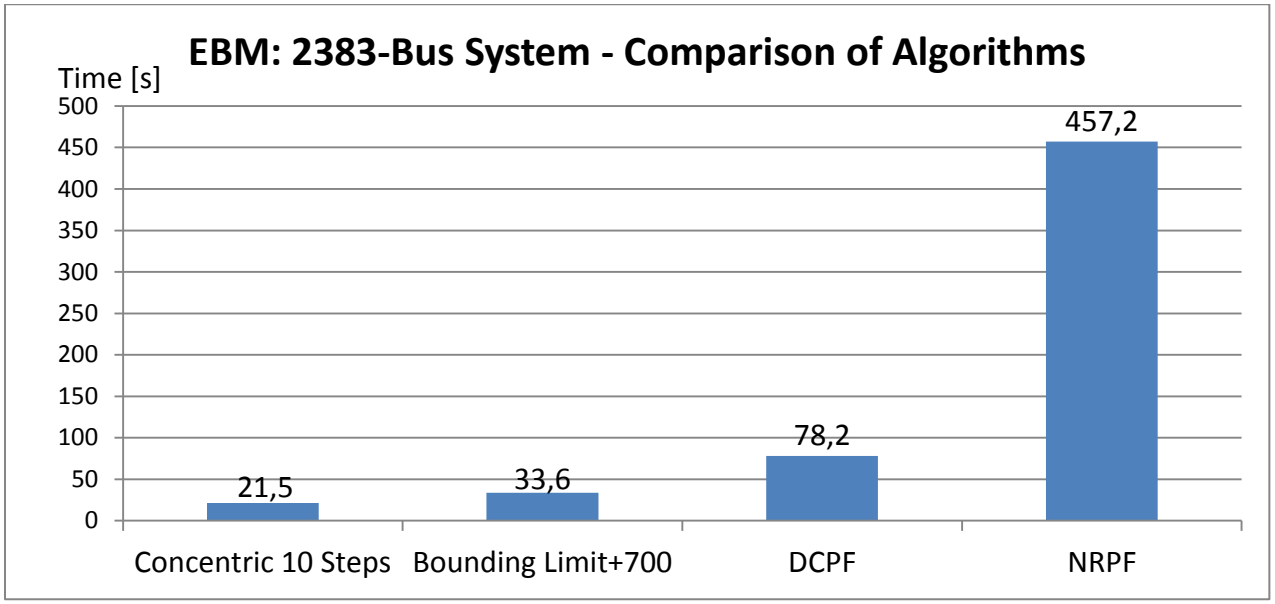

Figure 11: EBM: 2383-Bus System - Comparison of Algorithms

The graph above illustrates how much time the different algorithms need, for 2396 branch outages. The 500 branches, which will cause islanding of nodes, are left out for all of the algorithms, to make the comparison as just as possible. Even with a max steps setting of 10, the CRM utilizes 70 % less time compared to normal DCPF! When compared to NRPF, the reduction is close to 95 %!

The EBM and the CRM act as screening algorithms with no information regarding voltage levels, and the CRM assumes the impact of an outage has limited geographical impact. When time is a critical factor, algorithms like EBM and CRM give a fast estimate of the most important contingencies. Then a more thorough analysis can be carried out with NRPF or FDPF, starting with the contingency with highest PI.

The CRM and the EBM [limit + 700] arrange the PIs in a similar way. Some of the PIs differ a bit for the different step settings. As the number of steps increase for the CRM, the list of the most important contingencies becomes more stable. The CRM is carried out by setting both **`start_steps`** and **`max_steps`** to 10 in the `bounding.m` algorithm.

Table 10: CRM: 2383-Bus System - Contingency With Highest PIs

CRM: 2383-Bus System - Contingencies With Highest PIs					
Number	**Branch Outage**				
	Step = 2	**Step = 4**	**Step = 6**	**Step = 8**	**Step = 10**
1	440	626	626	626	626
2	626	1404	1404	1404	1404
3	1404	1406	1406	1406	1406
4	1406	2714	2714	2714	2714
5	2714	692	169	169	169
6	2896	1374	51	52	52
7	2862	705	292	51	51
8	426	772	52	292	262
9	427	1373	58	58	292
10	2622	292	262	20	58
11	692	169	264	32	360
12	772	168	64	168	64
13	1921	52	168	262	20
14	482	58	96	264	2562
15	1386	51	32	281	32
16	607	262	296	64	23
17	705	64	20	61	300
18	799	1291	322	225	2862
19	704	20	360	1291	61
20	2299	360	281	360	2109

The graph below displays who much time the CRM needs, as a function of maximum steps.

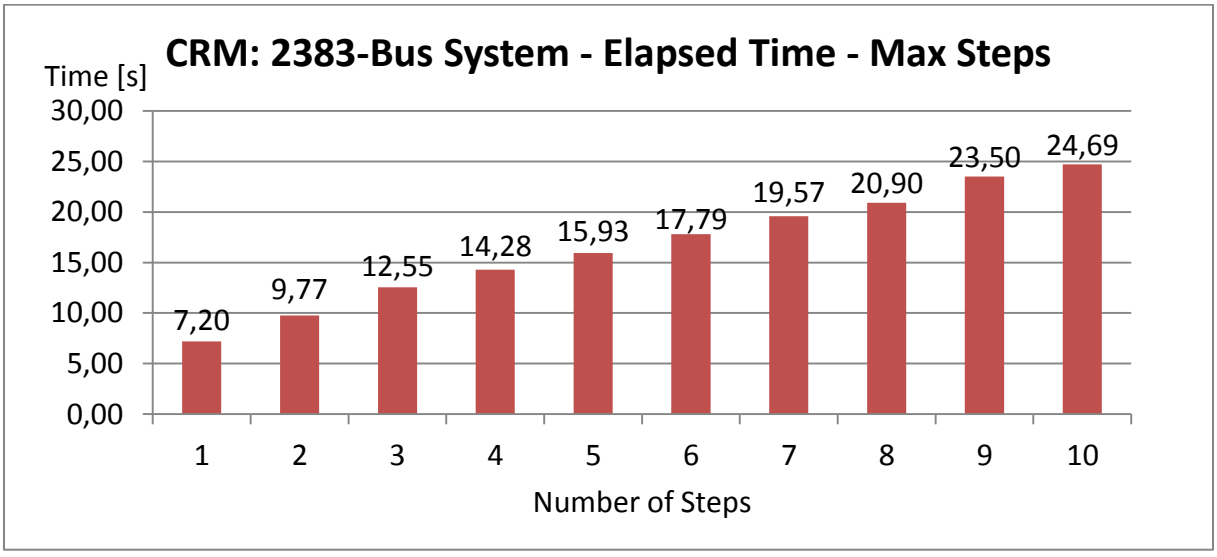

Figure 12: CRM: 2383-Bus System - Elapsed Time - Max Steps

Few steps will make the algorithm faster, but the risk of not detecting important overloads will increase. There seems to be an approximately linear dependency for the CRM and max steps.

Different values for the variable **`raiseLimit`** are used. The time elapsed and the limit's value is registered. The table below displays the results for some arbitrary values of "raise limit".

Table 11: Time & Limit Dependency of Raise Limit

Raise Limit	Time	Limit
0	51.1	-1.12E-05
10	33.7	3.40E-04
50	34.5	3.97E-04
100	35.6	4.16E-04
200	33.5	9.49E-04
300	31.5	2.12E-03
400	30.5	4.02E-03
500	31.7	5.65E-03
700	32.5	8.61E-03
1000	35.0	5.65E-03
1300	37.3	2.09E-02
2309	42.8	7.11E-02

The variable "Raise Limit" in the table above is not scaled evenly; the values are selected to show the tendency of the variable.

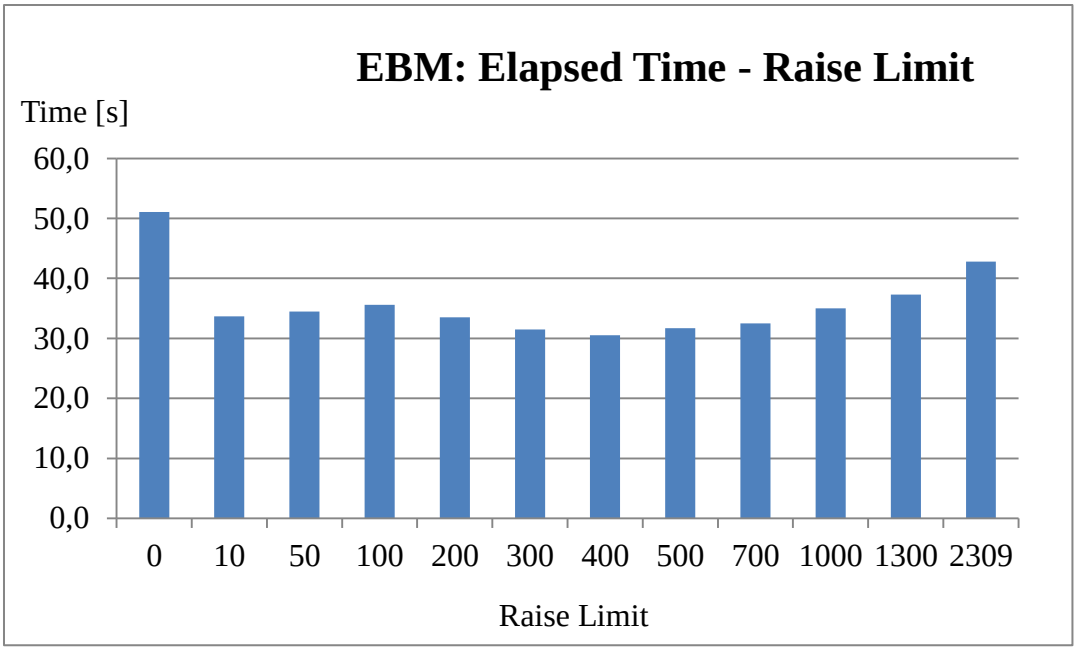

Figure 13: EBM: Elapsed Time - Raise Limit

The elapsed time as function of raise limit is plotted in the graph above. When “raise limit” is 0, the EBM will traverse all the nodes during many steps. This is not optimal. When the limit is raised by 10, the time consumed is substantial lower. However, then the limit is raised to 100 the elapsed time will increase a bit again. The graph indicates that when the limit is raised by 400, the time consumption is 30 s. This is close to the optimal setting for this system. When the limit is raised further from 400, the time consumed increases. For the alternative 2309, all the branches are calculated at once, and there is no need for stepping. This is the reason why elapsed time is less compared to the case where "raise limit" is 0.

6.5 Single DC Contingency Compensation Algorithm

6.5.1 Large System

Since both EBM and CRM have been implemented, it is of interest to compare with SDCCCA, which utilizes the post-compensation technique. It takes only 5.9 seconds to test for 2396 branch outages! This means that the SDCCCA will be faster than the CRM. Even with maximum setting of just one step, it takes 7.2 seconds as discussed previously. A maximum step of one, for the CRM, will only produce a rough estimate, and it will hardly be

sufficient. The SDCCCA on the other hand, will provide information about all the voltage angles in the power system. A short list of the highest PIs can be found in the table below:

Table 12: SDCCA: Highest PIs

SDCCA: Highest PIs	
Branch Outage	**PI**
2562	1882
772	1596
661	729
705	653
662	381
1256	328
2299	56
1373	52
169	39

As can be seen from the table above, there are just a couple of contingencies which cause very high PIs.

6.6 Generator Outage Algorithm

6.6.1 Large System

The GOA has been tested at the large 2383-bus system. The algorithm runs through all generator outages in a short time. It takes only 483 ms to run through 326 generator outages.

It takes 317 ms to pre-calculate the needed inverse columns of **B**, hence more than half of the time. This is illustrated in the diagram below.

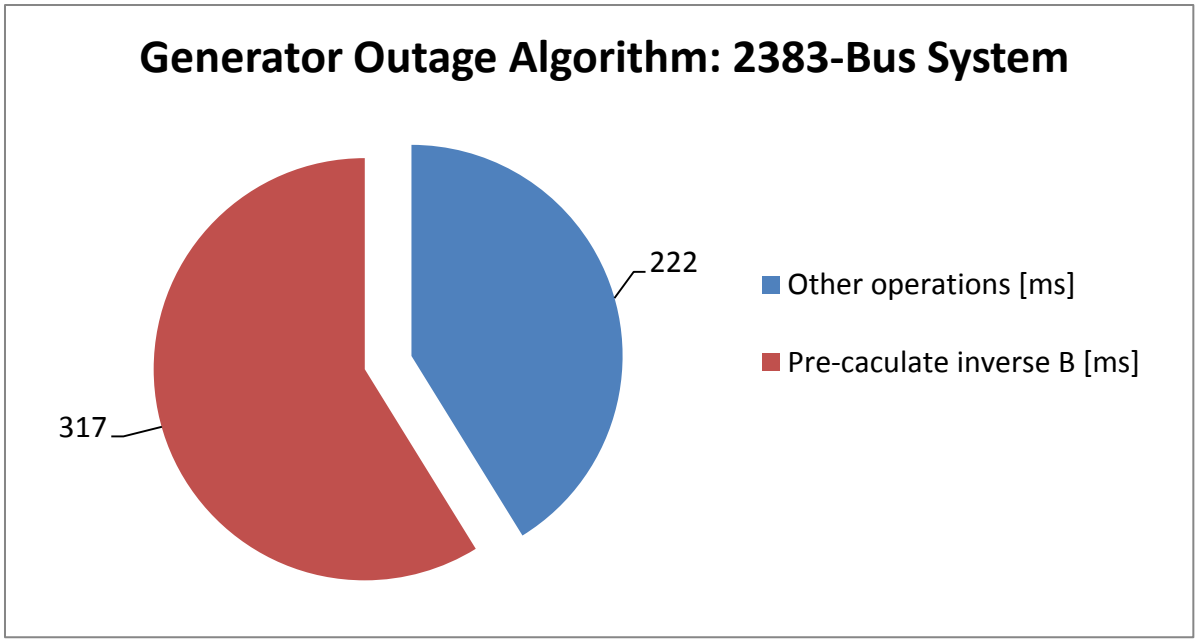

Figure 14: Generator Outage Algorithm - 2383-Bus System

The most important contingencies are displayed in the table below. There are some initial overloads for the polish system, thus many of the generator outages have a PI in the interval [12-16]. As can be seen, only a few generator outages have a PI above this interval. These contingencies should be studied more detailed with ACPF.

Table 13: Generator Outage Algorithm: 2383-Bus System - Highest PIs

Generator Outage Algorithm: 2383-Bus System - Highest PIs		
Generator Index	**Bus**	**PI**
32	127	23.02
33	131	22.26
81	494	21.08
188	1537	18.14
208	1674	17.79
250	1921	17.60
55	281	17.49
83	515	17.40
264	1995	17.24
41	182	17.20

7 Overall Discussion

7.1 Final Comparison of Single Contingency Algorithms

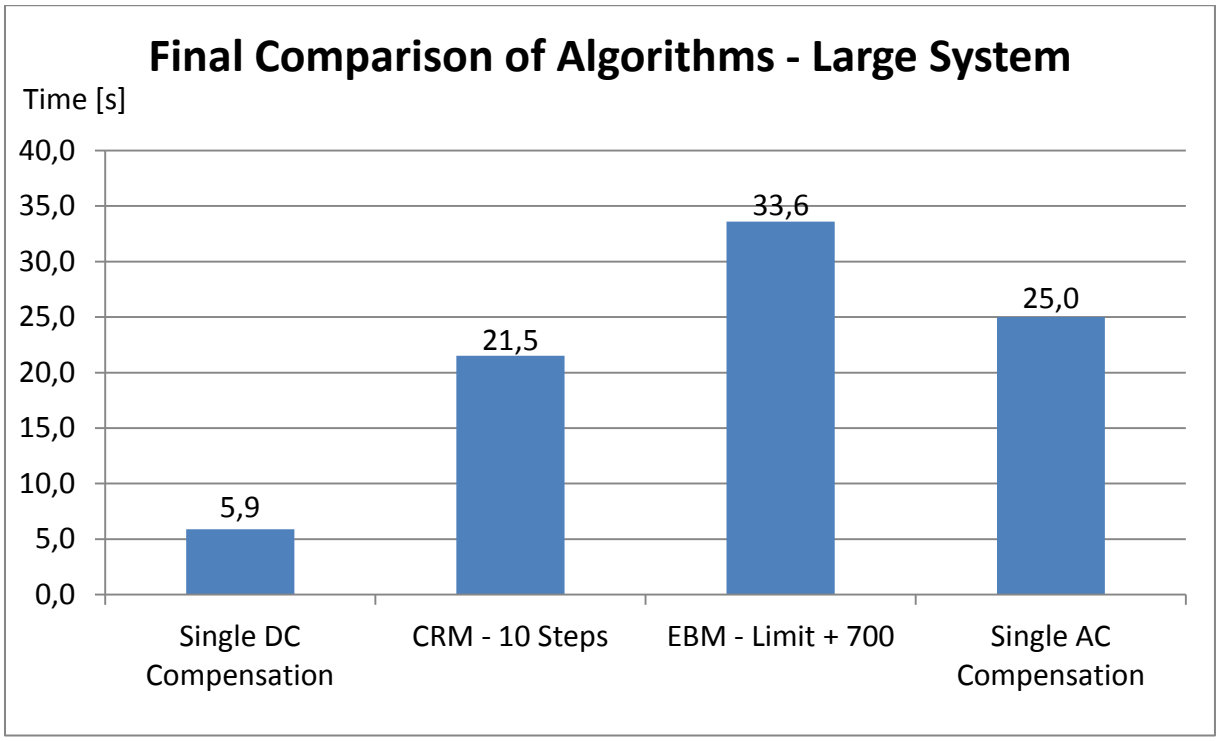

Figure 15: Final Comparison of Algorithms - Large System

The graph above illustrates the performance of the single contingency analysis algorithms implemented. The analysis is done at the Polish 2383-bus system. The SDCCCA is much faster than the other algorithms. The CRM and EBM must do several steps, and the EBM must check the boundary criterion several times during the analysis. Here, 10 steps are used for the CRM, fewer steps will decrease the time consumption, but the CRM will remain slower compared to the SDCCCA. In addition, the SDCCCA calculates all the angle values for all the nodes. Thus this method is faster than the EBM and CRM, and it provides a more detailed output. The SACCCA, which needs 25 s, also demonstrates the benefit of the compensation technique. This method is faster than the EBM! One of the reasons why the EBM does not perform well is that many of the lines are close to overload. In addition, some of the reactance values are quite small and this will cause convergence problematic.

7.2 The Branch Limit Issue

The short-term thermal capacity ratings of the branches are used as limits when doing contingency analysis. However, this may not be realistic. Because of protection settings, and thermal capacity of other equipment, for instance transformers, the branches' thermal capacity limits may be too large. Setting the limits too low will result in many overloads, and the system might not even be N-1 secure, whereas setting them too large will damage equipment. The limits of the branches must be set by experienced system operators in advance. When all the limits are decided, these can be implemented easily into existing contingency algorithms.

7.3 The Limitation of Static Analysis

A power flow is just a snap-shot of the system's state for a given time. The dynamic issues are also important to consider when it comes to power system reliability. This is not the scope of this report. Here, the main emphasis is at the new steady state condition establishing after an outage of a component and try to model this as fast and as possible with reasonable accuracy. Static reactance values may not always be the case. When power flow changes some of the reactance values may change too [4]. This will make DC algorithms, based upon linear analysis, inaccurate. However, if the dynamic of changing reactance values can be modelled in the algorithms, it will provide a better estimate, but this presuppose that the reactance values can be modelled as function of load.

7.4 Performance Index

The strategy of calculating the scalar value of the PI, only when there is an overload, is an elegant way of avoiding the "masking error" problematic. Whenever there is an overload, the index will increase with at least +1, and the rest of the increase is due to the quadratic increase. For instance, an overload of 50 %, will give a scalar value of 2.25.

However, there is a chance, that lines which are identified as safe with DC algorithms will be overloaded when controlling with ACPF. This can be counteracted by adding a sufficient margin to the flow limits of the branches. In the scripts of this report, there is added a 5 % margin. This might not be enough, when the X/R-ratio of a power system is relative small.

The PI is useful when it comes to a quick understanding of the seriousness of a contingency. A scalar number is easy to interpret; higher value means higher consequence. However, the contingencies with highest PIs should be investigated more thoroughly by using ACPF. By doing this, a better physical understanding of the contingencies' consequences can be obtained.

8 Conclusion

The primal version of the fast decoupled power flow has a better convergence characteristic than the dual one, for power systems with a low X/R-ratio.

Indirect ranking of contingencies using a PI has been implemented in the DC contingency analysis algorithms. A modification is done to the method, thus only overloaded branches contribute to the index, and hence it prevents the phenomena known as "masking". The contingencies with highest PIs should be further investigated by ACPF.

Linear sensitivity factors have been used in several of the algorithms. They can be calculated when they are needed, and the factors do not occupy a great part of memory in a modern computer. LODF are used in the SACCCA. It takes only 2.4 s to build the LODF, and it needs only 64 MB of computer memory for the large system. PTDF are used in the GOA. The GOA uses 483 ms to run through 326 generator outages. SF are used in EBM and CRM. Pre-calculation of inverse columns, saved a great amount of time for the large Polish system.

Both EBM and CRM are faster than normal DC power flow. The EBM and the CRM are best used when only the largest flow violations are of interest. For the large Polish 2386-bus system, EBM + 700, uses 57% less time than DCPF. The time reduction for the CRM with max 10 steps, compared to DCPF, is 72.5%. The option to raise the limit for the EBM code is an important feature. It leads to fever expansion steps and faster convergence.

SDCCCA is the fastest algorithm. It solves 2396 outages of the large power system in just 5.9 s, and it uses 92.5% less time than DCPF! The post-compensation technique gives a more detailed output, and is faster than the EBM and the CRM.

The DDCCCA is effective, but the number of combinations becomes high for double contingency analysis and large systems. Test of ten branches against all unique combinations takes 75 s and 23 905 combinations. To test all possible combinations will take hours. Thus, it is recommended that DDCCCA only test for the branches with highest initial power flow, against all possible outage combinations.

The SACCCA is the only algorithm which utilizes AC analysis. It has been discovered, that running a base case flow before the contingency analysis starts, is of great importance. This will save thousands of iterations for the larges system because the largest mismatches are

corrected faster, rather than run the entire analysis using flat start. Most contingencies will converge within 4 iterations with a tolerance limit of 1%. The SACCCA solves the large Polish system in just 25 s! The NRPF will use about 476 s and the FDPF 207 s. Hence, the SACCCA uses 95% less time than the NRPF. Even the EBM used more time than the SACCCA when testing the large power system.

Efficient power flow algorithms are useful when time frame is limited. However, it is important to know the different algorithms' limitations. DC algorithms should not be used for systems with low X/R-ratio for instance, and the reactance values might change during change of flows. This is a dynamic phenomenon that must be taken into account when using DC algorithms where reactance values are assumed static.

9 Future Work

The efficient bounding method has been implemented with promising results when compared to normal DC. There is a more updated version, the complete bounding method [10], which also estimates the change in reactive power flow and phase angles. There was not enough time to test the complete bounding method. Thus it would be of interest to find out how the complete version compare to the other algorithms. However, the SACCCA will be faster for sure.

The adaptive localization method has not been implemented, but the importance of not using a flat start when running the SACCCA, has been inspired by this method. The implementation of the adaptive localization algorithm should make the SACCCA faster.

The reactive power limits has not been enforced in this project. In order to make the AC contingency analysis algorithm more robust, it should be implemented.

A more advanced generator outage algorithm should be implemented. More generators should share the burden of lost generation. PTDF can model this by the use of superposition as explained in the theory chapter.

The DDCCCA is efficient, but the possible combinations increase rapidly for the largest power systems. Techniques for updating the LU-factors used in the second for loop, based on the initial ones, should be investigated. This will make the double outage algorithm even more efficient.

.

List of References

1. Schavemaker, P. and L.V.D Sluis. *electrical power system essentials*. 2008. The Atrium, Chishester: Wiley.
2. Kirchen, D.S. *Do Investments Prevent Blackouts?* Power Engineering Society General Meeting. 2007: p. 1-5. DOI: 10.1109/PES.2007.385653.
3. Adjacency matrix https://en.wikipedia.org/wiki/Adjacency_matrix. Mars 2015.
4. Fosso, O.B. *ELK-14 Transmisson System Operating Planning*. 2014. Trondheim: NTNU. Course materials
5. Saadat, H. *Power System Analysis Third Edition*, 2010. USA: PSA Publishing
6. Stott, B. and O. Alsac. *Fast Decoupled Load Flow*. 1973. Power Systems Laboratory. University of Manchester Institute of Science and Technology, Manchester, U.K.
7. Amerongen, R.A.B. *A General-Purpose Version of the Fast Decoupled Loadflow*. 1989. Vol. 4(2). IEEE Transactions on Power Systems.
8. Monticelli, A. et. al. *Fast Decoupled load flow: Hypothesis, Derivations, and Testing*. 1990. Vol. 5(4). IEEE Transactions on Power systems, Vol. 5(4).
9. Brandwajn, V. *Efficient Bounding Method for Linear Contingency Analysis*. 1988. Vol. 3(1). IEEE transactions on Power Systems.
10. Brandwajn, V. and Lauby, M.G. *Complete Bounding Method for AC Contingency Screening*. 1989. Vol. 4(2). IEEE Transactions on Power Systems.
11. Zaborszky, J. et. al. *Fast Contingency Evaluation Using Concentric Relaxation*. 1980. Vol. PAS-99(1). IEEE Transactions on Power Apparatus and Systems.
12. Monticelli, A., et. al. *Fast Decoupled Load Flow: Hypothesis, Derivations and Testing*. 1990. Vol. 5(4). IEEE Transactions on Power Systems.
13. Ejebe, G.C. et. al. *An Adaptive Localization Method for Real-Time Security Analysis*. 1992. Vol. 7(2). Transactions on Power Systems.
14. Balu, N. et. al. *On-Line Power System Security Analysis*. Proceedings of the IEEE. 1992. Vol. 80(2).
15. Alsac, O. et. al. *Sparsity-Oriented Compensation Methods for Modified Network Solutions*. 1983. IEEE Transactions on Power Apparatus and Systems. Vol. PAS-102(5).
16. Ejebe, G.C. and B.F Wollenberg. *Automatic Contingency Selection*. 1979. IEEE Transactions on Power Apparatus and Systems. Vol.PAS-98(1).

17. Wollenberg, B.F and Wood, A.J. *Power Generation, Operation, and Control*. 2. edition 1996. United States of America: John Wiley & Sons, Inc.
18. Wollenberg, B.F. et. al. Power *Generation, Operation, and Control*. 3. edition. 2014. New Jersey: John Wiley & Sons, Inc.
19. MATLAB. Documentation. http://se.mathworks.com/products/datasheets/pdf/matlab.pdf. April 2015.
20. Fallan, B.T and Line, S. *Introduksjon til MATLAB*. 2003. Trondheim: Tapir akademiske Forlag.
21. Zimmerman, R.D. and C.E. Murillo-Sánchez. MATPOWER 5.0b1 User's Manual. 2014. http://www.pserc.cornell.edu/matpower/manual.pdf
22. Pedersen, R.F. *Screening methods for probabilistic reliability assessment*. 2012. Trondheim: NTNU
23. Pedersen, R.F *Prospective and Efficient Techniques for Model Reduction in Reliability Calculations*. 2013. Trondheim: NTNU.
24. Roska, G. *Efficient Power Flow Algorithms for Risk Management in Electrical Power Systems*. Specialization Project December 2014. Trondheim: NTNU.
25. 118-bus system. http://www.al-roomi.org/power-flow/118-bus-system. May 2015.
26. 24-bus system. http://www.scielo.org.co/scielo.php?pid=S1794-91652011000100001&script=sci_arttext. Mars 2015.

Appendix A - Linear Sensitivity Factors

Power Transfer Distribution Factors

The following derivation of the PTDF are based upon reference [18].

The PTDF gives the fraction of the transferred power, from sending bus s to receiving bus r, that ends up flowing on line l . The PTDF spans from -1 to +1.

$$PTDF_{s,r,l} = \frac{\Delta f_l}{\Delta P_{s\,to\,r}} \tag{A.1}$$

The new flow becomes:

$$\widehat{f_l} = f_l^0 + PTDF_{s,r,l}\Delta P_{s\,to\,r} \tag{A.2}$$

For transfer in the opposite direction:

$$PTDF_{r,s,l} = -PTDF_{s,r,l} \tag{A.3}$$

The incremental DC power flow can be expressed as:

$$\Delta\theta = [X]\Delta P_{s\,to\,r} \tag{A4}$$

The phase angle changes becomes:

$$\begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \vdots \\ \\ \\ \Delta\theta_{n\,\mathrm{bus}} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & & X_{1,n\,\mathrm{bus}} \\ X_{21} & X_{22} & & & \\ \vdots & & \ddots & & \\ & & & & \\ X_{n\,\mathrm{bus},1} & & & & X_{n\,\mathrm{bus},n\,\mathrm{bus}} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ +1 \\ -1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ s \\ r \\ \\ \\ \end{matrix}$$

Figure 16: Derivation of PTDF - DCPF

The phase angle change on bus i and j are then:

$$\Delta\theta_i = X_{is} - X_{ir} \tag{A.5}$$

$$\Delta\theta_j = X_{js} - X_{jr} \tag{A.6}$$

The change in flow on the line l is:

$$\Delta f_l = \frac{1}{x_l}(\Delta\theta_i - \Delta\theta_j) \tag{A.7}$$

$$\Delta f_l = \frac{1}{x_l}((X_{is} - X_{ir}) - (X_{js} - X_{jr})) \tag{A.8}$$

Then, the PTDF is:

$$PTDF_{\ s,r,l} = \frac{1}{x_l}((X_{is} - X_{ir}) - (X_{js} - X_{jr})) \tag{A.9}$$

- If i is the reference bus, $X_{is} = 0$ & and $X_{ir} = 0$
- If j is the reference bus, $X_{js} = 0$ & and $X_{jr} = 0$
- If s is the reference bus, $X_{is} = 0$ & and $X_{js} = 0$
- If r is the reference bus, $X_{ir} = 0$ & and $X_{jr} = 0$

The PTDF depends only on the network parameters and is not affected by the loading or voltages on the network. In addition, the PTDF does not depend on the location of the reference bus in the network

There is no loop in the network that allows power to flow from s to r with some of that power transfer passing through line k if PTDF = 0. When the PTDF = 1, it indicates that all of the transferred power from s to r must flow through line l. A PTDF = 1 also indicates that line, if opened, will cause islanding.

Line Outage Distribution Factors

When monitoring line l after an outage on line k the LODF is [18]:

$$LODF_{l,k} = \frac{\Delta f_l}{f_k^0} \tag{A.10}$$

The new flow then becomes (f_l^0 and f_k^0 are the pre-outage flows) :

$$\widehat{f_l} = f_l^0 + LODF_{l,k} f_k^0 \tag{A.11}$$

In order to simulate the opening of line k, an injection is added into bus n and bus m respectively. Line k is connected to other lines in the remainder of the system through breakers at bus n and m.

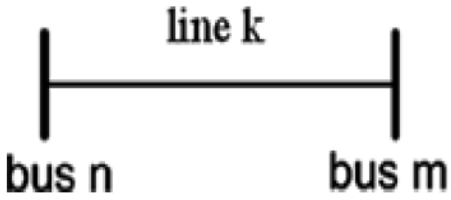

Figure 17: Derivation of LODF - line k from bus n to bus m.

Let P_{nm}be the original power flowing on line k from bus n to bus m . ΔP_n is added into bus n and ΔP_n is added into bus m. This will result in a new flow $\hat{P}_{nm}$ on line k .

The opening of line k can be simulated if:

$$\Delta P_n = \hat{P}_{nm} \tag{A.12}$$

$$\Delta P_m = -\hat{P}_{nm} \tag{A.13}$$

This means that all of the injected power into bus n flows in line k and out of bus m; there will be no flows through the breakers.

$\hat{P}_{nm}$can be expressed as:

$$\hat{P}_{nm} = P_{nm} + PTDF_{n,m,k} \Delta P_n \tag{A.14}$$

Setting $\Delta P_n = \hat{P}_{nm}$ in the formula above gives:

$$\hat{P}_{nm} = P_{nm} + PTDF_{n,m,k} \hat{P}_{nm} \tag{A.15}$$

$$\hat{P}_{nm}(1 - PTDF_{n,m,k}) = P_{nm} \tag{A.16}$$

$$\hat{P}_{nm}\left(\frac{1}{1-PTDF_{n,m,k}}\right)P_{nm} \tag{A.17}$$

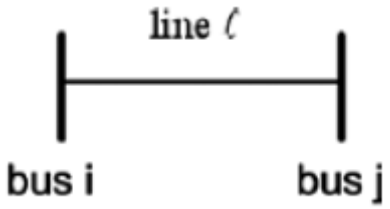

Figure 18: Derivation of LODF - line l from bus i to bus j

The change in flow one line l from i to j in the same power system is then:

$$\Delta f_l = PTDF_{n,m,l}\hat{P}_{nm} = PTDF_{n,m,l}\left(\frac{1}{1-PTDF_{n,m,k}}\right)P_{nm} \tag{A.18}$$

Thus, the LODF giving the change in flow on line l becomes:

$$LODF_{l,k} = PTDF_{n,m,l}\left(\frac{1}{1-PTDF_{n,m,k}}\right) \tag{A.19}$$

$$\Delta f_l = LODF_{l,k}P_{nm} \tag{A.20}$$

Setting $P_{nm}$to f_k^0 gives the original formula:

$$\widehat{f_l} = f_l^0 + LODF_{l,k}f_k^0 \quad \text{Q.E.D.} \tag{A.21}$$

- This way of calculating LODF is done in MATPOWER.

Sensitivity Factors

The derivation of the sensitivity factor follows the same notation system as in the chapter of LODF. It is also similar to the derivation of the LODF. The formulas presented below are derived from the figures in the chapter of LODF.

The change in the phase angles at node n and m can be written [17]:

$$\Delta\theta_n = X_{nn}\Delta P_n + X_{nm}\Delta P_m \quad \text{(A.22)}$$

$$\Delta\theta_m = X_{mn}\Delta P_n + X_{mm}\Delta P_m \quad \text{(A.23)}$$

The outage modelling criteria requires that the incremental injections ΔP_n and ΔP_m equal the power flowing over the outaged line after the injections are imposed:

$$\hat{P}_{nm} = \Delta P_n = -\Delta P_m \quad \text{(A.24)}$$

$$\hat{P}_{nm} = \frac{1}{x_k}(\Delta\hat{\theta}_n - \Delta\hat{\theta}_m) \quad \text{(A.25)}$$

$$\Delta\theta_n = (X_{nn} - X_{nm})\Delta P_n \quad \text{(A.26)}$$

$$\Delta\theta_m = (X_{mn} - X_{mm})\Delta P_n \quad \text{(A.27)}$$

$$\Delta\hat{\theta}_n = \theta_n + \Delta\theta_n \quad \text{(A.28)}$$

$$\Delta\hat{\theta}_m = \theta_m + \Delta\theta_m \quad \text{(A.29)}$$

$$\hat{P}_{nm} = \frac{1}{x_k}\left(\Delta\hat{\theta}_n - \Delta\hat{\theta}_m\right) = \frac{1}{x_k}(\theta_n - \theta_m) + \frac{1}{x_k}(\Delta\theta_n - \Delta\theta_m) \quad \text{(A.30)}$$

Utilizing that $X_{nm} = X_{mn}$ due to symmetry gives:

$$\hat{P}_{nm} = P_{nm} + \frac{1}{x_k}(X_{nn} + X_{mm} - 2X_{nm})\Delta P_n \quad \text{(A.31)}$$

Then using the fact that $\hat{P}_{nm}$ is set to ΔP_n:

$$\Delta P_n = \left[\frac{1}{1 - \frac{1}{x_k}(X_{nn} + X_{mm} - 2X_{nm})}\right] P_{nm} \quad \text{(A.32)}$$

The sensitivity factor δ is the ratio of change in phase angle θ, anywhere in the system, to the original power P_{nm} flowing over a line nm before its outage.

$$\delta_{i,nm} = \frac{\Delta\theta_i}{P_{nm}} \quad \text{(A.33)}$$

If neither n nor m is the system reference bus, two injections, ΔP_n and ΔP_m , are imposed at buses n and m, respectively. This gives a change in phase angle at bus i equal to:

$$\Delta\theta_i = X_{in}\Delta P_n + X_{im}\Delta P_m \tag{A.34}$$

The relationship $-\Delta P_n = \Delta P_m$ gives:

$$\delta_{i,nm} = \frac{\Delta P_n(X_{in}-X_{im})}{\Delta P_n(1-\frac{1}{x_k}(X_{nn}+X_{mm}-2X_{nm}))} = \frac{(X_{in}-X_{im})x_k}{x_k-(X_{nn}+X_{mm}-2X_{nm})} \tag{A.35}$$

If either n or m is the reference bus, only one injection is made:

$$\delta_{i,nm} = \frac{X_{in}x_k}{(x_k-X_{nn})} \quad m = ref \tag{A.36}$$

$$\delta_{i,nm} = \frac{-X_{im}x_k}{(x_k-X_{mm})} \quad n = ref \tag{A.37}$$

$\delta_{i,nm}$= 0 if i is the reference bus. (A.38)

The connection between sensitivity factors and LODF for a branch between bus i and bus j, is simply:

$$LODF_{l,k} = \frac{\Delta f_l}{f_k^0} = \frac{\frac{1}{x_l}(\Delta\theta_i-\Delta\theta_j)}{f_k^0} = \frac{1}{x_l}\left(\frac{\Delta\theta_i}{P_{nm}} - \frac{\Delta\theta_j}{P_{nm}}\right) = \frac{1}{x_l}\left(\delta_{i,nm} - \delta_{j,nm}\right) \tag{A.39}$$

- Sensitivity factors are used in the implementation of the CRM and the EBM in this report.

Appendix B - MATLAB

defineNamedIndices.m

This is an important script which is needed to run most of the other scripts in this appendix.

```
% Define named indices into bus, gen, branch matrices:
[PQ, PV, REF, NONE, BUS_I, BUS_TYPE, PD, QD, GS, BS, BUS_AREA, VM, ...
    VA, BASE_KV, ZONE, VMAX, VMIN, LAM_P, LAM_Q, MU_VMAX, MU_VMIN] ...
    = idx_bus;

[F_BUS, T_BUS, BR_R, BR_X, BR_B, RATE_A, RATE_B, RATE_C, ...
    TAP, SHIFT, BR_STATUS, PF, QF, PT, QT, MU_SF, MU_ST, ...
    ANGMIN, ANGMAX, MU_ANGMIN, MU_ANGMAX] = idx_brch;

[GEN_BUS, PG, QG, QMAX, QMIN, VG, MBASE, GEN_STATUS, PMAX, PMIN, ...
    MU_PMAX, MU_PMIN, MU_QMAX, MU_QMIN, PC1, PC2, QC1MIN, QC1MAX, ...
    QC2MIN, QC2MAX, RAMP_AGC, RAMP_10, RAMP_30, RAMP_Q, APF] = idx_gen;
```

IslandAndParallel.m

This is also an important script which are used in several of the other scripts in this appendix. The algorithm will search through the MATLAB case-files and indentify which buses which will be islanded if the branch connected to it is outaged. It will also indentify parallel branches, which must be handled correctly in the compensation process.

```
fBus = branch(:,F_BUS);
tBus = branch(:,T_BUS);
count = 0;
% Find isolated nodes:
for i = 1:nb
Xf = find(fBus == i);
Xt = find(tBus == i);
X = [Xf;Xt];
if length(X) < 2
    count = count + 1;
    temp(count) = i;
end
end
countIsland = count;
% Find branch indeces of isolated nodes
```

```
K = zeros(count,1);
for i = 1:count
    if ismember(temp(i),fBus)
    K(i) = find(fBus == temp(i));
    else
    K(i) = find(tBus == temp(i));
    end
end
indexIsland = sort(K);
% Find branch indexes of parallel lines:

index = zeros(nl,1);
countParallel = 0;
for i = 2:nl
    if fBus(i) == fBus(i-1) && tBus(i) == tBus(i-1)
        index(i) = i;
        index(i-1) = i-1;
        countParallel = countParallel + 2;
    end
end % (Asumes the parallel lines are listed stright after each other)
indexParallel = nonzeros(index);
```

---

fdlfPrimal.m

This is an updated version of the code used in the specialization report of 2014 [*]. The scaling of the X/R ratio is made better. This code is simple and just illustrate the FDPF for a simple system.

```
clc
clear
format shortg

%% Primal-version (BX version starting with P-iteration)
%------------v------del----Pg-----Qg-----Pd-----Qd----type--
bus = [1    1.01     0.00    0.00    0.00    0.00    0.00     1 ;
       2    1.02     0.00    1.00    0.00    0.00    0.00     2 ;
       3    1.02     0.00    0.80    0.00    0.00    0.00     2 ;
       4    1.00     0.00    0.00    0.80    0.00    0.00     3 ;
       5    1.00     0.00    0.00    0.00    0.40    0.00     3 ;
       6    1.00     0.00    0.00    0.00    0.50    0.00     3 ;
       7    1.00     0.00    0.00    0.00    0.30    0.00     3 ;
       8    1.00     0.00    0.00    0.00    0.40    0.00     3 ;
       9    1.00     0.00    0.00    0.00    0.40    0.50     3];

%-----from--to-----r--------x---
line = [1    2    0.0546    0.2112 ;
        2    3    0.0218    0.0845 ;
        3    4    0.0328    0.1267 ;
```

```
        4    5    0.0546    0.2112 ;
        5    6    0.0218    0.0845 ;
        6    7    0.0328    0.1267 ;
        7    8    0.0546    0.2112 ;
        8    9    0.0218    0.0845 ;
        9    1    0.0328    0.1267 ;
        1    4    0.0546    0.2112 ;
        5    7    0.0218    0.0845 ;
        8    1    0.0328    0.1267 ];

buses = max(bus(:,1));
[temp1,~] = size(line);
lines = temp1;

% Update line matrix with chosen x/r-ratio:
R = 1.0;
X = 0.15;
line(:,3) = line(:,3) .* R;
line(:,4) = line(:,4) .* X;

% Prepare to create ybus-matrix & simplified ybus-matrix using 1/x:
ytemp = zeros(buses,buses);
xtemp = zeros(buses,buses);
for x = 1:lines
    i = line(x,1); % From
    j = line(x,2); % To
    ytemp(i,j) = 1 / (line(x,3) + 1i*line(x,4)); % Admittance value
    ytemp(j,i) = ytemp(i,j); % Symmetric
    xtemp(i,j) = 1/(1i*line(x,4));
    xtemp(j,i) = xtemp(i,j);
end

% Build admittance matrix:
ybus = zeros(buses,buses);
xbus = zeros(buses,buses);
for i = 1:buses
    for j = 1:buses
        if i==j
            ybus(i,j) = sum(ytemp(i,:));
            xbus(i,j) = sum(xtemp(i,:));
        else
            ybus(i,j) = -1*ytemp(i,j);
            xbus(i,j) = -1*xtemp(i,j);
        end
    end
end

% Build b' matrtix:
b = -imag(ybus);
b1 = zeros(buses-1,buses-1);
for i = 1:buses-1
    for j = 1:buses-1
        b1(i,j) = b(i+1,j+1);
    end
end

% Build b" matrix:
b = -imag(xbus);
b2 = zeros(buses-3,buses-3);
for i = 1:(buses-3)
```

```
    for j = 1:(buses-3)
        b2(i,j) = b(i+3,j+3);
    end
end

V = bus(:,2);
del = bus(:,3);
Pg = bus(:,4);
Qg = bus(:,5);
Pd = bus(:,6);
Qd = bus(:,7);
Pspec = Pg-Pd;
Qspec = Qg-Qd;
G = real(ybus); % Conductance
B = imag(ybus); % Susceptance

iter = 0;
tolerance = 0.001;
condition = 1;
while condition == 1;
    P = zeros(buses,1);
    Q = zeros(buses,1);
    iter = iter+1;

% Active power:
for i=1:buses
    for j=1:buses
        P(i) = P(i) + (V(i)*V(j)*(G(i,j)*cos(del(i)-del(j)) ...
               +B(i,j)*sin(del(i)-del(j))));
    end
end

% Delta P divided by V:
for i = 1:(buses-1)
    delP(i,1) = Pspec(i+1)-P(i+1);
    delPbyV(i,1) = delP(i,1) ./ V(i+1,1);
end

% Correction vector, angle:
    deldel = b1 \ delPbyV;

% Update voltage angles:
for i = 1:(buses-1)
    del(i+1,1) = del(i+1,1) + deldel(i,1);
end

% Reactive power:
for i=1:buses
    for j=1:buses
        Q(i) = Q(i)+ (V(i)*V(j)*(G(i,j)*sin(del(i)-del(j)) ...
               -B(i,j)*cos(del(i)-del(j))));
    end
end

% Delta Q divided by V:
c=0;
for i=1:buses
    if bus(i,8) == 3
        c = c+1;
        delQ(c,1) = Qspec(i) - Q(i);
```

```
        delQbyV(c,1) = delQ(c,1) ./ V(i,1);
    end
end

% Correction vector, voltage magnitudes:
    delv = b2 \ delQbyV;

% Update voltages:
c=0;
for i = 1:buses
    if bus(i,8) == 3
        c = c+1;
        V(i,1) = V(i,1) + delv(c,1);
    end
end

iter
P
Q
V
del

if max(abs(delP)) < tolerance && max(abs(delQ)) < tolerance
    condition = 0;
elseif iter == 30
    condition = 0;
end

end
Ploss = sum(P)
```

---

contTest1.m

This is another code from the former report []. The mistake with the compensation process is corrected.

```
clear
clc

mpc = loadcase(case4gs);

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

% Split up:
[baseMVA, bus, gen, branch] = ...
    deal(mpc.baseMVA, mpc.bus, mpc.gen, mpc.branch);

% Create reference:
```

```
mpopt = mpoption('model','DC');
[results1, success1] = runpf(mpc, mpopt);

Va0 = results1.bus(:,VA) * pi/180;

% Build B-matrix:
[Bbus, ~, ~, ~] = makeBdc(baseMVA, bus, branch);

% Builds index lists for each type of bus (REF, PV, PQ):
[ref, pv, pq] = bustypes(bus, gen);

% Factorization:
Bt = Bbus([pv; pq],[pv; pq]);
[L, U, P, Q] = lu(Bt);

nb = length(bus(:,1));
resVa = zeros(nb,1);

%% Compensation:
% Branch index of line:
b1 = 1;

bf1 = branch(b1,F_BUS);
bt1 = branch(b1,T_BUS);

M = sparse([bf1 bt1],1,[1 -1],nb,1);
M = M([pv;pq]);

invBbusM = Q * (U \ (L \ (P * M )));
z = M' * invBbusM;
dy = Bbus(bf1,bt1);
c = 1 / ((1/dy) + z);

deltaVa = invBbusM * c * M' * Va0([pv;pq]);

VaComp = Va0([pv;pq]) - deltaVa;
VaComp = VaComp * 180/pi;

resVa([pv;pq]) = VaComp;

% New flows:
mpc.branch(b1,BR_STATUS) = 0;
[results2, success2] = runpf(mpc, mpopt);

comparsion = [results2.bus(:,VA) resVa]
```

DCcompAlg.m

This is the corrected version of the corresponding code of the project report [24].

```
clear
clc

mpc = loadcase(case4gs);

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

% Copy:
[baseMVA, bus, gen, branch] = ...
    deal(mpc.baseMVA, mpc.bus, mpc.gen, mpc.branch);

% Create reference:
mpopt = mpoption('model','DC');
[r1, s1] = runpf(mpc, mpopt);

Va0 = r1.bus(:,VA) * pi/180;

% Build B-matrix:
[Bbus,~,~,~] = makeBdc(baseMVA, bus, branch);

% Builds index lists for each type of bus (REF, PV, PQ):
[ref,pv,pq] = bustypes(bus, gen);

% Factorization:
Bt = Bbus([pv;pq],[pv;pq]);
[L, U, P, Q] = lu(Bt);

nb = length(bus(:,1));
nl = length(branch(:,1));

bf0 = branch(:,F_BUS);
bt0 = branch(:,T_BUS);

% Maximum  200% loading:
maxVa = zeros(nl,1);
for i = 1:nl
maxVa(i) = 2.0 * abs(Va0(bt0(i))-Va0(bf0(i)));
end

% Compensation:
success = ones(nl,2);
success(:,1) = 1:nl;
resVa = zeros(nb,nl);
Va = zeros(nl,nl);
for i = 1:nl

bf = branch(i,F_BUS);
bt = branch(i,T_BUS);

M = sparse([bf bt],1,[1 -1],nb,1);
M = M([pv;pq]);

invBbusM = Q * (U \ (L \ (P * M )));
z = M' * invBbusM;
dy = Bbus(bf,bt);
c = 1 / (1/dy + z);
```

```
deltaVa = invBbusM * c * M' * Va0([pv;pq]);

VaComp = Va0([pv;pq]) - deltaVa;

resVa([pv;pq],i) = VaComp;

% Check if angle difference is violated:

for j = 1:nl
    Va(j,i) = abs(resVa(bt0(j),i)-resVa(bf0(j),i));
    if  Va(j,i) > maxVa(j) && Va(j,i) > 0.02 && i~=j
        success(i,2) = 0;
    end
end

end
% Display success matrix:
success
resVa
Va0
Va
```

---

testBoundingAngle.m

This code has been made in order to test that the phase angles are updated correctly in the EBM.

```
%% ~* Test angle updating in the efficient bounding Method*~
clear
clc
format shortg

%% *Choose paramters*
branch_out = 1; % Interruption of this branch
node = 5; % Find out how the voltage-angle of this node change.

%% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

%% System parameters:
casedata = case24_ieee_rts; %; % case2383wp;    case4gs
mpc = loadcase(casedata);
[baseMVA, bus, gen, branch] = ...
    deal(mpc.baseMVA, mpc.bus, mpc.gen, mpc.branch);
[ref, pv, pq] = bustypes(bus, gen);

%% Referene power flow:
mpopt = mpoption('model','DC');
[r1,s1] = runpf(mpc, mpopt);
```

```
f0 = r1.branch(:,PF) ./ baseMVA; % f_pq^0

%% Add zero columns to branch for flows if needed:
if size(mpc.branch,2) < QT
mpc.branch = [mpc.branch zeros(size(mpc.branch, 1), ...
    QT-size(mpc.branch,2))];
end

%% Build B matrices and phase shift injections:
[B, Bf, Pbusinj, Pfinj] = makeBdc(baseMVA, bus, branch);

%% Set values:
nb = size(bus,1); % Number of buses.
stat = branch(:,BR_STATUS); %% Ones at in-service branches.
x = branch(:,BR_X); %% Branch reactances.
indices = find(stat == 0);
x(indices) = NaN; % Disconnected branches.

%% Node lists of the branches:
bf = branch(:,F_BUS);
bt = branch(:,T_BUS);
% LUPQ factorization of B:
[L, U, P, Q] = lu(B([pv;pq],[pv;pq]));

%% Calculate the two inverse X-columns needed for each contingency:
if bf(branch_out) ~= ref
one_bf = sparse(bf(branch_out),1,1,nb,1);
X_bf([pv;pq]) = Q * (U \(L \(P * one_bf([pv;pq]))));
end

if bt(branch_out) ~= ref
one_bt = sparse(bt(branch_out),1,1,nb,1);
X_bt([pv;pq]) = Q * (U \(L \(P * one_bt([pv;pq]))));
end

delta = zeros(1,1);
if node == ref
delta(1) = 0;
elseif bf(branch_out) == ref
delta(1) = (X_bt(node)*x(branch_out))/(x(branch_out)-X_bt(bt(branch_out)));
elseif bt(branch_out) == ref
delta(1) = -(X_bf(node)*x(branch_out))/ ...
    (x(branch_out)-X_bf(bf(branch_out)));
else
delta(1) = (X_bf(node)-X_bt(node))*x(branch_out)/ ...
    (x(branch_out)-(X_bt(bt(branch_out))+ ...
    X_bf(bf(branch_out))-2*X_bf(bt(branch_out))));
end

angle = delta * f0(branch_out);

% New flows:
mpc.branch(branch_out,BR_STATUS) = 0;
[r2,s2] = runpf(mpc, mpopt);

%% Output:
voltageAngle0 = r1.bus(:,VA)
voltageAngle = r2.bus(:,VA)
angleChange = angle * 180/pi
```

```
updatedAngle = voltageAngle0(node) + angleChange
branch_out
node
```

---

generatorOutage.m

This is the script which has been used to test for contingency analysis of generator outages. The algorithm use PTDF.

```
%% ~*Single DC Contingency Analysis - Generator Outage*~
clear
clc
format shortg
tic

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

%% Power system data:
mpc = loadcase(case2383wp);  %   case118  case24_ieee_rts
[bus,gen,branch,baseMVA] = deal(mpc.bus,mpc.gen,mpc.branch,mpc.baseMVA);
% Builds index lists for each type of bus (REF, PV, PQ):
[ref, pv, pq] = bustypes(bus,gen);

nb = length(bus(:,1)); % Number of buses.
ng = length(gen(:,1)); % Number of generators.
nl = length(branch(:,1)); % Number of lines.

%% Create reference (testing):
mpopt = mpoption('model','DC');
[r1, s1] = runpf(mpc, mpopt);

%% Thermal limits:
thermal = zeros(nl,1);
if branch(:,RATE_B) == 0
    % 1.3 x the largest flow:
    thermalMax = max(abs(r1.branch(:,PF))) .* 1.3;
    thermal(:,1) = thermalMax;
    indices = abs(r1.branch(:,PF)) < 100;
    thermal(indices) = 250; % Set capacity to 250 for lines < 100
else
    thermal = (branch(:,RATE_B)) .* 0.95 ;  % 5% margin due to DC.
end

%% Build B matrices and phase shift injections:
[B,~,~,~] = makeBdc(baseMVA, bus, branch);

%% LUPQ factorization of B:
[L,U,P,Q] = lu(B([pv;pq],[pv;pq]));
```

```
%% Calculate the needed inverse columns with consecutive indexing:
timeX = tic;
genList = gen(:,GEN_BUS);
genListUnique = unique(genList); % Might be > 1 generator at each node.
nglu = nnz(genListUnique);
X = zeros(nb,nb);
for i = 1:nglu
    one_node = sparse(genListUnique(i),1,1,nb,1);
    X([pv;pq],genListUnique(i)) = Q * (U \ (L \ (P * one_node([pv;pq]))));
end
timeX = toc(timeX);

%% Branch vectors:
bf = branch(:,F_BUS);
bt = branch(:,T_BUS);

%% Reactance values:
stat = branch(:,BR_STATUS); %% Ones at in-service branches.
indices = find(stat == 0);
x = branch(:,BR_X); %% Branch reactances.
x(indices) = NaN;

%% Generator outage analysis:
newFlow = zeros(nl,1); % New flows for each generator outage.
count = 0; % The number of generator outages tested.
PI = zeros(ng,1); % Performance Index.
for i =  1:ng
    j = gen(i,GEN_BUS);

    if j == ref
        continue; % Slack bus.
    end

    temp = X(:,j);
    PTDF = (temp(bf) - temp(bt)) ./ x;
    newFlow = r1.branch(:,PF) + PTDF .* -r1.gen(i,PG);

    indices = find(abs(newFlow(:)) > thermal);
    ni = nnz(indices);
    for k = 1:ni
        PItemp = (abs(newFlow(indices(k))) / thermal(indices(k)))^2;
        PI(i) = PI(i) + PItemp;
    end

count = count + 1
end

%% Find the highest PI and sort them:
I = find(PI);
nI = nnz(I);
result = zeros(nI,3);
for a = 1:nI
    result(a,1) = I(a);
    result(a,2) = gen(I(a),GEN_BUS);
    result(a,3) = PI(I(a));
end
resultPI = sortrows(result,-3);
totalTime = toc;
```

```
genList
resultPI
timeX
totalTime
whos X

%{
% New flows (testing):
generator = 1;
mpc.gen(generator,GEN_STATUS) = 0;
[r2, s2] = runpf(mpc, mpopt);
newFlowOutput = newFlow
%}
```

---

singleCompensationDC.m

This is an single DC contingency analysis algorithm. It is focused on outage of branches and is best used as an screening algorithm. It is used for comparison between the other algorithms.

```
%% ~*Single DC Contingency Analysis With Compensation*~
clear
clc
format shortg
tic

mpc = loadcase(case2383wp); % case118  case24_ieee_rts

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

% Split up:
[baseMVA,  bus,  gen,  branch]  =  deal(mpc.baseMVA,  mpc.bus,  mpc.gen,
mpc.branch);

nb = length(bus(:,1)); % Number of buses. Used in "IslandAndParallel".
nl = length(branch(:,1));

%% Find isolated nodes and parallel lines:
IslandAndParallel;

% Create reference:
mpopt = mpoption('pf.alg','NR', 'pf.tol',10^-5);
[r1, s1] = runpf(mpc, mpopt);
Va0 = r1.bus(:,VA) * pi/180; % Initial voltage-angles.

%% Thermal limits:
f0 = r1.branch(:,PF) ./ baseMVA; % Initial flow.
thermal = zeros(nl,1);
```

```
if branch(:,RATE_B) == 0
    thermalMax = max(abs(f0)) * 1.3; % 1.3 x the largest flow.
    thermal(:) = thermalMax;
    indices = abs(f0) < (100 / baseMVA);
    % Set capacity to 250 for lines < 100:
    thermal(indices) = 250 / baseMVA;
    thermal = thermal .* 0.95; % 5% margin due to DC.
else
    thermal = (branch(:,RATE_B) ./ baseMVA) .* 0.95; % Thermal limits.
end

%% Determine angle spread limits for each line:
x = branch(:,BR_X); % Branch reactances.
maxAngle = thermal .* x; % Set max angle-spread for each line.

% Build B-matrix:
[Bbus1,~,~,~] = makeBdc(baseMVA, bus, branch);

% Builds index lists for each type of bus (REF, PV, PQ):
[ref, pv, pq] = bustypes(bus, gen);

% Factorization:
Bt = Bbus1([pv; pq],[pv; pq]);
[L0, U0, P0, Q0] = lu(Bt);

nb = length(bus(:,1));
nl = length(branch(:,1));

bf = branch(:,F_BUS);
bt = branch(:,T_BUS);

resVa = zeros(nb,1);
PI = zeros(nl,1);
count = 0;

%% Single DC compensation starts:
for b1 = 1:nl

% If isolated node:
if ismember(b1,indexIsland)
    continue; % b1 = b1 + 1
end

bf1 = branch(b1,F_BUS);
bt1 = branch(b1,T_BUS);

M1 = sparse([bf1 bt1],[1 1],[1 -1],nb,1);
Mindex = M1([pv;pq]);

invBbusM = Q0 * (U0 \ (L0 \ (P0 * Mindex )));
z = Mindex' * invBbusM;

if ismember(b1,indexParallel) % If parallel lines:
    dy = Bbus1(bf1,bt1) / 2;
else
    dy = Bbus1(bf1,bt1);
end

c = 1 / ((1/dy) + z);
```

```
deltaVa = invBbusM * c * Mindex' * Va0([pv;pq]);
VaComp = Va0([pv;pq]) - deltaVa;
resVa([pv;pq]) = VaComp;

angleSpread = resVa(bf) - resVa(bt); % Angle-spread of all branches.

violation = find(abs(angleSpread) > maxAngle); % Find overloaded branches.
for k = 1:nnz(violation)

    if violation(k) == b1 % Is out of service.
        continue;
    end

    tempPI = (angleSpread(violation(k)) / maxAngle(violation(k)))^2 ;

    %if PI(b1) > 10^3 % If huge overload.
    %PI(b1) = 10^3; % Max PI value.
    %break;
    %end

    PI(b1) = PI(b1) + tempPI;

end

count = count + 1
end

[I] = find(PI);
ni = length(I);
resultPI = zeros(ni,2);
for a = 1:ni
    resultPI(a,1) = I(a);
    resultPI(a,2) = PI(I(a));
end
resultPI = sortrows(resultPI,-2);
totalTime = toc;

%% Output:
resultPI
count
totalTime

%{
% New flows (testing):
mpopt = mpoption('model','DC');
%mpopt = mpoption('pf.alg','NR', 'pf.tol',10^-5);
branchOut = 7;
mpc.branch(branchOut,BR_STATUS) = 0;
[r2, s2] = runpf(mpc, mpopt);
resVa = resVa .* 180/pi
% Comment: When the reference flow is AC,
% the results will be a bit different when running a DC algorithm.
% AC gives a more accurate inital voltage-angles.
%}
```

---

doubleCompensationDC.m

This is an double DC contingency analysis algorithm. It is focused on outage of branches and is best used as an screening algorithm.

```
%% ~*Double DC Contingency Analysis With Compensation*~
clear
clc
format shortg
tic

mpc = loadcase(case24_ieee_rts); %  case118  case2383wp

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

% Split up:
[baseMVA, bus, gen, branch] = ...
    deal(mpc.baseMVA, mpc.bus, mpc.gen, mpc.branch);

nb = length(bus(:,1)); % Number of buses. Used in "IslandAndParallel".
nl = length(branch(:,1));

%% Find isolated nodes and parallel lines:
IslandAndParallel;

% Create reference:
mpopt = mpoption('pf.alg','NR', 'pf.tol',10^-5);
[r1, s1] = runpf(mpc, mpopt);
Va0 = r1.bus(:,VA) * pi/180; % Initial voltage-angles.

%% Thermal limits:
f0 = r1.branch(:,PF) ./ baseMVA; % Initial flow.
thermal = zeros(nl,1);
if branch(:,RATE_B) == 0 % If no thermal data present.
    thermalMax = max(abs(f0)) .* 1.3; % 1.3 x the largest flow.
    thermal(:,1) = thermalMax;
    indices = abs(f0) < (100 / baseMVA);
    % Set capacity to 250 for lines < 100 :
    thermal(indices) = 250 / baseMVA;
    thermal = thermal .* 0.95; % 5% margin due to DC.
else
    thermal = (branch(:,RATE_B) ./ baseMVA) .* 0.95; % Thermal limits.
end

%% Determine angle spread limits for each line:
x = branch(:,BR_X); % Branch reactances.
maxAngle = thermal .* x; % Define max angle spread for each line.
```

```
% Build B-matrix:
[Bbus1,~,~,~] = makeBdc(baseMVA, bus, branch);

% Builds index lists for each type of bus (REF, PV, PQ):
[ref, pv, pq] = bustypes(bus, gen);

% Factorization:
Bt = Bbus1([pv; pq],[pv; pq]);
[L0, U0, P0, Q0] = lu(Bt);

nb = length(bus(:,1));
nl = length(branch(:,1));

bf = branch(:,F_BUS);
bt = branch(:,T_BUS);

resVa1 = zeros(nb,nl);
resVa2 = zeros(nb,1);
PI = zeros(nl,nl);
count = 0;

%% Double compensation starts:
for b1 = 1:nl

% If isolated node:
if ismember(b1,indexIsland)
    continue;
end

bf1 = branch(b1,F_BUS);
bt1 = branch(b1,T_BUS);

M1 = sparse([bf1 bt1],[ 1 1],[1 -1],nb,1);
Mindex = M1([pv;pq]);

invBbusM = Q0 * (U0 \ (L0 \ (P0 * Mindex )));
z = Mindex' * invBbusM;

if ismember(b1,indexParallel) % If parallel lines:
    dy = Bbus1(bf1,bt1) / 2;
else
    dy = Bbus1(bf1,bt1);
end

c = 1 / ((1/dy) + z);

deltaVa = invBbusM * c * Mindex' * Va0([pv;pq]);
VaComp = Va0([pv;pq]) - deltaVa;
resVa1([pv;pq],b1) = VaComp;

Bbus2 = Bbus1 + M1*dy*M1';

% Factorization for second loop:
Bt = Bbus2([pv; pq],[pv; pq]);
[L, U, P, Q] = lu(Bt);

%% Second stage:
for b2 = 1:nl
```

```
if b2 <= b1
    continue % Make sure unique combinations.
end

% If isolated node:
if ismember(b2,indexIsland)
    continue;
end

%% Compensation:
bf2 = branch(b2,F_BUS);
bt2 = branch(b2,T_BUS);

M2 = sparse([bf2 bt2],[ 1 1],[1 -1],nb,1);
Mindex = M2([pv;pq]);

invBbusM = Q * (U \ (L \ (P * Mindex )));
z = Mindex' * invBbusM;

% If parallel lines:
if ~ismember(b1,indexParallel) && ismember(b2,indexParallel)
    dy = Bbus2(bf2,bt2) / 2;
elseif ismember(b1,indexParallel) && ismember(b2,indexParallel) && b1 ~= b2
    dy = Bbus2(bf2,bt2) / 2;
else
    dy = Bbus2(bf2,bt2);
end

c = 1 / ((1/dy) + z);

deltaVa = invBbusM * c * Mindex' * resVa1([pv;pq],b1);
VaComp = resVa1([pv;pq],b1) - deltaVa;
resVa2([pv;pq]) = VaComp;

angleSpread = resVa2(bf) - resVa2(bt);

violation = find(abs(angleSpread) > maxAngle); % Find overloaded branches.
for k = 1:nnz(violation)
    if violation(k) == b1 || violation(k) == b2 % Are out of service.
        continue;
    end

    tempPI = (angleSpread(violation(k)) / maxAngle(violation(k)))^2 ;
    PI(b1,b2) = PI(b1,b2) + tempPI;

    %if PI(b1,b2) > 10^5 % If huge overload.
    %PI(b1,b2) = 10^5; % Max PI value.
    %break;
    %end
end
count = count + 1
end
end

%% Find the highest PIs and sort them:
[row,col] = find(PI);
nr = length(row);
result = zeros(nr,3);
```

```
for a = 1:nr
    result(a,1) = row(a);
    result(a,2) = col(a);
    result(a,3) = PI(row(a),col(a));
end
resultPI = sortrows(result,-3);
time = toc;

%{
violation_branches = violation
violation_value = angleSpread(violation)
%}

%% Output:
indexIsland
indexParallel
resultPI
count
time

%{
% New flows:
mpopt = mpoption('pf.alg','NR', 'pf.tol',10^-5);
mpc.branch([7 24],BR_STATUS) = 0;
[r2, s2] = runpf(mpc, mpopt);

resVa2 = resVa2 .* 180/pi
%}
```

---

compensationAC.m

This is an AC single contingency algorithm. It will give information about voltage-angles and LODF is used to find the new flows fast.

```
%% ~*Fast Decoupled Load Flow Contingency Analysis With Compensation*~
clear
clc
format shortg

%% Start stopp watch:
tic; % Start.
t0 = tic;

%% *Control Parameters*
tol = 0.01; % 1% seems like a good trade-off between speed and accuracy.
max_it = 10;

% Define named indices into bus, gen, branch matrices:
defineNamedIndices;
```

```
%% Power system data:
mpc = loadcase(case3120sp);   %      case24_ieee_rts  case2383wp
[bus,gen,branch] = deal(mpc.bus,mpc.gen,mpc.branch);
[ref, pv, pq] = bustypes(bus,gen); % Make lists of node types.
nb = length(bus(:,1));
nl = length(branch(:,1));

%% Find isolated nodes and parallel lines:
IslandAndParallel;

%% Reference Flow:
mpopt = mpoption('pf.alg','NR', 'pf.tol',10^-5 );
[r1,s1] = runpf(mpc, mpopt);
[bus,gen,branch,baseMVA] = deal(r1.bus,r1.gen,r1.branch,r1.baseMVA);

%% initial state:
on = find(gen(:, GEN_STATUS) > 0); % Which generators are on?
gbus = gen(on, GEN_BUS); % What buses are they at?

V0  = bus(:, VM) .* exp(sqrt(-1) * pi/180 * bus(:, VA));
vcb = ones(size(V0)); % Create mask of voltage-controlled buses.
vcb(pq) = 0;          % Exclude PQ-buses.
k = find(vcb(gbus));  % In-service gens at v-c buses.
V0(gbus(k)) = gen(on(k), VG) ./ abs(V0(gbus(k))).* V0(gbus(k));
Va0 = angle(V0); % Initial voltage-angle magnitude.
Vm0 = abs(V0); % Initial voltage magnitude.

%% Build admittance matrix:
[Ybus, Yf, Yt] = makeYbus(baseMVA, bus, branch);

%% Series admittance, Ys:
stat = branch(:,BR_STATUS); % Ones at in-service branches.
Ys = stat ./ (branch(:,BR_R) + 1j * branch(:, BR_X));

%% Compute complex bus power injections (generation - load):
Sbus = makeSbus(baseMVA, bus, gen);

%% B-matrices for BX-version:
[Bp0, Bpp0] = makeB(baseMVA,bus,branch,'FDBX');
% Reduce B-matrices:
Bp = Bp0([pv; pq],[pv; pq]);
Bpp = Bpp0(pq,pq);

%% factor B matrices
[Lp, Up, Pp, Qp] = lu(Bp);
[Lpp, Upp, Ppp, Qpp] = lu(Bpp);

%% Calculate PTDFs & LODFs:
timeLODFsStart = tic;
PTDFs = makePTDF(baseMVA,bus,branch);
LODFs = makeLODF(branch,PTDFs);
timeLODFs = toc(timeLODFsStart);

%% Initializing:
% Define power flow matrices:
Pflow = branch(:,PF);
PflowThermal = branch(:,RATE_B) * 0.95; % Add 5% margin due to DC.
PflowNew = zeros(nl,nl);
```

```
PflowViolation = zeros(nl,3);
% Define voltage matrices:
voltageViolationLow = zeros(nb,3);
voltageViolationHigh = zeros(nb,3);
resVm = ones(nb,nl);
resVa = zeros(nb,nl);
% List of iterations count:
iterations = zeros(nb,1);
count = 0; % Number of cases tested.

%% Start the contingency analysis:
for cont = 1:nl
%% If isolated node:
if ismember(cont,indexIsland)
    resVm(:,cont) = NaN;
    resVa(:,cont) = NaN;
    PflowNew(:,cont) = NaN;
    iterations(cont) = NaN;
    continue; % Contiune for loop with x = x + 1.
end

%% Use pre-contingency values:
Va = Va0;
Vm = Vm0;

% Branch nodes:
fBus = branch(cont,F_BUS);
tBus = branch(cont,T_BUS);

%% Calculate compensation terms:
M = sparse([fBus tBus],1,[1 -1],nb,1);
Mp = M([pv;pq]);
Mq = M(pq);

% If parallel lines:
if ismember(cont,indexParallel)
    dyp = Bp0(fBus,tBus) / 2;
    dyq = Bpp0(fBus,tBus) / 2;
else
    dyp = Bp0(fBus,tBus);
    dyq = Bpp0(fBus,tBus);
end

invBp_Mp = Qp * (Up \(Lp \(Pp * Mp ))); % Pre-calculate Mp*H^-1.
invBpp_Mq = Qpp * (Upp \(Lpp \(Ppp *Mq )));
zp = Mp.' * invBp_Mp; %% M^t*H^-1*M
zq = Mq.' * invBpp_Mq; %% M^t*H^-1*M
cp = 1/(1/dyp + zp);
cq = 1/(1/dyq + zq);

%% Update Ybus:
Ybus2 = Ybus + M*-Ys(cont)*M';

%% Initial mismatch
mis = (V0 .* conj(Ybus2*V0) - Sbus) ./ Vm;
P = real(mis([pv;pq]));
Q = imag(mis(pq));

%% Iniatilizing:
converged = 0;
```

```
y = 0;

%% Run FDLF with compensation:
while (~converged && y < max_it)
y = y + 1;

%% ----- do P iteration, update Va -----
dVa = - Qp * (Up \ (Lp \ (Pp * P)));
dVaComp = cp * invBp_Mp * Mp.' * dVa;
Va([pv;pq]) = Va([pv;pq]) + dVa - dVaComp;

%% ----- do Q iteration, update Vm -----
dVm = - Qpp * (Upp \ (Lpp \ (Ppp * Q)));
dVmComp = cq * invBpp_Mq * Mq.' * dVm;
Vm(pq) = Vm(pq) + dVm - dVmComp;
V = Vm .* exp(1j * Va);

%% Evalute mismatches:
mis = (V .*conj(Ybus2 * V) - Sbus) ./ Vm;
P = real(mis([pv;pq]));
Q = imag(mis(pq));

%% check tolerance
normP = norm(P, inf);
normQ = norm(Q, inf);
if normP <= tol && normQ <= tol
    converged = 1;
    resVm(:,cont) = abs(V(:,1));
    resVa(:,cont) = Va(:,1);
    iterations(cont) = y;
    break;
end
if y == max_it
    iterations(cont) = max_it;
    break;
end
end

%% Use LODFs to estimate new active power flow:
PflowNew(:,cont) = Pflow + LODFs(:,cont) .* Pflow(cont);

%% Find the maximal flow overshooting:
% Find the highest violation.
[F1,I1] = min(PflowThermal - abs(PflowNew(:,cont)));
if F1 < 0 % Any flow-violations?
    PflowViolation(cont,1) = cont; % The contingency which cause it.
    % What line has the biggest overshooting (absolute value):
    PflowViolation(cont,2) = I1;
    % Find the MW-value:
    PflowViolation(cont,3) = abs(F1);
end

%% Find the minimal voltage-magnitude:
[V2,I2] = min(resVm(:,cont));
if V2 < 0.90 % Violation of the limit?
   voltageViolationLow(cont,1) = cont; % The contingency which cause it.
   voltageViolationLow(cont,2) = I2; % What node has the least voltage?
   voltageViolationLow(cont,3) = V2; % What is the value?
end
```

```
%% Find the maximal voltage-magnitude:
[V3,I3] = max(resVm(:,cont));
if V3 > 1.10
    voltageViolationHigh(cont,1) = cont; % The contingency which cause it.
    voltageViolationHigh(cont,2) = I3; % What node has the largest voltage?
    voltageViolationHigh(cont,3) = V3; % What is the value?
end

count = count + 1
end
totalTime = toc;

%% Output section:

%}  %}
% Additional output. Not to be used for big systems.
resVm;
resVa = resVa * 180/pi;
PflowNew;

countIsland
countParallel
iterations

% Display the violations:
I4 = nonzeros(PflowViolation(:,1));
PflowViolation = PflowViolation(I4,:)
I5 = nonzeros(voltageViolationLow(:,1));
voltageViolationLow = voltageViolationLow(I5,:)
I6 = nonzeros(voltageViolationHigh(:,1));
voltageViolationHigh = voltageViolationHigh(I6,:)

timeLODFs
totalTime
whos LODFs

% Activate to test the accuracy of the code:
%{
lineOut = 28;
mpc.branch(lineOut,BR_STATUS) = 0;
[r2,s2] = runpf(mpc, mpopt);
flow = PflowNew(:,lineOut)
voltageMagnitude = resVm(:,lineOut)
voltageAngle = resVa(:,lineOut)
%}
```

---

bounding.m

This is the implementation of the EBM, a contingency screening algorithm. The algorithm will indentify the boundary for all outages and it will calculate the new flows and test for

overloads. PI is used to rank the contingencies. Many output matrices and vectors give detailed information about the processing of the algorithm.

```
%% ~*Efficient Bounding Method*~
clear
clc
format shortg
tic;
t0 = tic;

%% Bounding parameters:
start_steps = 0; % Number of steps from faulted branch. 0,1,2...
max_steps = 10; % Max steps allowed.
raiseLimit = 0; % Rise the minimum limit to reduce expansion of layers.

%% Define named indices into bus, gen, branch matrices:
defineNamedIndices;

%% System parameters:
casedata = case2383wp; % case24_ieee_rts   case118
mpc = loadcase(casedata);
[baseMVA, bus, gen, branch] = ...
    deal(mpc.baseMVA, mpc.bus, mpc.gen, mpc.branch);
[ref, pv, pq] = bustypes(bus, gen);

%{
%% New flows (testing accuracy):
mpopt = mpoption('model','DC');
[r1, s1] = runpf(mpc, mpopt);
%}

%% Add zero columns to branch for flows if needed:
if size(mpc.branch,2) < QT
mpc.branch = ...
    [mpc.branch zeros(size(mpc.branch, 1), QT-size(mpc.branch,2))];
end

%% Voltage angles:
Va0 = bus(:,VA) * (pi/180);

%% Build B matrices and phase shift injections:
[B, Bf, Pbusinj, Pfinj] = makeBdc(baseMVA, bus, branch);

%% Compute complex bus power injections (generation - load):
% Adjusted for phase shifters and real shunts.
Pbus = real(makeSbus(baseMVA, bus, gen)) - Pbusinj - bus(:, GS) / baseMVA;

%% "Run" the power flow:
Va = Va0;

%% Update angles for non-reference buses:
Va([pv;pq]) = ...
    B([pv;pq],[pv;pq]) \ (Pbus([pv;pq]) - B([pv;pq],ref) * Va0(ref));

%% Update data matrices with solution:
```

```
branch(:, [QF, QT]) = zeros(size(branch, 1), 2);
branch(:, PF) = (Bf * Va + Pfinj) * baseMVA;
branch(:, PT) = -branch(:, PF);
bus(:, VM) = ones(size(bus, 1), 1);
bus(:, VA) = Va * (180/pi);

%% Set x-values:
nl = size(branch,1);
nb = size(bus,1);

%% Find isolated nodes and parallel lines:
IslandAndParallel;

%% Reactance values:
stat = branch(:,BR_STATUS); %% ones at in-service branches
indices = find(stat == 0);
x = branch(:,BR_X); %% Branch reactances.
x(indices) = NaN;

%% Make adjacent matrix:
bf = branch(:,F_BUS);
bt = branch(:,T_BUS);
adj = sparse(bf, bt, 1, nb, nb);
adj_mat = adj + adj' + speye(nb);

%% LUPQ factorization of B:
[L, U, P, Q] = lu(B([pv;pq],[pv;pq]));

%% Thermal limits:
f0 = branch(:,PF) ./ baseMVA; % f_pq^0
thermal = (branch(:,RATE_B) ./ baseMVA) .* 0.95; % Thermal limits.
if branch(:,RATE_B) == 0 % If no thermal data present.
    thermalMax = max(abs(f0)) .* 1.3; % 1.3 x the largest flow.
    thermal(:,1) = thermalMax;
    indices = abs(f0) < (100 / baseMVA);
    % Set capacity to 250 for lines < 100 :
    thermal(indices) = 250 / baseMVA;
    thermal = thermal .* 0.95; % 5% margin due to DC.
    dfmax = thermal;
else
    dfmax = (thermal - abs(f0));
end

xdfmax = zeros(nl,2);
xdfmax(:,1) = 1:1:nl;
xdfmax(:,2) = x .* dfmax; % delta f_pq^max*x_pq
sort_xdfmax = sortrows(xdfmax,2);
maxAngle = thermal .* x; % Define max angle-spread for all branches.

%% Find negative values:
negative_xdfmax_indices = find(xdfmax(:,2) < 0);
nNeg = length(negative_xdfmax_indices);

%% The limit is raised:
if nNeg > 0 || raiseLimit > 0
    limit = sort_xdfmax(nNeg+raiseLimit,2);
else
    limit = sort_xdfmax(1,2);
end
```

```
%% Branches which must be checked for violation when limit is raised:
if nNeg == 0
    branchCheck = sort_xdfmax(1:1:raiseLimit,1);
else
    branchCheck = sort_xdfmax(1:1:(nNeg+raiseLimit),1);
end

%% Nodes which must be calculated:
bf = branch(:,F_BUS);
bt = branch(:,T_BUS);
branchCheckNodes = union(bf(branchCheck),bt(branchCheck));

%% Calculate all inverse columns with consecutive indexing:
timeX = tic;
X = zeros(nb,nb);
for i = 1:nb
    one_node = sparse(i,1,1,nb,1);
    X([pv;pq],i) = Q * (U \ (L \ (P * one_node([pv;pq]))));
end
timeX = toc(timeX);

%% Initializing:
nodes_visited = zeros(nb,nl); % All nodes visited for each contingency.
numbers_of_nodes_visited = zeros(1,nl);
angle_spread_log = zeros(max_steps+1,nl); % First row is for n = 0!
success = ones(1,nl); % Keeps track of the success of the boundary method.
count = 0; % Keeps track of contingencies calculated.

%% Calculate node angle change for lines influenced:
nodeAngleChange = zeros(nb,1); % Change of voltage-angels.
violation = zeros(nl,nl); % What branches violates flow-limits?
PI = zeros(nl,1); % Performance Indices for contingency ranking.

%% Start the contingency analysis.
for cont = 1:nl
%% If isolated node:
if ismember(cont,indexIsland)
    angle_spread_log(:,cont) = NaN;
    nodes_visited(:,cont) = NaN;
    numbers_of_nodes_visited(1,cont) = NaN;
    success(1,cont) = NaN;
    violation(:,cont) = NaN;
    continue; % cont = cont + 1.
end

delta = zeros(nb,1); % Initialize vector of sensitivity factors.

branch_nodes = zeros(2,1);
branch_nodes(1) = bf(cont);
branch_nodes(2) = bt(cont);
node_vec = sparse(branch_nodes,1,1,nb,1);
node_vec_copy_1 = node_vec;

%% Number of start-steps from outaged branch for all contingencies.
n = start_steps; % n keeps track of the steps for rest of the code.
for i = 1:n
    node_vec = adj_mat * node_vec;
    if n == 1
        node_vec_center = node_vec_copy_1;
```

```
    end
    if n > 1 && i == n-1
        node_vec_center = node_vec;
    end
end

all_nodes = bus(node_vec > 0); % All nodes n start_steps from fault branch.
all_nodes_copy = all_nodes; % Needed for second while loop.
if n == 0
boundary_nodes = bus(node_vec > 0);
center_nodes = boundary_nodes;
end
if n > 0
    center_nodes = bus(node_vec_center > 0);
    boundary_nodes = setdiff(all_nodes,center_nodes);
end
node_vec_copy_2 = node_vec; % Needed for the second while loop.

%% The two inverse X columns needed for each contingency:
if bf(cont) ~= ref
X_bf = X(:,bf(cont));
end
if bt(cont) ~= ref
X_bt = X(:,bt(cont));
end

%% Traverse in order to check the angle-spread criterion:
while length(center_nodes) < nb && n <= max_steps

if n == start_steps
    nodes_visited((all_nodes_copy),cont) = all_nodes_copy(:);
    numbers_of_nodes_visited(1,cont) = length(find(nodes_visited(:,cont)));
else
    nodes_visited(all_nodes,cont) = all_nodes(:);
    numbers_of_nodes_visited(1,cont) = length(find(nodes_visited(:,cont)));
end

%% Calculate angle change:
% Calculate sensitivity factors of the center nodes (inside boundary):
if n == start_steps
    cn = center_nodes; % Shorter name.
    for i = 1:nnz(cn)
        if cn(i) == ref
        delta(cn(i)) = 0;
        elseif bt(cont) == ref
        delta(cn(i)) = (X_bf(cn(i))*x(cont))/(x(cont)-X_bf(bf(cont)));
        elseif  bf(cont) == ref
        delta(cn(i)) = -(X_bt(cn(i))*x(cont))/(x(cont)-X_bt(bt(cont)));
        else
        delta(cn(i)) = (X_bf(cn(i))-X_bt(cn(i)))*x(cont)/ ...
        (x(cont)-(X_bf(bf(cont))+X_bt(bt(cont))-2*X_bt(bf(cont))));
        end
    end
end

% Calculate sensitivity factors for the nodes due to the "raiseLimit":
if n == start_steps
    bc = branchCheckNodes; % Shorter name.
    for i = 1:nnz(bc)
        if bc(i) == ref
```

```
        delta(bc(i)) = 0;
        elseif bt(cont) == ref
        delta(bc(i)) = (X_bf(bc(i))*x(cont))/(x(cont)-X_bf(bf(cont)));
        elseif  bf(cont) == ref
        delta(bc(i)) = -(X_bt(bc(i))*x(cont))/(x(cont)-X_bt(bt(cont)));
        else
        delta(bc(i)) = (X_bf(bc(i))-X_bt(bc(i)))*x(cont)/ ...
        (x(cont)-(X_bf(bf(cont))+X_bt(bt(cont))-2*X_bt(bf(cont))));
        end
    end
end

% Calculate sensitivity factors for the boundary nodes:
nBn = nnz(boundary_nodes); % Number of boundary-nodes.
bn = boundary_nodes; % Shorter name of variable.
for i = 1:nBn
    if bn(i) == ref
    delta(bn(i)) = 0;
    elseif bt(cont) == ref
    delta(bn(i)) = (X_bf(bn(i))*x(cont))/(x(cont)-X_bf(bf(cont)));
    elseif bf(cont) == ref
    delta(bn(i)) = -(X_bt(bn(i))*x(cont))/(x(cont)-X_bt(bt(cont)));
    else
    delta(bn(i)) = (X_bf(bn(i))-X_bt(bn(i)))*x(cont)/ ...
    (x(cont)-(X_bf(bf(cont))+X_bt(bt(cont))-2*X_bt(bf(cont))));
    end
end

% Max change for boundary nodes' sensitivity factors:
delta_bn = delta(bn);
delta_max = max(delta_bn);
delta_min = min(delta_bn);

% Max change angular spread between the boundary nodes:
angleSpread = (delta_max-delta_min) * abs(f0(cont));
if angleSpread > 2.0 % Extreme cases, might indicate "islanding" of node.
    angleSpread = NaN;
    success(1,cont) = 0;
end
angle_spread_log(n+1,cont) = angleSpread;

% Check the criterion:
if angleSpread <= limit
    break;
elseif length(all_nodes) == nb % angleSpread <= limit not possible.
    success(cont) = 0;
    break;
elseif n == max_steps % Criterion not fulfiled for max_steps.
    success(cont) = 0;
    break;
else

%% Expand boundary further:
n = n + 1; % Number of steps from fault branch.

if n == start_steps + 1
node_vec = node_vec_copy_2;
node_vec_center = node_vec_copy_2;
else
node_vec_center = node_vec;
```

```
end

node_vec = adj_mat * node_vec;

all_nodes = bus(node_vec > 0); % All nodes k start_steps from fault branch.
center_nodes = bus(node_vec_center > 0);
boundary_nodes = setdiff(all_nodes,center_nodes);
end
end

%% Calculate the new flows and PI:
nodeAngleChange = f0(cont) .* delta;
newAngle = Va + nodeAngleChange;
angleSpread = newAngle(bf) - newAngle(bt);

% Find overloaded branches:
findViolation = find(abs(angleSpread) > maxAngle);
violation(findViolation,cont) = findViolation;
for k = 1:nnz(findViolation)

    if findViolation(k) == cont % Are out of service.
        continue;
    end

    tempPI = ...
        (angleSpread(findViolation(k)) / maxAngle(findViolation(k)))^2 ;
    PI(cont) = PI(cont) + tempPI;

    %if PI(cont) > 10^5 % If huge overload.
    %PI(cont) = 10^5; % Max PI value.
    %break;
    %end

end

count = count + 1
end

%% Find the highest PIs and sort them:
ind = find(PI);
ni = length(ind);
result = zeros(ni,2);
for a = 1:ni
    result(a,1) = ind(a);
    result(a,2) = PI(ind(a));
end
resultPI = sortrows(result,-2);
time = toc;

%% Success & total time consumption:
not_success = find(success == 0);
totalTime = toc(t0);

%% Output:
negative_xdfmax_indices
limit
numbers_of_nodes_visited
not_success
resultPI
```

```
timeX
totalTime
whos X

%{
% New flow for testing accuracy & additional output:
branchOut = 3;
mpc.branch(branchOut,BR_STATUS) = 0;
[r2,s2] = runpf(mpc, mpopt);

nodeAngleChange = nodeAngleChange .* 180/pi
angleUpdated = r1.bus(:,VA) + nodeAngleChange
violation_test = nonzeros(violation(:,branchOut))
numbers_violation_branches = nnz(violation_test)
nodes_visited_test = nonzeros(nodes_visited(:,branchOut))
angle_spread_log_test = angle_spread_log(:,branchOut)
%}
```
