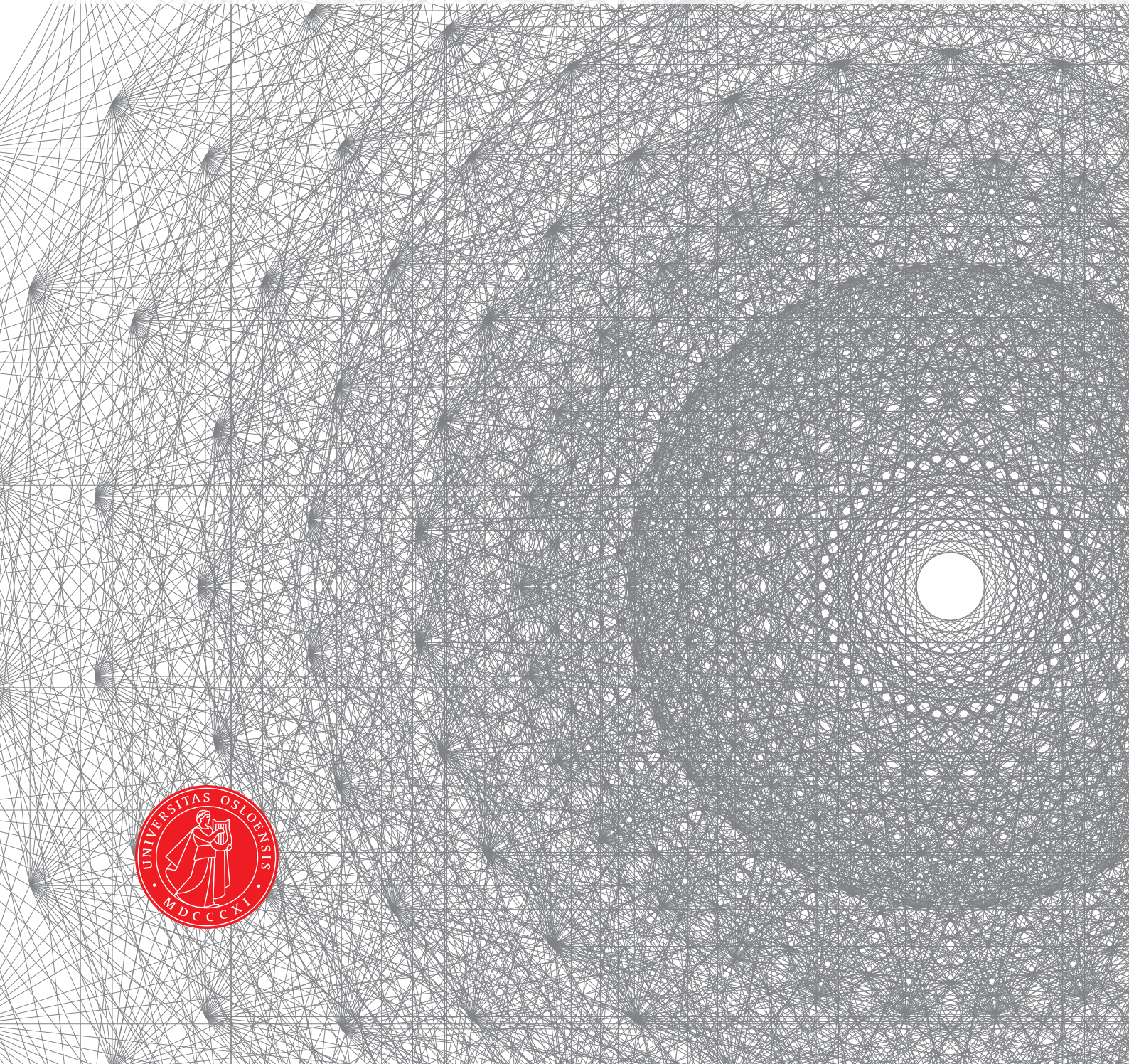
UiO : Department of Mathematics
University of Oslo

# Comparison of two lower bounds for availabilities in multistate monotone systems -an established one and a new one

**Tobias Abrahamsen**

Master's Thesis, Autumn 2017



This master's thesis is submitted under the master's programme *Modelling and Data Analysis*, with programme option *Finance, Insurance and Risk*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group $E_8$, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

**Abstract**

The lower bounds are used to determine availabilities of a system, and are used in many different companies.
Such is for example a company that supplies water to a city. The company would most likely want to know how reliable the water system is. Does the water system deliver minimum 40% or more of its water capacity to the city? It is important to have a lower bound that is close to the real value.
I will therefore in this paper show, and explain, what a Multistate Monotone System (MMS) is, and compare two lower bounds. The bounds are a new lower bound and an established lower bound. I will also present the theorems linked to the bounds, and establish some points to make computer implementation easier. I am also going to show something that I have figured out, a new theorem, and an algorithm which makes it easier to identify minimal cut vectors.
Part of the computer code will be discussed and explained, where some of the algorithms will be shown as Pseudocode. The computer simulation gives data that will be shown in plots and used to compare the two bounds. The comparison of the two bounds shows that the new bound is better than the established bound.

**Acknowledgements**

I would like to thank my supervisor Arne Bang Huseby for all the times I have been to his office to get help and to discuss the programming. I have also appreciated all the nice conversations we have had. I would also like to thank Bent Natvig for the support, proofreading and for all the help with the theoretical part. Thanks to Jørund Gåsemyr for deep questions that made me think about parts of the new algorithm that I have figured out.

I would like to thank my partner, family and friends because they have been there and given support when I have needed it. I would also like to give an extra thanks to my partner, who has let me work far out and in the early morning hours the times I have had to finish up my thoughts. A big thank you to my fellow students, especially Jonas Christensen for all the deep conversations and the friendships we have got over the years at the University of Oslo.

# Contents

# 1 Notation

## 1.1 Introduction

In multistate reliability theory we consider *Multistate Monotone Systems* (MMS). An MMS (C,$\phi$) consists of the *component set* $C = \{1, 2, ..., n\}$ and a *structure function* $\phi$. The *states* of component $i$ is $S_i = \{0, 1, ..., M_i\}$ for all $i \in C$, while the states of the system is $S = \{0, 1, ..., M\}$. Here $n$ and $M$ are positive integers. The state of component $i$ at time t is denoted by $x_i(t)$, and it belongs to a subset $S_i$ of $S$, which is assumed in Natvig (2011) [5] to contain 0 and M. We will allow $M_i = \max(S_i)$ to be smaller than the M. If we let $M = M_i = 1$, all the results will cover the binary case.
The system state is supposed to be a non-decreasing function of the component states, and is given by $\phi(X(t))$, where $X(t) = (X_1(t), ..., X_n(t))$. Let us assume that $\phi(0, ..., 0) = 0$ and $\phi(M_1, ..., M_n) = M$. Let us consider the time points t in some subset $\tau(I)$ of an interval $I$ of interest, with $\tau(I)$ being finite and $\tau(I) = I$ to being typical special cases. $I$ will typically be chosen for operational considerations.
An MMS is a generalization of the concept of a *Binary Monotone System* (BMS). This will give us a more refined description of a system than a BMS. This can often be necessary in the cases where we need to handle more complex systems that can perform at different levels. The systems could e.g. be a water transportation system, or electrical grid, where the probabilities for the system performing above certain given levels at any given time interval $I$ are of interest. In some applications there may be natural to let $S$ and $S_i$ consist of arbitrary real numbers. In general, the elements of $S$ and $S_i$ will be representing an ordering of meaningful performance levels.
The following is needed .

$$(\cdot_i, \mathbf{x}) = (x_1, ..., x_{i-1}, \cdot, x_{i+1}, ..., x_n).$$
$$\mathbf{y} < \mathbf{x} \text{ means } y_i \leq x_i \text{ for } i = 1, ..., n, \text{ and } y_i < x_i \text{ for some } i.$$

We will also need the following definitions from Natvig(2011) [5].

**Definition 1.1.** *Let $\phi$ be the structure function of a BMS. A vector $\boldsymbol{x}$ is said to be a path vector iff $\phi(\boldsymbol{x}) = 1$. The corresponding path set is $C_1(\boldsymbol{x})$. A minimal path vector is a path vector $\boldsymbol{x}$ such that $\phi(\boldsymbol{y}) = 0$ for all $\boldsymbol{y} < \boldsymbol{x}$ The corresponding minimal path is $C_1(\boldsymbol{x})$.*

**Definition 1.2.** *Let $\phi$ be the structure function of an MMS and let $j \in \{1, ..., M\}$. A vector $\boldsymbol{x}$ is said to be a path vector to level $j$ iff $\phi(\boldsymbol{x}) \geq j$. The corresponding path set and binary type path set are respectively given by*

$$C_\phi^j(\boldsymbol{x}) = \{i | x_i \geq 1\} \text{ and } C_{\phi\ BT}^j(\boldsymbol{x}) = \{i | x_i \geq j\}$$

*A minimal path vector to level $j$ is a path vector $\boldsymbol{x}$ such that $\phi(\boldsymbol{y}) < j$ for all $\boldsymbol{y} < \boldsymbol{x}$. The corresponding path set and binary type path set are also said to be minimal.*

**Definition 1.3.** *Let $\phi$ be the structure function of a BMS. A vector $\boldsymbol{x}$ is said to be a cut vector iff $\phi(\boldsymbol{x}) = 0$. The corresponding cut set is $C_0(\boldsymbol{x})$. A minimal cut vector is a cut vector $\boldsymbol{x}$ such that $\phi(\boldsymbol{y}) = 1$ for all $\boldsymbol{y} > \boldsymbol{x}$ The corresponding minimal cut is $C_0(\boldsymbol{x})$.*

**Definition 1.4.** *Let $\phi$ be the structure function of an MMS and let $j \in \{1, ..., M\}$. A vector $\boldsymbol{x}$ is said to be a cut vector to level $j$ iff $\phi(\boldsymbol{x}) < j$. The corresponding cut set and binary type cut set are respectively given by*

$$D^j_\phi(\boldsymbol{x}) = \{i | x_i < M\} \text{ and } D^j_{\phi\ BT}(\boldsymbol{x}) = \{i | x_i < j\}$$

*A minimal cut vector to level $j$ is a cut vector $\boldsymbol{x}$ such that $\phi(\boldsymbol{y}) \geq j$ for all $\boldsymbol{y} > \boldsymbol{x}$. The corresponding cut set and binary type cut set are also said to be minimal.*

Let us illustrate this:

Let C $= \{1, 2\}, S_i = S = \{0, 1, 2, 3\}$ and
$\phi(\mathbf{x}) = \max(x_1, x_2)$
Then this system consists of two components that are in a parallel system.

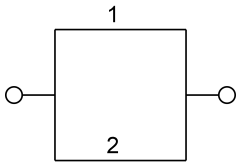



**Figure 1:** Example of a parallel system

The performance processes to the components $\{X_i(t), t \in \tau(I)\},\ i = 1, 2,$ are random, possibly stochastically dependent processes that are involving repair at fixed or random points of time. The processes are assumed to be continuous from the right.

Throughout this paper we will consider methods based on the component *availabilities* and *unavailabilities*.

$p_i^j = P(X_i(t) \geq j \text{ for all } t \in \tau(I)) = P(\min_{t \in \tau(I)} X_i(t) \geq j),$
$i = 1, ..., n, j = 0, ..., M$

$q_i^j = P(X_i(t) < j \text{ for all } t \in \tau(I)) = P(\min_{t \in \tau(I)} X_i(t) < j),$
$i = 1, ..., n, j = 0, ..., M$

Here, $1 = p_i^0 \geq p_i^1 \geq \cdots \geq p_i^{M_i} \geq 0$, and $p_i^j = p_i^{j-1}$ if $j - 1 \notin S_i$, $p_i^j = 0$ if $M_i < j$, $j = \{0, ..., M\}$. We denote by $\mathbf{P}_\phi\{p_i^j\}_{\substack{i=1,..,n \\ j=1,..,M}}$ the vector

consisting of all the availabilities for all the components. The determination of $p_i^j$ can be based on expert opinions, test data or operational data, or on a combination of these information sources.

The components' availabilities over an interval I do not determine the corresponding system availabilities

$$p_\phi^j = P(\phi(\mathbf{X}(t)) \geq j \text{ for all } t \in \tau(I)), j = 1, ..., M$$

The components' unavailabilities over an interval I do not determine the corresponding system unavailabilities

$q_\phi^j = P(\phi(\mathbf{X}(t)) < j \text{ for all } t \in \tau(I)), j = 1, ..., M$
$\mathbf{Q}_\phi = \{q_i^j\}_{\substack{i=1,..,n \\ j=1,..,M}}$

These system availabilities can not be calculated, even in the case of independent components, and we have to resort to bounds.

Basic bounds are based on the sets of minimal path vectors and minimal cut vectors to level j.

### 1.2 Flow network

An MMS$(C, \phi)$ is said to be a *generalized flow network* (GF-system), if $\phi$ can be written in the following way:

$$\phi = \phi(\mathbf{X}) = \min_{1 \leq k \leq m_\phi} \sum_{i \in K_k} x_i,$$

where $K_1, ..., K_{m_\phi}$ are non-empty subsets of $C$, the set of components. Let us assume that the subsets $K_1, ..., K_{m_\phi}$ are distinct incomparable sets. That is, no set is contained in any of the other sets, i.e. $K_i \not\subseteq K_l$ for all $i \neq l$. The sets $K_1, ..., K_{m_\phi}$ are referred to as the *minimal flow cut sets* of the system. Note that if $K_1, ..., K_{m_\phi}$ are the minimal flow cut sets in a directed two-terminal flow network with edge set $C$, and $\phi$ denotes the maximal flow that can be sent through the network from one terminal to another, then $(C, \phi)$ is called a flow network. A generalized flow network is a more general concept since we do not require that the system can be represented as a directed two-terminal flow network.
As an example think that we will send water through 2 different systems. One of them is an directed network and the other one is an undirected network. If you look at the figures 1 and 2 you can see that the directed network got directions on which way the component can transport water. The undirected networks' components can transport water in any directions.

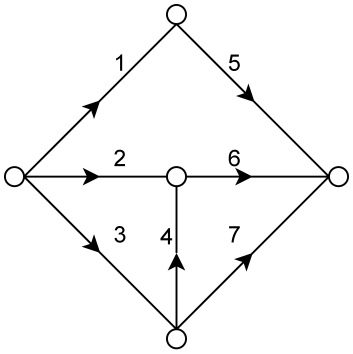



**Figure 2:** Example of an directed network

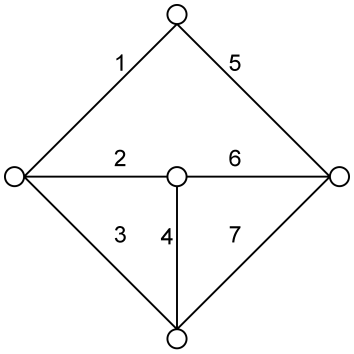



**Figure 3:** Example of an undirected network

So here we will get the following minimal cut sets for a directed network:
$\{1,2,3\},\ \{1,2,4,7\},\ \{1,6,3\},\ \{1,6,7\},\ \{5,2,3\},\ \{5,2,4,7\},\ \{5,6,3\},$
$\{5,6,7\}$

For the undirected network we will get the following minimal cut sets:
$\{1,2,3\},\ \{1,2,4,7\},\ \{1,4,6,3\},\ \{1,6,7\},\ \{5,2,3\},\ \{5,2,4,7\},$
$\{5,4,6,3\},\ \{5,6,7\}$

The minimal cut sets are almost identical, but there are 2 large differences: In the directed network where you have the minimal cut sets $\{1,6,3\}$ and $\{5,6,3\}$ you must in the undirected system add the component 4. This is done because there are no direction limits in the undirected system. So it is important to use the right type of system for analysis. If we make the mistake to use an undirected network to analyze a system that is a directed network, an analysis will in this case predict a greater chance of performing at a higher state than what the system actually does. Throughout this thesis we will only consider directed network systems.

### 1.3 Established lower bounds

We are going to go through bounds from Natvig (2011) [5]. Definition 1.2 and 1.4 are used here. We will also need the following definition:

For $p_i \in [0,1]$ we have $\coprod_{i=1}^{n} p_i = 1 - \prod_{i=1}^{n}(1-p_i)$

With these definitions in our toolbox, we can now look at theorem 2.27 and theorem 2.28 from Natvig(2011) [5].

**Theorem 2.27 Natvig(2011) [5]**
Let $(C, \phi)$ be an MMS. Furthermore, for $j \in \{1, \ldots, M\}$ let $\mathbf{y}_k^j = (y_{1k}^j, \ldots, y_{nk}^j), k = 1 \ldots, n_\phi^j$ be its minimal path vectors to level $j$ and $\mathbf{z}_k^j = (z_{1k}^j, \ldots, z_{nk}^j), k = 1 \ldots, m_\phi^j$ be its minimal cut vectors to level $j$ and

$$C_\phi^j(\mathbf{y}_k^j), k = 1 \ldots, n_\phi^j \text{ and } D_\phi^j(\mathbf{z}_k^j), k = 1 \ldots, m_\phi^j$$

the corresponding minimal path and cut sets to level $j$. Let

$$\ell_\phi^{''j} = \max_{1 \le k \le n_\phi^j} P\big[\cap_{i \in C_\phi^j(\mathbf{y}_k^j)} (X_i \ge y_{ik}^j)\big] \qquad \bar{\ell}_\phi^{''j} = \max_{1 \le k \le m_\phi^j} P\big[\cap_{i \in D_\phi^j(\mathbf{z}_k^j)} (X_i \le z_{ik}^j)\big]$$

Then

$$\ell_\phi^{''j} \le p_\phi^j \le 1 - \bar{\ell}_\phi^{''j}$$

Furthermore, let

$$\ell_\phi^{'j}(\mathbf{P}_\phi) = \max_{1 \le k \le n_\phi^j} \prod_{i \in C_\phi^j(\mathbf{y}_k^j)} p_i^{y_{ik}^j} \qquad \bar{\ell}_\phi^{'j}(\mathbf{Q}_\phi) = \max_{1 \le k \le m_\phi^j} \prod_{i \in D_\phi^j(\mathbf{z}_k^j)} q_i^{z_{ik}^j + 1}$$

If $X_1, ..., X_n$ are associated, then

$$\ell_\phi^{'j}(\mathbf{P}_\phi) \le p_\phi^j \le 1 - \bar{\ell}_\phi^{'j}(\mathbf{Q}_\phi)$$

**Theorem 2.28 Natvig(2011) [5]**
Let $(C,\phi)$ be an MMS where $X_1,\ldots,X_n$ are associated. Furthermore, for $j\in\{1,\ldots,M\}$ let $\mathbf{y}_k^j=(y_{1k}^j,\ldots,y_{nk}^j), k=1\ldots,n_\phi^j$ be its minimal path vectors to level $j$ and $\mathbf{z}_k^j=(z_{1k}^j,\ldots,z_{nk}^j), k=1\ldots,m_\phi^j$ be its minimal cut vectors to level $j$ and

$$C_\phi^j(\mathbf{y}_k^j), k=1\ldots,n_\phi^j \text{ and } D_\phi^j(\mathbf{z}_k^j), k=1\ldots,m_\phi^j$$

the corresponding minimal path and cut sets to level $j$. Let

$$\ell_\phi^{*j}=\prod_{k=1}^{m_\phi^j} P\big(\cup_{i\in D_\phi^j(\mathbf{z}_k^j)}(X_i>z_{ik}^j)\big) \qquad \bar{\ell}_\phi^{*j}=\prod_{k=1}^{n_\phi^j} P\big(\cup_{i\in C_\phi^j(\mathbf{y}_k^j)}(X_i<y_{ik}^j)\big)$$

Then

$$\ell_\phi^{*j}\le p_\phi^j\le 1-\bar{\ell}_\phi^{*j}$$

Furthermore, let

$$\ell_\phi^{**j}(\mathbf{P}_\phi)=\prod_{k=1}^{m_\phi^j}\coprod_{i\in D_\phi^j(\mathbf{z}_k^j)} p_i^{z_{ik}^j+1} \qquad \bar{\ell}_\phi^{**j}(\mathbf{Q}_\phi)=\prod_{k=1}^{n_\phi^j}\prod_{i\in C_\phi^j(\mathbf{y}_k^j)} q_i^{y_{ik}^j}$$

If $X_1,...,X_n$ are independent, then

$$\ell_\phi^{*j}=\ell_\phi^{**j}(\mathbf{P}_\phi)\le p_\phi^j\le 1-\bar{\ell}_\phi^{**j}(\mathbf{Q}_\phi)=1-\bar{\ell}_\phi^{*j}$$

We will in the chapter "Early assumption and attempts" go through these two theorem and make some practical changes which makes it easier for us to find the following lower bound:

$$B_\phi^{*j}(\mathbf{P}_\phi)=\max_{j'\ge j}\max(\ell_\phi^{**j'}(\mathbf{P}_\phi),\ell_\phi'^{j'}(\mathbf{P}_\phi))$$

## 1.4 New lower bounds

We will now go through the new lower bound given in the paper "Improved availability bounds for binary and multistate monotone systems with independent component processes"[4].

Let us first look at a special case where the interval $I$ collapses to a point, $I = [t, t]$.Let us also assume that the states at $t$ are independent, and that the system availabilities are deterministic functions $h_\phi^j(\mathbf{P}_\phi)$ of the component availabilities. To see this, define for $i = 1, \dots n,\ k \in S_i$ (with $p_i^{M_i+1} = 0$),

$$r_i^k = P(X_i(t) = k) = p_i^k - p_i^{k+1}$$

collected in the vector $\mathbf{r}$. Then by the independence of $X_i(t),\ i = 1, \dots, n$

$$P(\mathbf{X}(t) = x) = \prod_{i=1}^{n} r_i^{x_i}$$

for each $j = 1, \dots, M$. We may then write

$$p_\phi^j = E(I(\phi(\mathbf{X}(t)) \geq j)|\mathbf{r}) = \sum_{\mathbf{x} \in S_1 \times \cdots \times S_n} I(\phi(\mathbf{x}) \geq j) \prod_{i=1}^{n} r_i^{x_i} =$$

$$\sum_{\mathbf{x} \in S_1 \times \cdots \times S_n} I(\phi(\mathbf{x}) \geq j) \prod_{i=1}^{n} (p_i^{x_i} - p_i^{x_i+1}) \stackrel{\text{def}}{=} h_\phi^j(\mathbf{P}_\phi)$$

If we can calculate the last equation numerically, then the need of a lower bound is eliminated in this special case.
Returning to the general interval $I$ , we can still evaluate the function $h_\phi^j(\mathbf{P}_\phi)$, even when $\mathbf{p}$ now represents the component availabilities in $I$. The idea is to use the number $h_\phi^j(\mathbf{P}_\phi)$ as a lower bound in the case of independent component processes.
In the following theorem it will be shown that a lower bound is in fact obtained that way.

**Theorem 1.5.** *Assume that the component processes are independent. Define*

$$\check{X}_i = \min_{t \in \tau(I)} X_i(t),\ i = 1, \dots, n$$

*and let* $\check{\boldsymbol{X}} = (\check{X}_1, \dots, \check{X}_n)$. *Let* $\boldsymbol{p}$ *be the vector with components*

$$p_i^k = P\Big(\check{X}_i \geq k,\ i = 1, \dots, n, \quad k \in S_i\Big)$$

*Define*

$$\tilde{l}_\phi^j(\boldsymbol{P}_\phi) = h_\phi^j(\boldsymbol{P}_\phi),\ j = 1, \dots, M.$$

*Then*

$$p_\phi^j \geq \tilde{l}_\phi^j(\boldsymbol{P}_\phi)$$

*If the component processes are independent, then*

$$\tilde{l}_\phi^j = h_\phi^j(\boldsymbol{P}_\phi)$$

## 1.5 Why lower bounds?

In this paper we are going to investigate if the new lower bound is better than the established lower bound. But a general question is: Why is the lower bound of general interest?
The lower bound would in many situations be a better approach to find system reliability than what an upper bound does. This is because we are interested in what the worst situation would be. We are more often interested in finding the worst case, than what the best case would be.

Let us look at an example of why we often like to use the lower bound.
Say that you own a company that supplies water to a city. You would then therefore most likely want to know how reliable the water system is. Does the water system deliver minimum 40% or more of its water capacity to the city? It will in this type of case be crucial that you use the lower bound and not the upper bound. Since the upper bound in this situation can give us a better picture of the situation than what it really is. We would therefore use the lower bound since it can give us the worst picture of the situation than what it really is. You can then find which part of the water system that needs improvement to fulfill wanted system reliability.

But this will not say that the upper bound is useless and never used. There are situations where you want to find the upper bound, e.g. if you have a power grid where you want the power capacity to be the highest possible without the transformers being overloaded. In this situation the upper bound gives the best picture of the situation and could therefore also be used.

# 2 Early assumptions and attempts

## 2.1 Some practical notational assumptions

When we later are going to simulate and calculate the new and the established lower bounds we need to make some assumptions.
First let us look at the established lower bounds. Since we are going to use the minimal path and cut sets the following parts of Theorem 2.27 and 2.28 from Natvig(2011) [5] are going to be used to calculate the lower bound for the established ones.

$$\text{(From theorem 2.27)}\ \ \ell_{\phi}^{'j}(\mathbf{P}_{\phi}) = \max_{1\leq k\leq n_{\phi}^{j}} \prod_{i\in C_{\phi}^{j}(\mathbf{y}_{k}^{j})} p_{i}^{y_{ik}^{j}}$$

$$\text{(From theorem 2.28)}\ \ \ell_{\phi}^{**j}(\mathbf{P}_{\phi}) = \prod_{k=1}^{m_{\phi}^{j}} \coprod_{i\in D_{\phi}^{j}(\mathbf{z}_{k}^{j})} p_{i}^{z_{ik}^{j}+1}$$

Here we will assume that $p_i^j = P[X_i \geq j]$ and for $p_i \in [0,1]$ $\coprod_{i=1}^{n} p_i = 1 - \prod_{i=1}^{n}(1-p_i)$. We can therefore write:

$$p_{i}^{y_{ik}^{j}} = P[X_i \geq y_{ik}^{j}]$$

and

$$p_{i}^{z_{ik}^{j}+1} = P[X_i \geq z_{ik}^{j} + 1].$$

To make 2.28 more easy to implement on a computer we write it as follows:

$$\text{(From theorem 2.28)}\ \ \ell_{\phi}^{**j}(\mathbf{P}_{\phi}) = \prod_{k=1}^{m_{\phi}^{j}} \Big(1 - \prod_{i\in D_{\phi}^{j}(\mathbf{z}_{k}^{j})} (1 - p_{i}^{z_{ik}^{j}+1})\Big)$$

Further let us assume that the component processes are independent. We can then assume that the newer bound is

$$\tilde{l}_{\phi}^{j} = h_{\phi}^{j}(\mathbf{p})$$

This gives us the opportunity to use the following simple but inefficient expression

$$h_{\phi}^{j}(\mathbf{p}) = \sum_{\mathbf{x}|\phi(\mathbf{x})\geq j} \prod_{i=1}^{n} r_{i}^{x_i}$$
$$\text{where } r_{i}^{x_i} = p_{i}^{x_i} - p_{i}^{x_i+1}$$

These changes will make the implementation go much easier on the computer.

## 2.2 Identifying minimal cut vectors

In the simulation later in this paper we need to identify the minimal path vectors and the minimal cut vectors in a flow network. The algorithm for doing this is shown on page 18 under the chapter "Find vector class". In order to show that this algorithm is correct we need some results about flow networks.
The first result provides a necessary condition for when a vector is a minimal cut vector. For a similar and essentially equivalent approach see Jørund Gåsemyr "Note on flow network systems" [3].

**Theorem 2.1.** *Assume that $\boldsymbol{x}$ is a minimal cut vector to level $j \leq M$. Then there exists a minimal flow cut set $K$ such that*

$$\sum_{i \in K} x_i = j - 1, \text{ and } x_i = M_i \text{ for all } i \notin K.$$

*Proof.* By the definition of $\phi$ we know that:

$$\phi(\boldsymbol{x}) = \min_{1 \leq j \leq m_\phi} \sum_{i \in K_j} x_i.$$

Hence, for any $\boldsymbol{x}$ there must exist a minimal flow cut set $K$ with total flow through $K$ equal to the system flow, that is:

$$\phi(\boldsymbol{x}) = \sum_{i \in K} x_i.$$

Assume that $\boldsymbol{x}$ is a minimal cut vector to level $j$, and let $K$ be chosen so that the total flow through $K$ is equal to the system flow. Then it follows that $\phi(\boldsymbol{x}) < j$. In fact, since all component states are integers, it follows that $\phi(\boldsymbol{x}) \leq j - 1$.

If $\phi(\boldsymbol{x}) < j - 1$, we may increase the state of one of the components in $K$ by 1, and still have $\phi \leq j - 1$. However, this contradicts that $\boldsymbol{x}$ is a minimal cut vector to level $j$. Thus, we conclude that $\phi(\boldsymbol{x}) = j - 1$.

If there exists a component $i \notin K$ such that $x_i < M_i$, we may replace $x_i$ by $M_i$ without changing the flow through $K$, and thus still have $\phi \leq j - 1$. However, this also contradicts that $\boldsymbol{x}$ is a minimal cut vector to level $j$.

Hence, we conclude that $\sum_{i \in K} x_i = \phi(\boldsymbol{x}) = j - 1$ and $x_i = M_i$ for all $i \notin K$, which completes the proof. □

Theorem 2.1 shows that in order to identify the minimal cut vectors of the system, we must look among vectors of the form $\boldsymbol{x}$ where for some minimal flow cut set $K$ we have:

$$\sum_{i \in K} x_i = j - 1, \text{ and } x_i = M_i \text{ for all } i \notin K.$$

Unfortunately, while all vectors of this form must be cut vectors to level $j$, not all such vectors are *minimal* cut vectors to this level. Thus, we need a way to determine whether or not a given cut vector $\boldsymbol{x}$ is minimal or not.

To determine if a vector is a minimal cut vector at level $j \leq M$ or not in a simple way we first need to sort the minimal flow cut sets. We sort them by number of elements:

$$\boldsymbol{K} = \{K_1, K_2, ...., K_{m_\phi}\}, \text{ where } |K_1| \leq |K_2| \leq ... \leq |K_{m_\phi}|$$

We then choose $K_r \in \boldsymbol{K}$. Based on this $K_r$ we choose an $\boldsymbol{x}$ satisfying the condition given in theorem 2.1. The next step is to determine if this is a minimal cut vector or not. There are 3 different scenarios:

**Scenario 1:** $\exists K_s \neq K_r$ such that $\sum_{i \in K_s} x_i < j - 1$. This implies that $\phi(\boldsymbol{x}) < j - 1$. We can then discard the current $\boldsymbol{x}$, since this implies that $\boldsymbol{x}$ obviously is not minimal at level $j$.

**Scenario 2:** $\forall K_s \neq K_r$ we have $\sum_{i \in K_s} x_i > j - 1$. Then obviously $\boldsymbol{x}$ is a minimal cut vector at level $j$. Hence, we can then keep the current $\boldsymbol{x}$.

**Scenario 3:** $\forall K_s \neq K_r$ we have $\sum_{i \in K_s} x_i \geq j - 1$, and $\exists K_s \neq K_r$ such that $\sum_{i \in K_s} x_i = j - 1$. For all such minimal flow cut sets we need to check if $\exists i \in K_r - K_s$ such that $x_i < M_i$. If this is the case, we can increase $\boldsymbol{x}$ by replacing $x_i$ by $M_i$ without increasing the value of $\phi$. Hence, $\boldsymbol{x}$ cannot be a minimal cut vector and must be discarded. Assume instead that the opposite is the case, i.e., $x_i = M_i$ for all $i \in K_r - K_s$, for all $K_s \in \boldsymbol{K}$ such that $\sum_{i \in K_s} x_i = j - 1$. If there are several minimal flow cut sets, $K_s$, with this property, we choose the one with the lowest index. In this case $\boldsymbol{x}$ cannot be increased without increasing $\phi$. Hence, $\boldsymbol{x}$ must be a minimal cut vector. However, we may have found the same $\boldsymbol{x}$ earlier in the process. If this is the case we will discard this $\boldsymbol{x}$ at this stage in the process in order to avoid duplicates. This happens if $s < r$. On the other hand if $s > r$, $\boldsymbol{x}$ has not been found earlier in the process, so in this case we can keep the current $\boldsymbol{x}$.

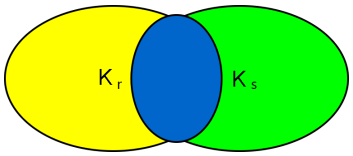



**Figure 4:** $K_r - K_s$ is the yellow part of the Venn diagram.

Note that in order to identify the scenarios described above in the general case we need to check all $K_s \in \boldsymbol{K}$ where $s \neq r$. However, if $M_1 = \cdots = M_n$, it is sufficient to check all $K_s \in \boldsymbol{K}$ where $s < r$. The reason for this is that we have sorted the minimal flow cut sets with respect to their sizes. As a result we must have $\sum_{i \in K_s} x_i \geq j - 1$ for any $s > r$. Thus, if $\sum_{i \in K_s} x_i < j - 1$ for some $s \neq r$, we must have $s < r$. That is, Scenario 1 can only occur for $K_s$ where $s < r$. Assume conversely that $\sum_{i \in K_s} x_i \geq j - 1$ for all $s < r$. Then it may happen that there exists an $s > r$ such that $\sum_{i \in K_s} x_i = j - 1$.

However, if this is the case, we must have $x_i = M_i$ for all $i \in K_r - K_s$ (since $|K_s| \geq |K_r|$). Hence, $\boldsymbol{x}$ will not be discarded because of this $K_s$.

Let us illustrate how we can identify minimal cut vectors.
Example.
We have minimal flow cut sets $K_1 = \{2, 4\}$ and $K_2 = \{1, 2, 3\}$ with states $j = \{0, 1, 2, 3\}$ and system state $\phi_j = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. We have the following sorting of the minimal flow cut sets $\boldsymbol{K} = \{K_1, K_2\}$. We can then find all the minimal cut sets for $\phi_j = 2$ by using theorem 2.1 and here we have the special case where all the components have the same states.

Let us look at the minimal flow cut $K_1$ first.
First we need to find the component states for the components in the current cut that fulfills:

$$\sum_{i \in K_1} x_i = 2$$

We can first of all set all of the components that aren't in the cut to their maximum states. $x_i = 3, \text{ for all } i \notin K_1$
We can now look at the components in the current minimal flow cut set. We see that $x_2 = 2$ and $x_4 = 0$ fulfills claims on $\sum_{i \in K_1} x_i = 2$. Since there are no minimal flow cut sets before $K_1$, we therefore have the following minimal cut vector:

$$\mathbf{x} = (3, 2, 3, 0)$$

The next $x_2, x_4$ that fulfills the claims is $x_2 = 1, x_4 = 1$. We have the following minimal cut vector:

$$\mathbf{x} = (3, 1, 3, 1)$$

And the last one for the current minimal flow cut set we have $x_2 = 0, x_4 = 2$. We then have the following minimal cut vector:

$$\mathbf{x} = (3, 0, 3, 2)$$

Lets now look at the next minimal cut set $K_2 = \{1, 2, 3\}$.

We now set $x_i = 3, \text{ for all } i \notin K_2$
First we need to find the component states for the components in the current cut that fulfills:

$$\sum_{i \in K_2} x_i = 2$$

We see that $x_1 = 2$, $x_2 = 0$ and $x_3 = 0$ fulfills claims on $\sum_{i \in K_2} x_i = 2$. We then check the minimal flow cut set which is before the current cut set in $\boldsymbol{K}$. $K_1$ is the only one and it does have a cut state larger than 2, we therefore have the following minimal cut vector:

$$\mathbf{x} = (2, 0, 0, 3)$$

$x_1 = 1, x_2 = 1, x_3 = 0$ is the next one. $K_1$ is larger than 2, we therefore have the following minimal cut vector

$$\mathbf{x} = (1, 1, 0, 3)$$

We can find the rest of the the minimal cut vectors for $\phi_j = 2$ by continuing. It is the same way for all the system states and all the minimal cut vectors for this system is:

$$\{0, 1, 1, 3\}, \{1, 0, 1, 3\}, \{1, 1, 0, 3\}, \{2, 0, 0, 3\}, \{0, 2, 0, 3\}, \{0, 0, 2, 3\},$$
$$\{3, 1, 3, 1\}, \{3, 2, 3, 0\}, \{3, 0, 3, 2\}$$

# 3 Computer code

## 3.1 Computer language and starting points

The main program that computes and *simulate* is written in Java The plots are made through R. The Java code is provided at the end of this paper in the section "Full computer code".
To represent a GF-system do we need three types of objects: A system object, a component object and a minimal flow cut set object.
The system needs lists that says which components and minimal flow cut sets that are in the system. The component object needs to know which minimal flow cut sets it is in and also what state it has. When we have this then we have a good starting point. So let us get more into it.

## 3.2 Some practical information about the simulation

The simulation is a *discrete event simulation* (DES) which means that a state change is occurring at a given time. The given time has a distribution which says when the next state change is going to happen.
To perform a DES it is four main elements that we need: States, a timer, number of simulations and an event handler. [1]
The event handler finds the next event, and is the one that executes when the next event is happening. In our case is the event handler the component class. The event handler tells the event queue when a new event is going to happen, and what's going to happen. The event queue alerts the event handler when the event is happening and what it should do.

The time distributions in this simulation is an exponential distribution with parameter $\lambda = 2.0$.

$$\text{The exponential distribution: } f(x, \lambda) = \lambda e^{-\lambda x},\ \lambda > 0,\ x \in [0, \infty]$$

[2]

## 3.3 Component class

In short the component class has to contain all the states it can go to and what state it currently is in. It needs a starting state for when the simulation is starting. For practical reasons will the component have a name and have a component number.
Example
Component one will be called x1 and have component number 0.

The component also needs to know which minimal flow cut sets it is in to alert them when it changes its state. The Component will be the class that handles the simulation events. When the program is finished with the simulation it needs to save the current state it is in. This value will later be used in a class that calculates the bounds.

## 3.4 Cut set class

The minimal flow cut set class needs a list with the components that are in it. When the class is starting, it needs to find all the states it can be in, what the max state is, then sets what system class it belongs to, and what index it has in the system class, and the components' classes minimal flow cut set lists.

The minimal flow cut set class alerts the system class when it's updates it state.
As the component class this class also saves it's current state when the program is finished with the simulation.

## 3.5 System class

The system class needs to know all the component classes, and all the minimal flow cut set classes that are in the system. The system class needs to find all the minimal path vectors, and all the minimal cut vectors. To make this more easy it has got its own class where it finds all the vectors, let us call it find vector.
As the two other classes this class also saves it's current state when the program is finished with the simulation.

## 3.6 Find vector class

These classes purpose is to find all the minimal cut vectors, and all the minimal path vectors. It uses the new theorem 2.1 and the 3 scenarios with the special case where all the components have the same states to find the minimal cut vectors, and it uses a similar approach to find the minimal path vectors. After finding one vector does the program store it in a list for later uses.

### 3.6.1 Algorithm for Minimal cut Vectors

Make a list minCutVectors that will contain all the minimal cut vectors

For each element in system states

> Make a start list that contains the components and their corresponding max state
>
> For all components in a cut set
>
> > Make a copy of the starting list
> >
> > Go through all the component states until the cut set state is equal to the system state element before the current system state element
> >
> > Change the states for the components that has changed it's state in the copied list to the new component states

Check that all cut sets with an element size smaller than the current cut set have a cut state larger or equal to the system state. If this holds store the copied list in minCutVectors

### 3.6.2 Algorithm for Minimal path Vectors

Make a list minPathVectors that will contain all the minimum path vectors

Make a start list that contains the components and their corresponding max state

For each element in system states

Make a copy of the starting list

Change the component states until all the cut sets state is equal the current system state

Change the states for the components that has changed it's state in the copied list to the new component states and store the copied list in minPathVectors

## 3.7 Calculate class

After the simulation we can start to calculate the different bounds. To do that we need to find the $r_i$ for all components, minimal flow cut sets and for the system. The values we can find by using the stored values of current state, and divide it by number of simulations that has been done. When we now have found the $r$'s we can easily find $p_i$ for components, minimal flow cut sets and for the system. The latter one is the estimated value for the system, this value converges to the real value if we have enough simulations. We find $p$ by adding the $r$'s For example:

$$r_{\text{ state 3}} = 0.1,\ r_{\text{ state 2}} = 0.3,\ r_{\text{ state 1}} = 0.4,\ r_{\text{ state 0}} = 0.2$$

then we have

$$p_{\text{ state 3}} = r_{\text{ state 3}} = 0.1$$
$$p_{\text{ state 2}} = p_{\text{ state 3}} + r_{\text{ state 2}} = 0.4$$
$$p_{\text{ state 1}} = p_{\text{ state 2}} + r_{\text{ state 1}} = 0.8$$
$$p_{\text{ state 0}} = p_{\text{ state 1}} + r_{\text{ state 0}} = 1$$

Now that the program has the $p$ values it can start to find the established lower bound, and the new bound. After we have found them the program is finished, and uses then R to make the plots.

### 3.7.1 Algorithm for the new bound here called $l_j$

Make a list $l_j$ that contains all the float values and set the values equal 0

Make a float variable Value

Make a double list stateSums where we add the
components current state to

Do a for each on the components and a for each for all the component states in an other function where the input values is Value and stateSums

> Make a new float variable tempValue and set it equal
> Value times the components state corresponding $r_i$ value
>
> When at the last component find the cut with lowest state based on the stateSums
>
> Plus tempValue to all elements in $l_j$ that is bigger or equal to
> lowest cut state.
>
> Repeat till there are no more component states.

# 4 Results

## 4.1 Check of the program

In this section we will look at the example 4 in the article [4].

**Example 4. [4]** Let $C = \{1, 2\}, S_1 = S_2 = \{0, 1, 2, ..., 5\}, S = \{0, 1, 2, ..., 10\}$, and $\phi(\mathbf{x}) = x_1 + x_2$. Assume that the component states are independent. Let $\mathbf{p}_1 = \mathbf{p}_2 = (0.95, 0.90, 0.85, 0.80, 0.75)$. In the article are only showing lower bounds for the system availability to levels 9,6 and 3. But here I will present for all levels.

For level 1:

Minimal path set:

| Path 1 | Path 2 |
|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 |
| X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{'1}(\mathbf{P}_{\phi}) = 0.95$$

Minimal cut set:

| Cut 1 |
|---|
| X1 is in state: 0.0 |
| X2 is in state: 0.0 |

$$\ell_{\phi}^{**1}(\mathbf{P}_{\phi}) = 0.9975,\ \tilde{\ell}_{\phi}^{1}(\mathbf{P}_{\phi}) = 0.9975$$

For level 2:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 2.0 | X2 is in state: 1.0 | X2 is in state: 1.0 |

$$\ell_{\phi}^{'2}(\mathbf{P}_{\phi}) = 0.9025$$

Minimal cut set:

| Cut 1 | Cut 2 |
|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 |
| X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{**2}(\mathbf{P}_{\phi}) = 0.9900,\ \tilde{\ell}_{\phi}^{2}(\mathbf{P}_{\phi}) = 0.9925$$

For level 3:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 3.0 | X2 is in state: 2.0 | X2 is in state: 1.0 |

| Path 4 |
|---|
| X1 is in state: 3.0 |
| X2 is in state: 0.0 |

$$\ell_{\phi}^{'3}(\mathbf{P}_{\phi}) = 0.8550$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 2.0 | X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{**3}(\mathbf{P}_{\phi}) = 0.9752,\ \tilde{\ell}_{\phi}^{3}(\mathbf{P}_{\phi}) = 0.9850$$

For level 4:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 4.0 | X2 is in state: 3.0 | X2 is in state: 2.0 |

| Path 4 | Path 5 |
|---|---|
| X1 is in state: 3.0 | X1 is in state: 4.0 |
| X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{'4}(\mathbf{P}_{\phi}) = 0.8100$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 3.0 | X2 is in state: 2.0 | X2 is in state: 1.0 |

| Cut 4 |
|---|
| X1 is in state: 3.0 |
| X2 is in state: 0.0 |

$$\ell_{\phi}^{**4}(\mathbf{P}_{\phi}) = 0.9509,\ \tilde{\ell}_{\phi}^{4}(\mathbf{P}_{\phi}) = 0.9750$$

For level 5:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |
| Path 4 | Path 5 | Path 6 |
| X1 is in state: 3.0 | X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 2.0 | X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{\prime 5}(\mathbf{P}_{\phi}) = 0.7650$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 4.0 | X2 is in state: 3.0 | X2 is in state: 2.0 |
| Cut 4 | Cut 5 | |
| X1 is in state: 3.0 | X1 is in state: 4.0 | |
| X2 is in state: 1.0 | X2 is in state: 0.0 | |

$$\ell_{\phi}^{**5}(\mathbf{P}_{\phi}) = 0.9155,\ \tilde{\ell}_{\phi}^{5}(\mathbf{P}_{\phi}) = 0.9625$$

For level 6:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 1.0 | X1 is in state: 2.0 | X1 is in state: 3.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |
| Path 4 | Path 5 | |
| X1 is in state: 4.0 | X1 is in state: 5.0 | |
| X2 is in state: 2.0 | X2 is in state: 1.0 | |

$$\ell_{\phi}^{\prime 6}(\mathbf{P}_{\phi}) = 0.7225$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 0.0 | X1 is in state: 1.0 | X1 is in state: 2.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |
| Cut 4 | Cut 5 | Cut 6 |
| X1 is in state: 3.0 | X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 2.0 | X2 is in state: 1.0 | X2 is in state: 0.0 |

$$\ell_{\phi}^{**6}(\mathbf{P}_{\phi}) = 0.8072,\ \tilde{\ell}_{\phi}^{6}(\mathbf{P}_{\phi}) = 0.8775$$

For level 7:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 2.0 | X1 is in state: 3.0 | X1 is in state: 4.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |

| Path 4 |
|---|
| X1 is in state: 5.0 |
| X2 is in state: 2.0 |

$$\ell_{\phi}^{'7}(\mathbf{P}_{\phi}) = 0.6800$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 1.0 | X1 is in state: 2.0 | X1 is in state: 3.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |

| Cut 4 | Cut 5 |
|---|---|
| X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 2.0 | X2 is in state: 1.0 |

$$\ell_{\phi}^{**7}(\mathbf{P}_{\phi}) = 0.7204,\ \widetilde{\ell}_{\phi}^{7}(\mathbf{P}_{\phi}) = 0.7945$$

For level 8:

Minimal path set:

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| X1 is in state: 3.0 | X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |

$$\ell_{\phi}^{'8}(\mathbf{P}_{\phi}) = 0.6400$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 2.0 | X1 is in state: 3.0 | X1 is in state: 4.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |

| Cut 4 |
|---|
| X1 is in state: 5.0 |
| X2 is in state: 2.0 |

$$\ell_{\phi}^{**8}(\mathbf{P}_{\phi}) = 0.6521,\ \widetilde{\ell}_{\phi}^{8}(\mathbf{P}_{\phi}) = 0.7150$$

For level 9:

Minimal path set:

| Path 1 | Path 2 |
|---|---|
| X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 |

$$\ell_{\phi}^{\prime 9}(\mathbf{P}_{\phi}) = 0.6000$$

Minimal cut set:

| Cut 1 | Cut 2 | Cut 3 |
|---|---|---|
| X1 is in state: 3.0 | X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 5.0 | X2 is in state: 4.0 | X2 is in state: 3.0 |

$$\ell_{\phi}^{**9}(\mathbf{P}_{\phi}) = 0.6000,\ \tilde{\ell}_{\phi}^{9}(\mathbf{P}_{\phi}) = 0.6375$$

For level 10:

Minimal path set:

| Path 1 |
|---|
| X1 is in state: 5.0 |
| X2 is in state: 5.0 |

$$\ell_{\phi}^{\prime 10}(\mathbf{P}_{\phi}) = 0.5625$$

Minimal cut set:

| Cut 1 | Cut 2 |
|---|---|
| X1 is in state: 4.0 | X1 is in state: 5.0 |
| X2 is in state: 5.0 | X2 is in states: 4.0 |

$$\ell_{\phi}^{**10}(\mathbf{P}_{\phi}) = 0.5625,\ \tilde{\ell}_{\phi}^{10}(\mathbf{P}_{\phi}) = 0.5625$$

The program gives the same values and minimal cut and path sets as for the levels that are calculated in the article [4]. The only difference is at level 3 where it is a typo in the article.

## 4.2 The different systems

I have chosen to use 4 different systems that are related to each other in the structural way. The reason for this decision is to make it more easy to compare what is happening to the different bounds when we add a new serial-connection. The simplest one of them is the system with 8 components.

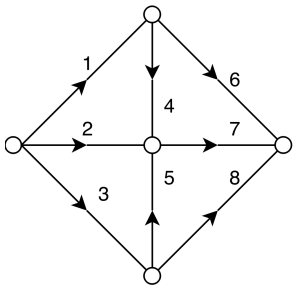



**Figure 5:** System with 8 components

Then we have the more complex one with 11 components. There it is added three serial-connections at component 1, 2 and 3 from the previous system.

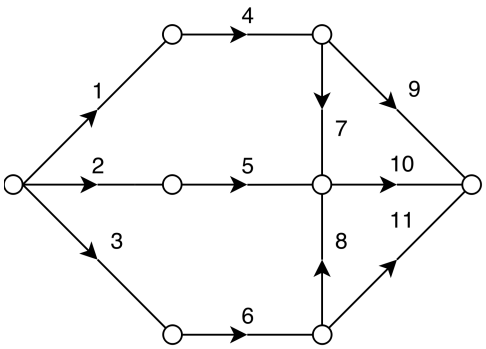



**Figure 6:** System with 11 components

The third system has 13 components. Here it is added two serial-connections at component 7 and 8 from the previous system.

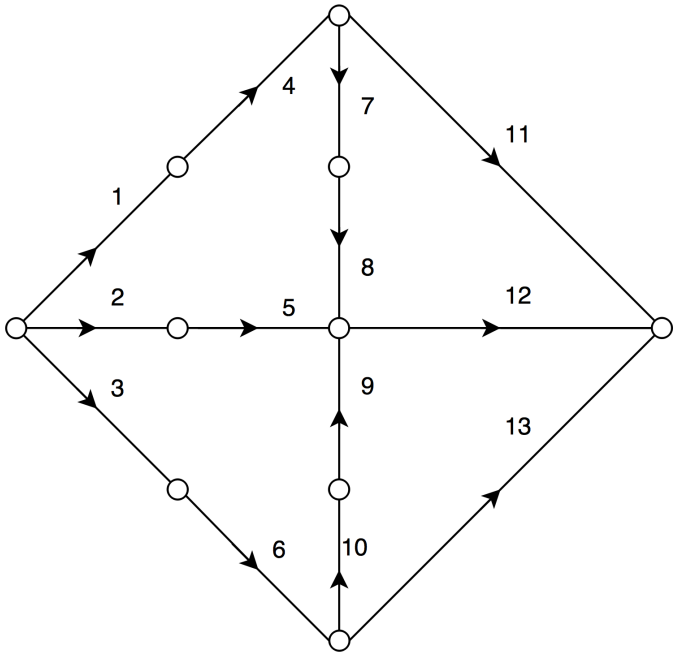



**Figure 7:** System with 13 components

At last we do have the largest system that we are going to look at. The system has 16 components. Here it is added two serial-connections at component 11, 12 and 13 from the previous system.

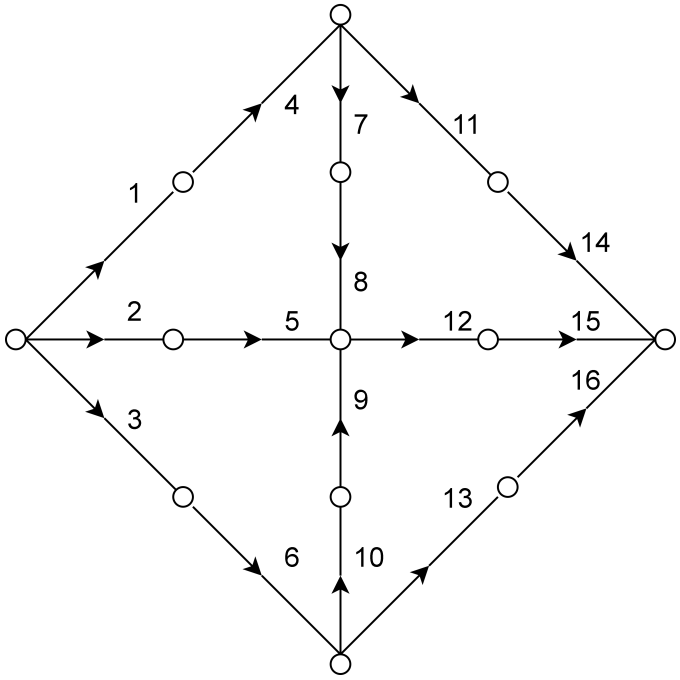



**Figure 8:** System with 16 components

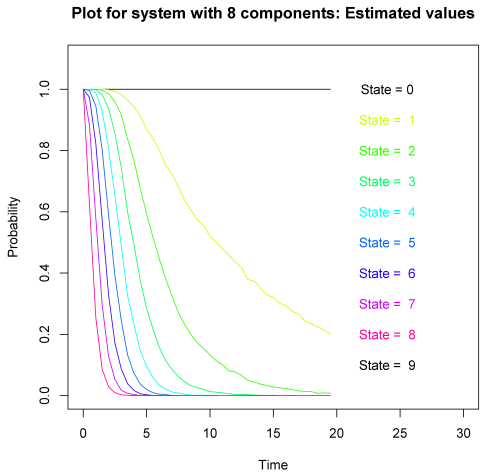
As you can see it is a large similarity between this three systems. This similarity we will use to our advantage to make our comparison more easy.

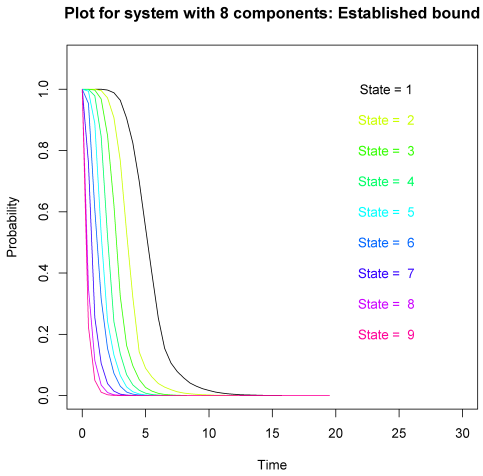
## 4.3 System with 8 components

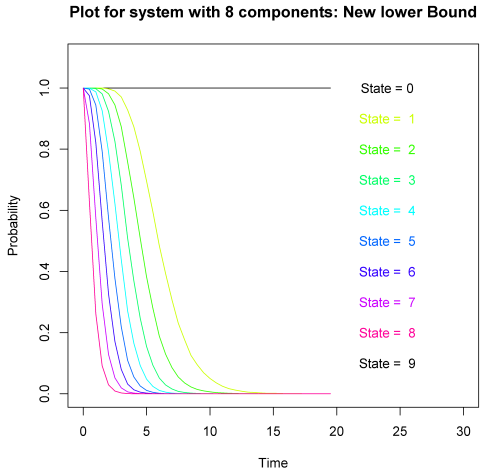
Let us first look at the system with 8 components. This system is a good starting point to check the different bounds, and how they are compared to the estimated value $p$.

### 4.3.1 Plots for the different bounds

First of all I will show three plots where you can see how the different bounds are changing over time.



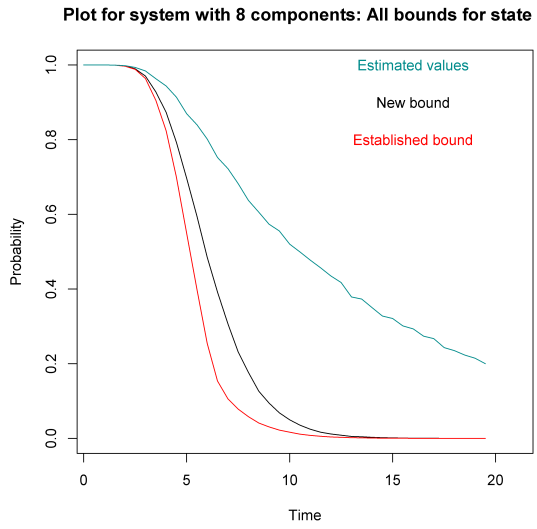



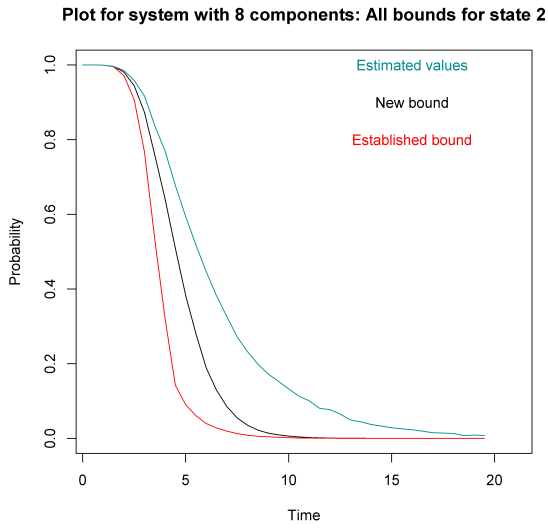



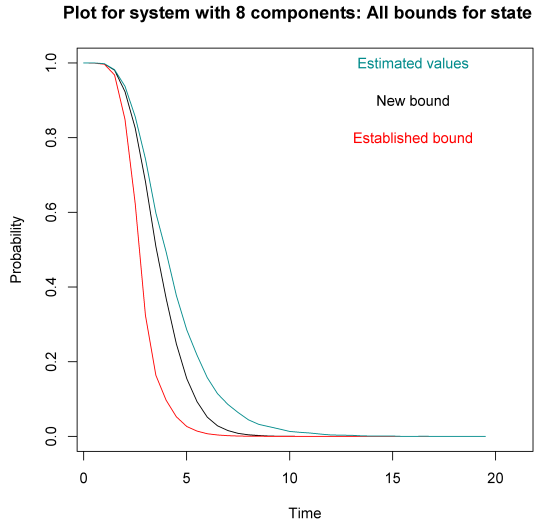

If we compare the established lower bound and the new lower bound against the estimated value, we see that both of the bounds miss on low states.

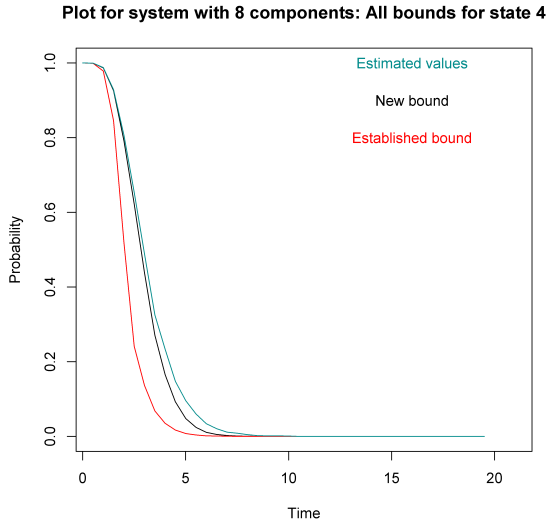
### 4.3.2 Plots one state at a time

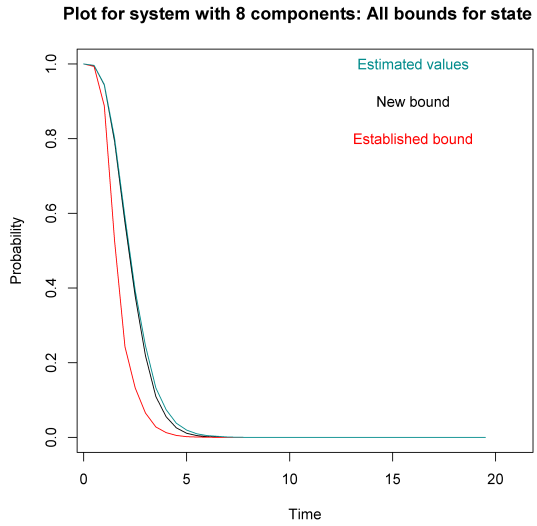
Let us plot all the bounds at one state at a time to better see the differences.

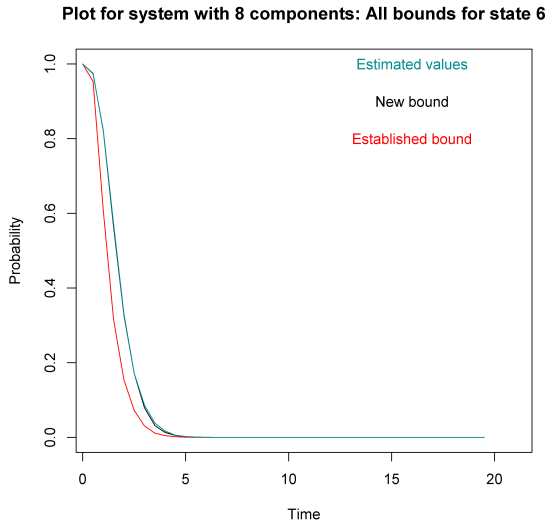


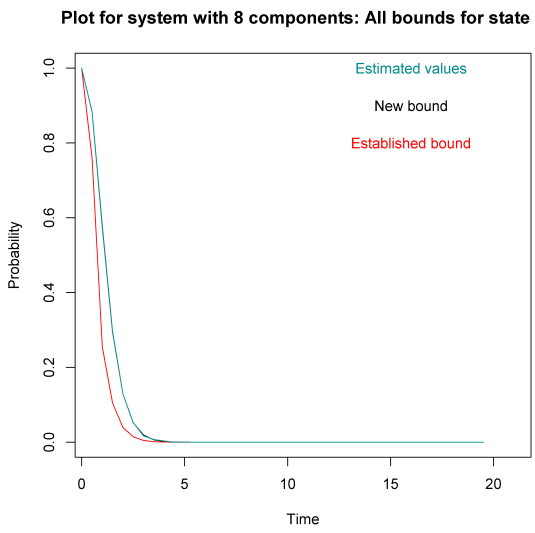



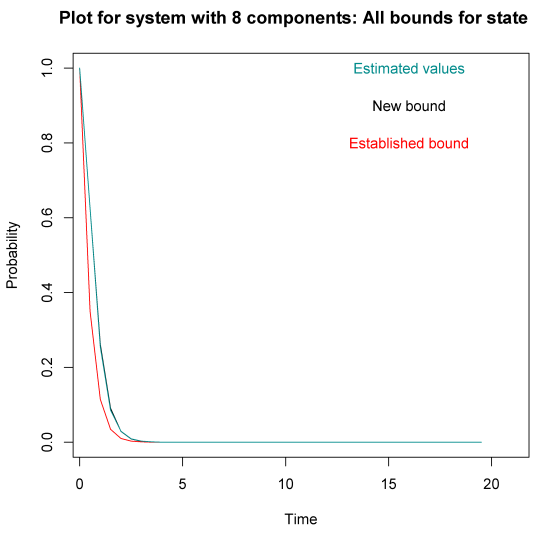



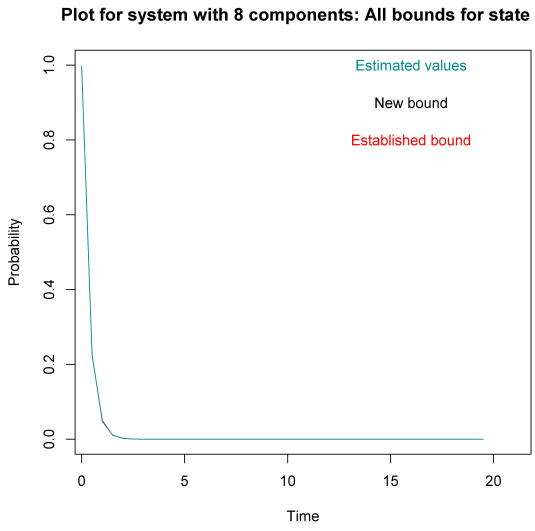





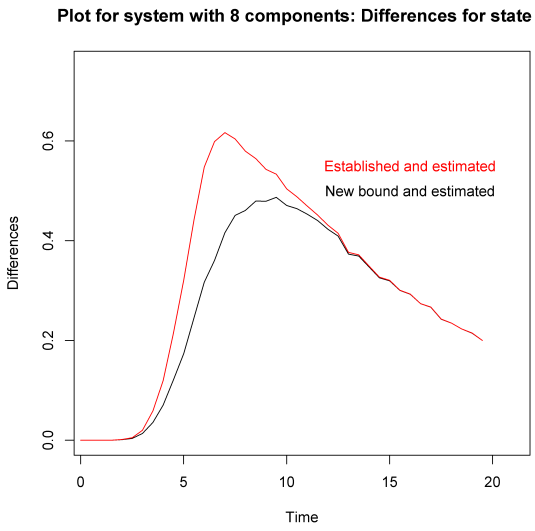



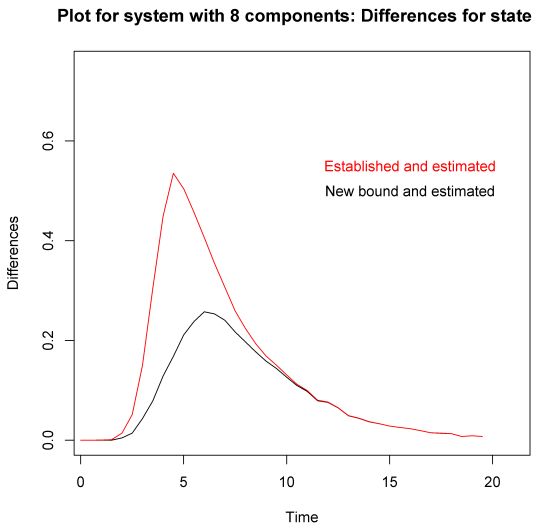



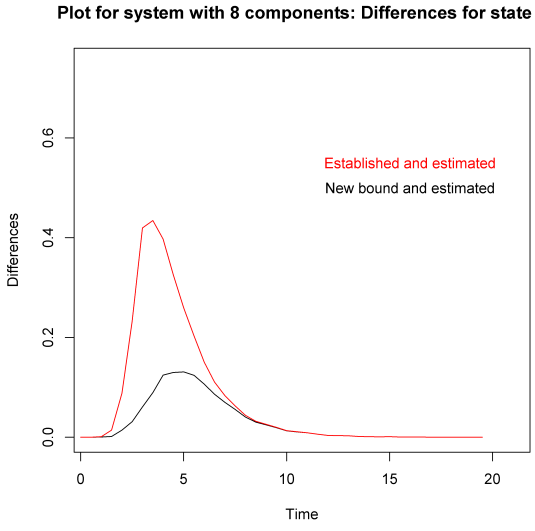

The new bound looks better than the old established one. Let us first find the differences between the lower bounds against the estimated value. This makes it easier to see how much better the new bound is.

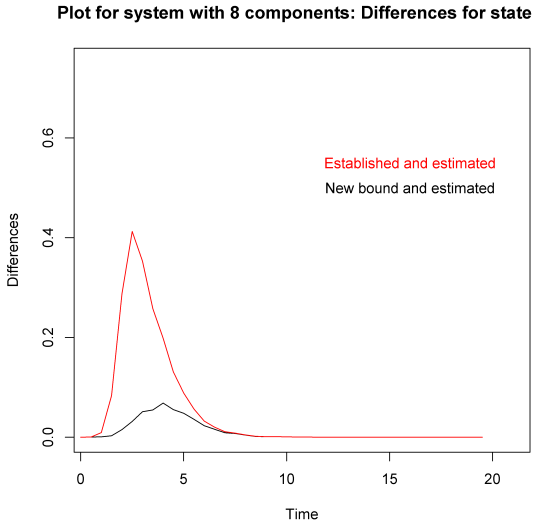
### 4.3.3 Plots of the differences

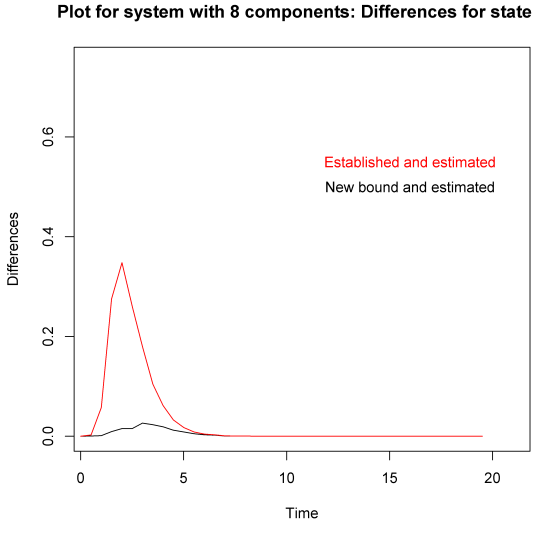
The differences are found by taking the value for the estimated value and subtract it by the value to one of the other bounds. The closer the plot is to zero, the closer is the bound to the estimated value. Let us plot the difference between the estimated value against the two others.

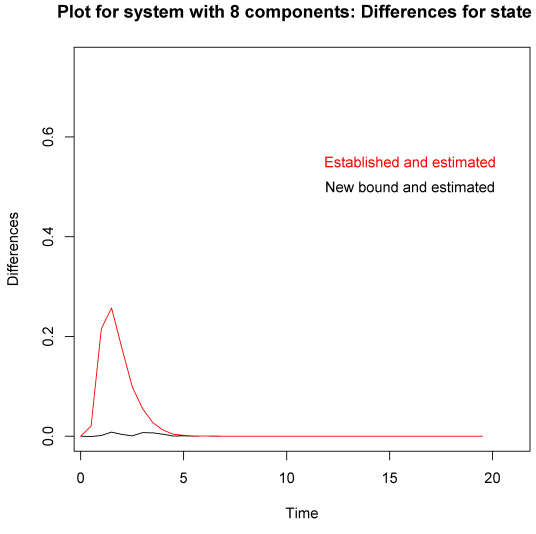

Plot for system with 8 components: Differences for state 1
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20

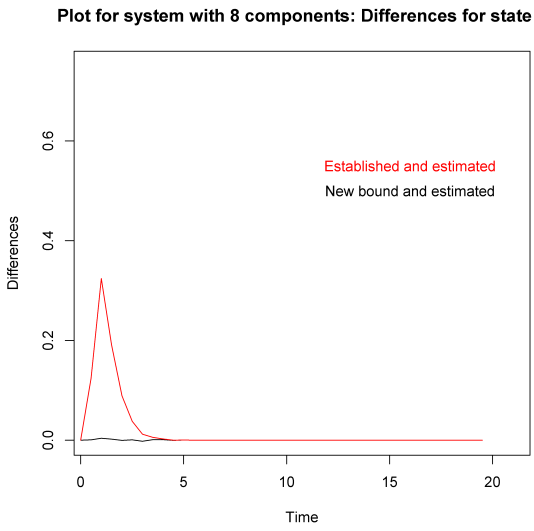

Plot for system with 8 components: Differences for state 2
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20

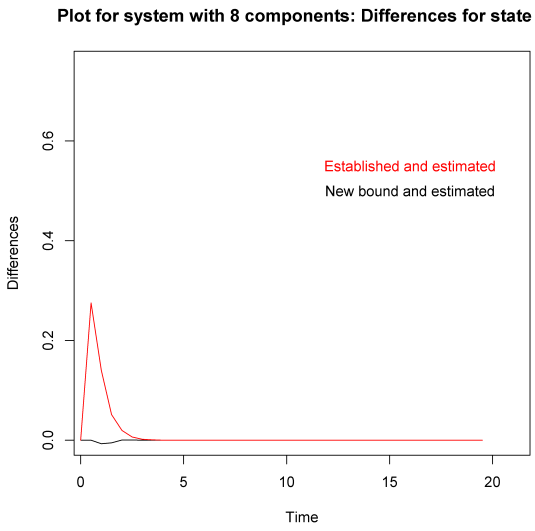

Plot for system with 8 components: Differences for state 3
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20

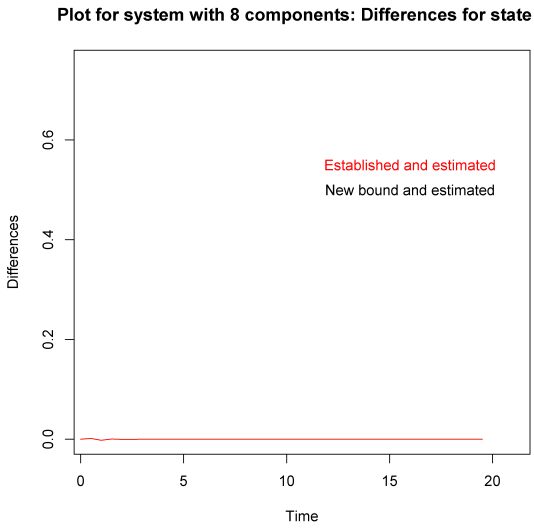

Plot for system with 8 components: Differences for state 4
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20


Plot for system with 8 components: Differences for state 5
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20


Plot for system with 8 components: Differences for state 6
Established and estimated
New bound and estimated
Differences
Time
0.0
0.2
0.4
0.6
0
5
10
15
20




### 4.3.4 Summary

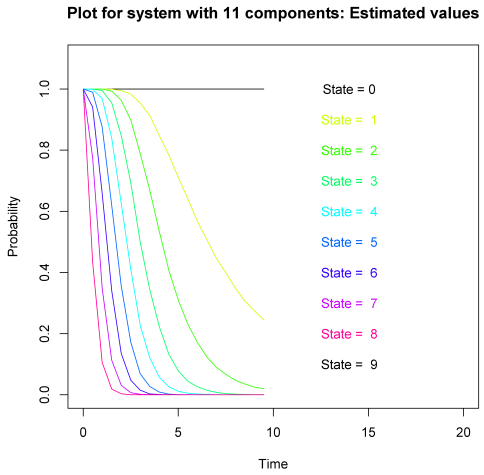
As you can see the new bound is good, and hits the estimated value on many points. The new bound gets faster similar the estimated value then what the established one does. Both of the bounds miss on low state values.

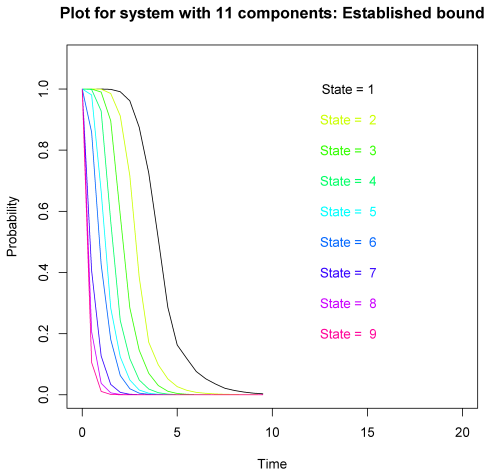
## 4.4 System with 11 components

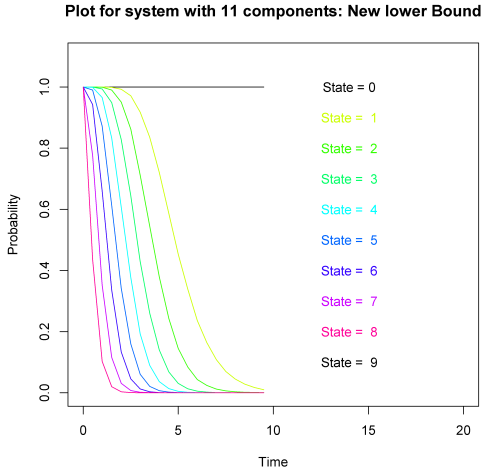
Let us now look at the system with 11 components. This system is extended with 3 more component's as shown earlier.

### 4.4.1 Plots for the different bounds

As for the system with 8 components I will first show three plots where you can see how the different bounds are changing over time.



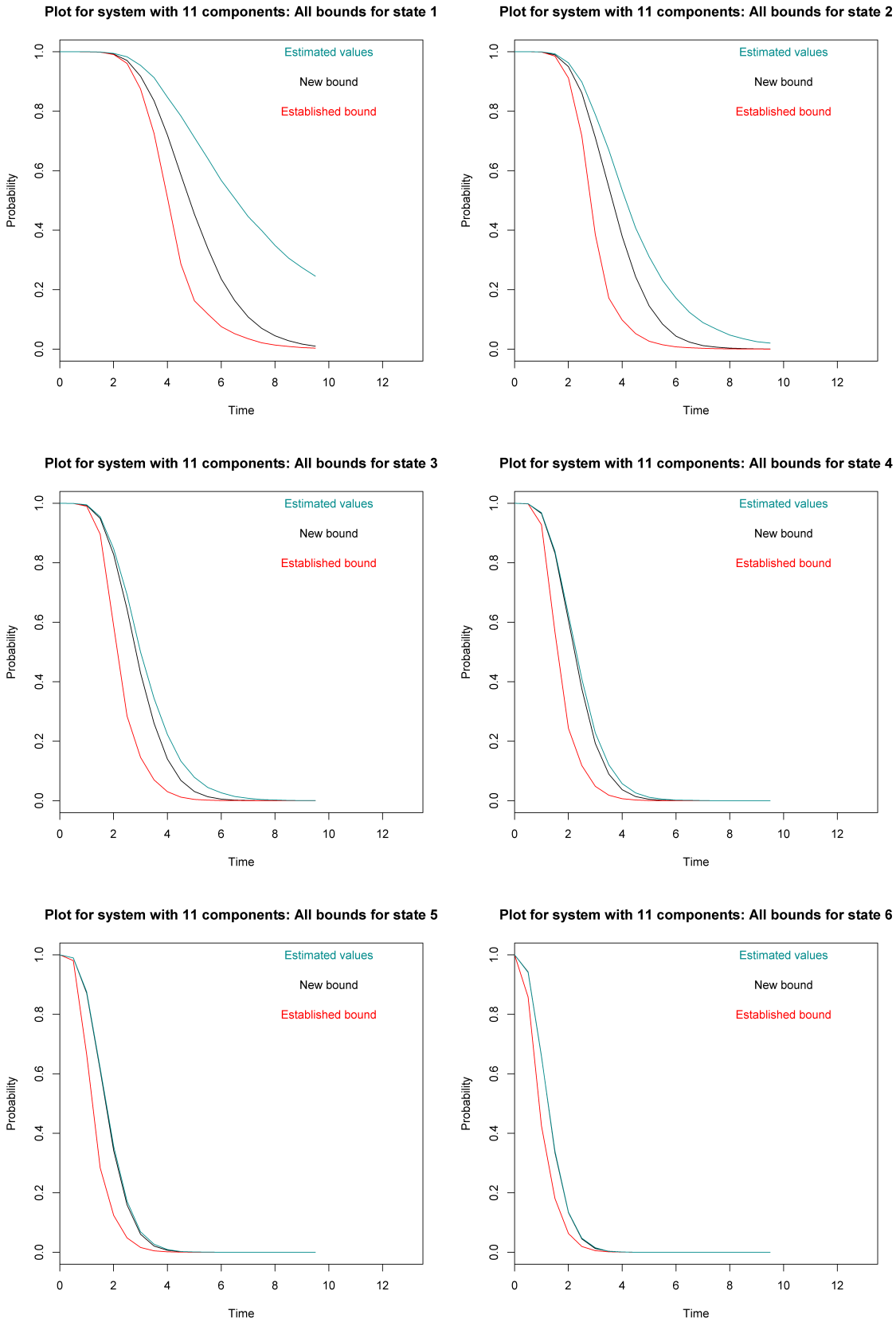



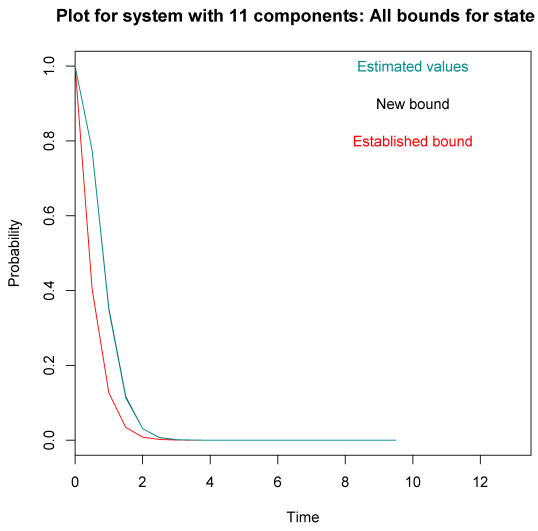



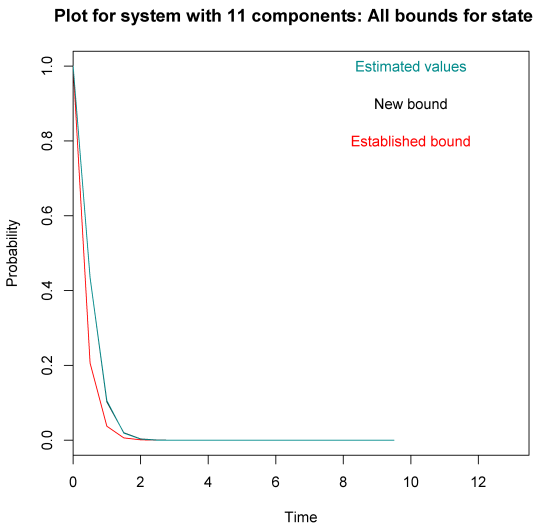

If we compare these new plots for established lower bound and the new lower bound against the estimated value, it now seams like the two bounds look much more like the estimated value.

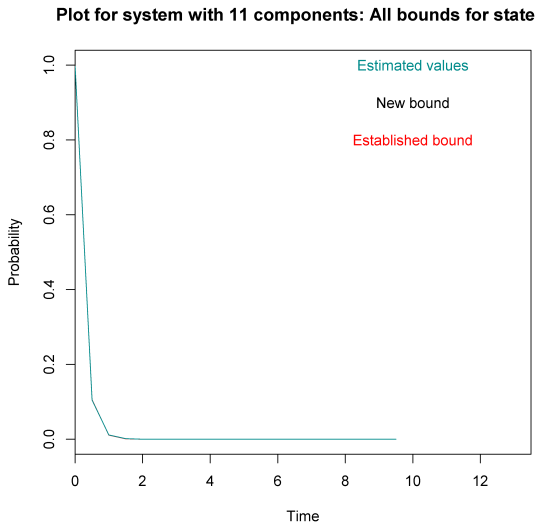
### 4.4.2 Plots one state at a time

Let us plot all the bounds at one state at a time to better see the differences.





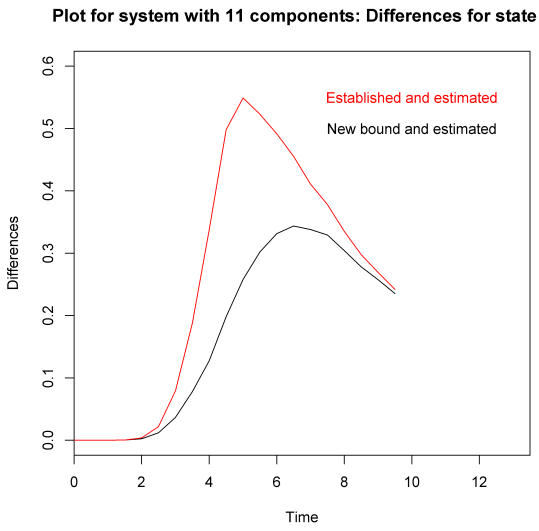



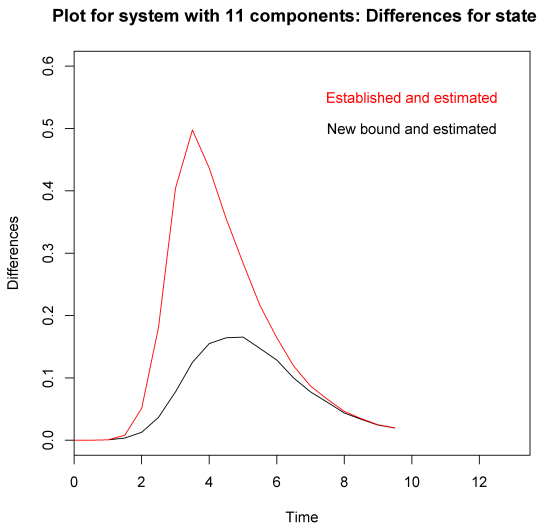



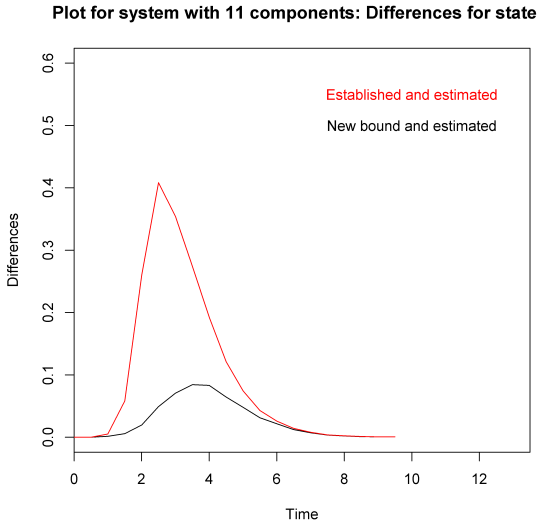

The bounds are now much closer to the estimated value. The new lower bound does here also look closer to the estimated value than what the established lower bound does.

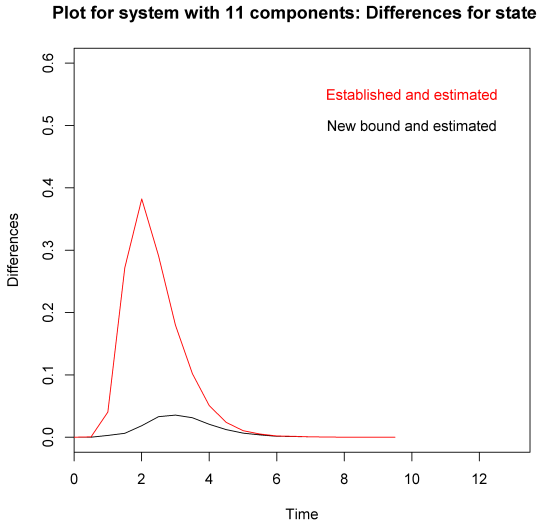
### 4.4.3 Plots of the differences

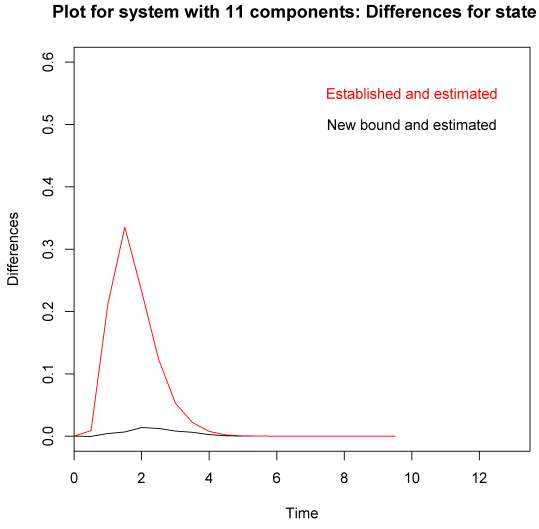
The differences are found by taking the value for the estimated value and subtract it by the value to one of the other bounds. The closer the plot is to zero, the closer is the bound to the estimated value. Let us plot the difference between the estimated value and the two others.

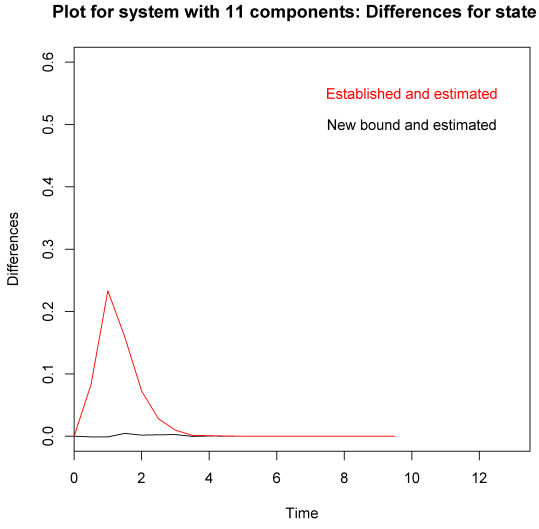

Plot for system with 11 components: Differences for state 1
Established and estimated
New bound and estimated
Differences
Time

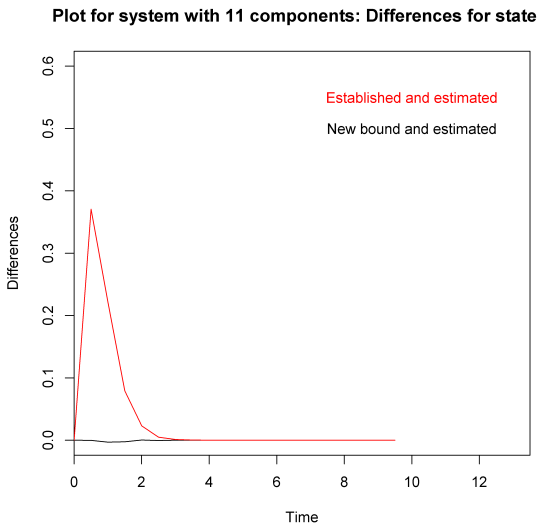

Plot for system with 11 components: Differences for state 2
Established and estimated
New bound and estimated
Differences
Time

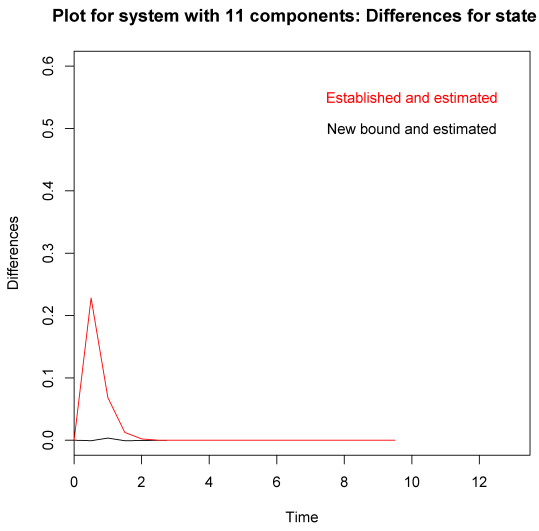

Plot for system with 11 components: Differences for state 3
Established and estimated
New bound and estimated
Differences
Time

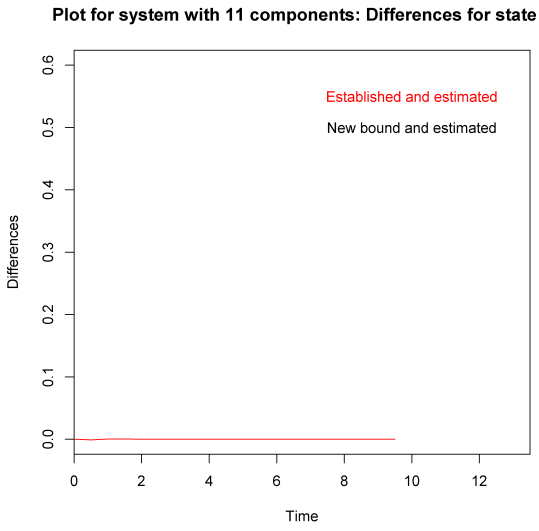

Plot for system with 11 components: Differences for state 4
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 11 components: Differences for state 5
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 11 components: Differences for state 6
Established and estimated
New bound and estimated
Differences
Time




### 4.4.4 Summary

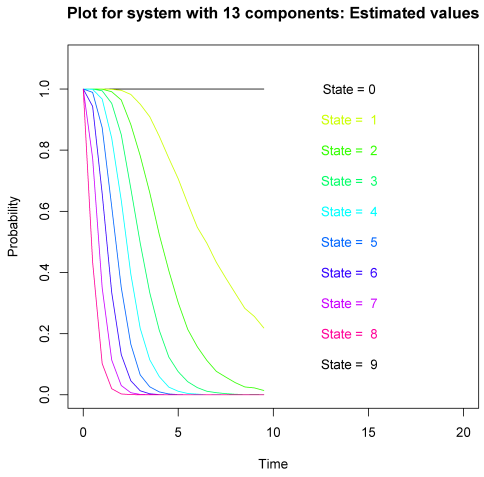
As you can see the new lower bound is better than the established lower bound. As time passes, the values of both bounds will be more similar and after some time be equal to the estimated value. The value of the new bound will faster be more similar, but also gradually be equal to the estimated value. The established bound is slower to get similar to the estimated value. Both of the bounds are closer to the estimated value at high states than at low states.

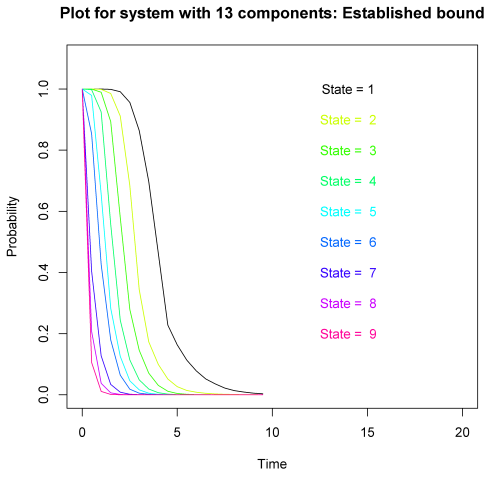
## 4.5 System with 13 components

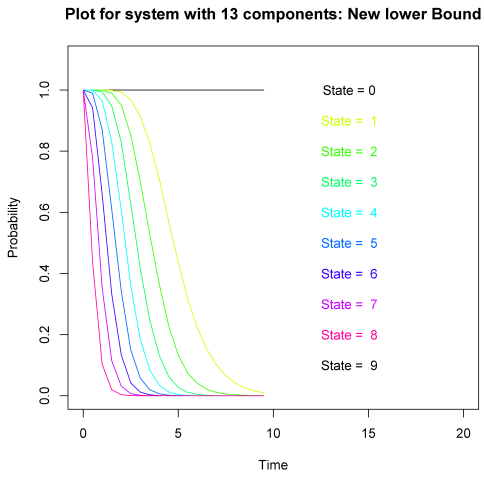
Let us now look at the system with 13 components. This system is a extension to the system with 11 components and here has added 2 more components.

### 4.5.1 Plots for the different bounds

As for the two other systems I will first show three plots where you can see how the different bounds are changing over time.



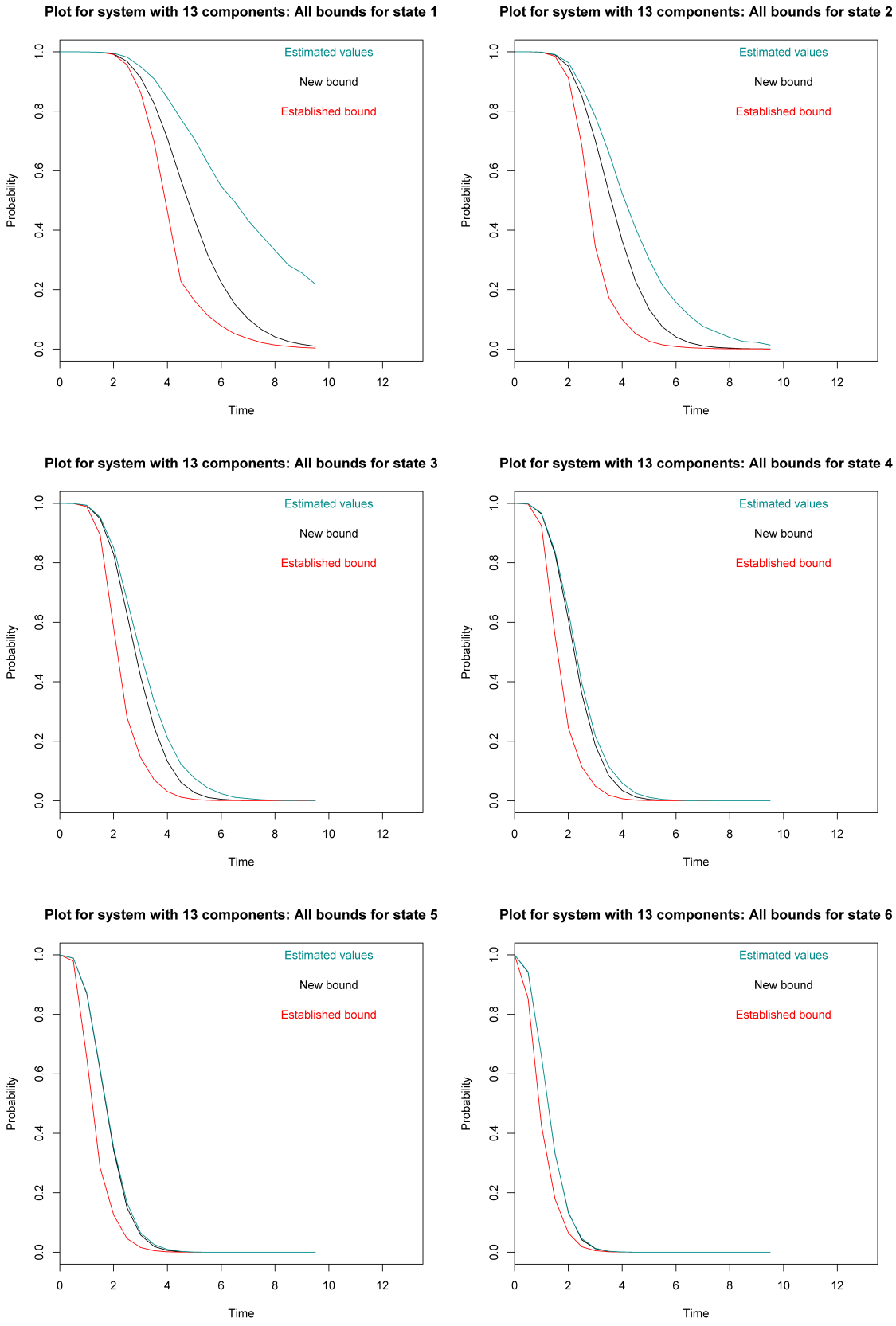



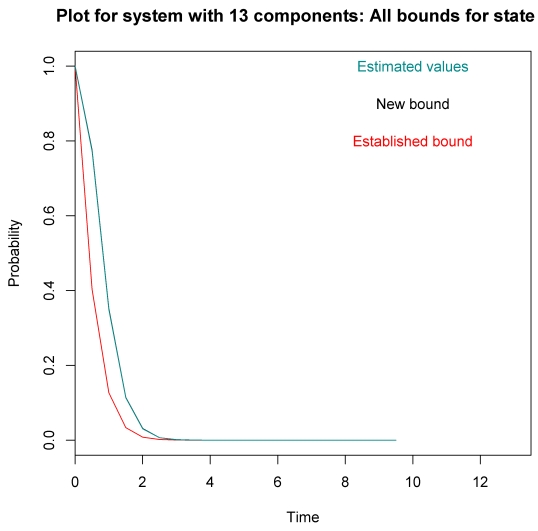



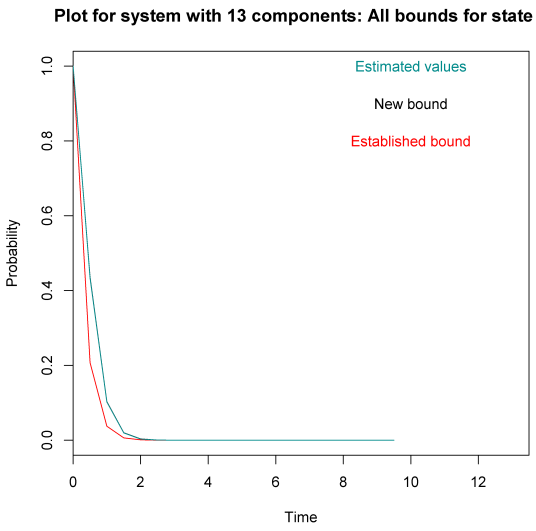

If we compare these new plots for the established lower bound, and the new lower bound against the estimated value, it seams like as for the system with 11 components that the two bounds look much more like the estimated value.

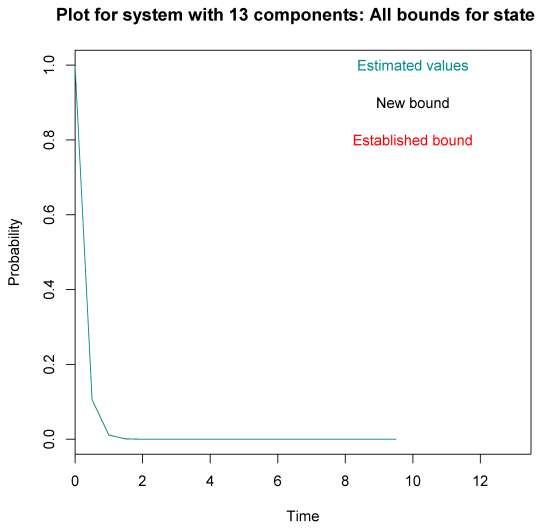
### 4.5.2 Plots one state at a time

Let us plot all the bounds at one state at a time to better see the differences.





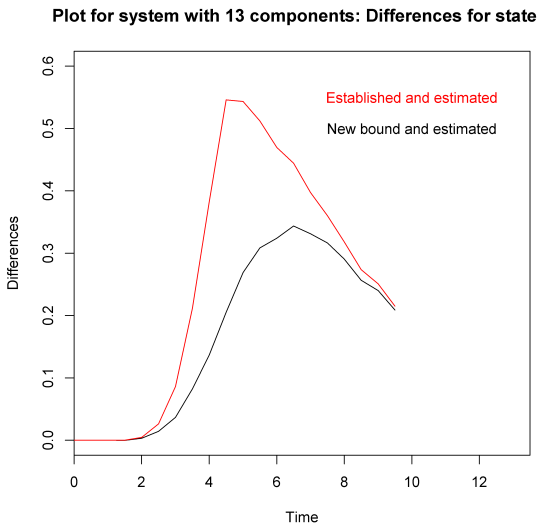



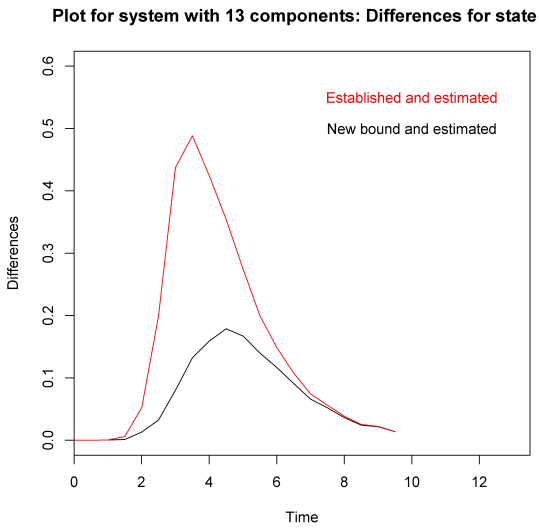



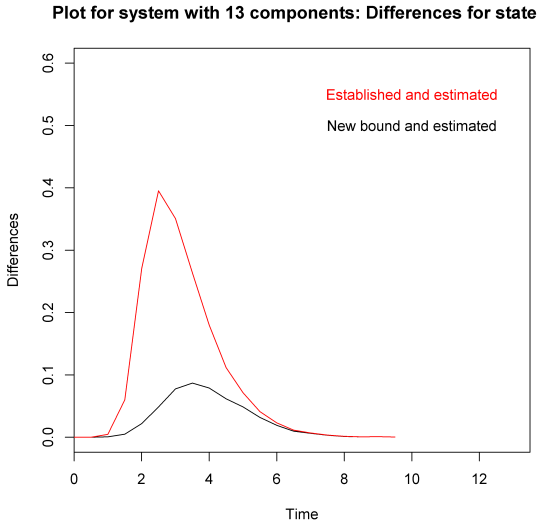

The new lower bound is also here closer to the estimated value than what the established lower bound is.

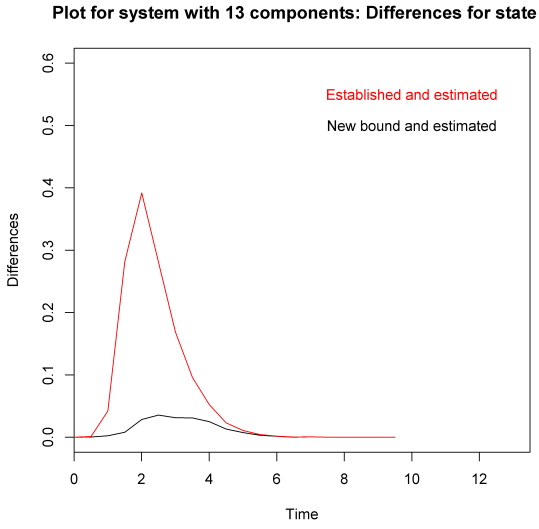
### 4.5.3 Plots of the differences

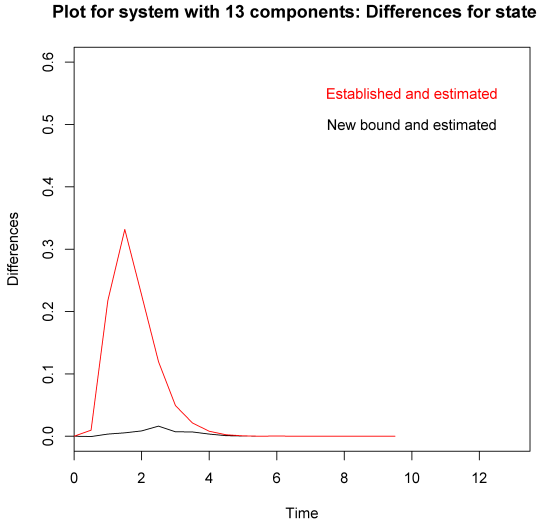
The differences are found by taking the value for the estimated value and subtract it by the value to one of the other bounds. The closer the plot is to zero, the closer the bound is to the estimated value. Let us plot the difference between the estimated value and the two others.

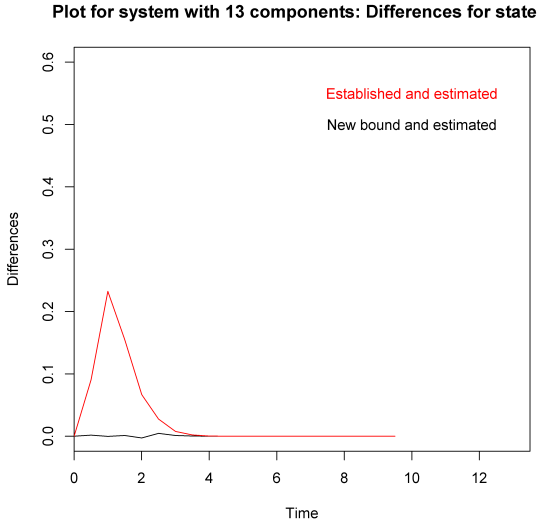

Plot for system with 13 components: Differences for state 1
Established and estimated
New bound and estimated
Differences
Time

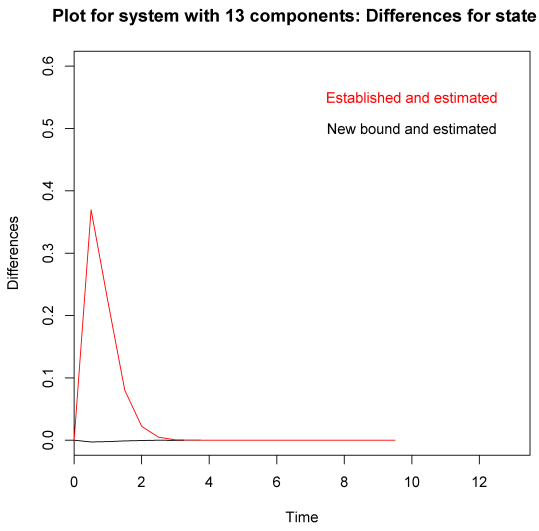

Plot for system with 13 components: Differences for state 2
Established and estimated
New bound and estimated
Differences
Time

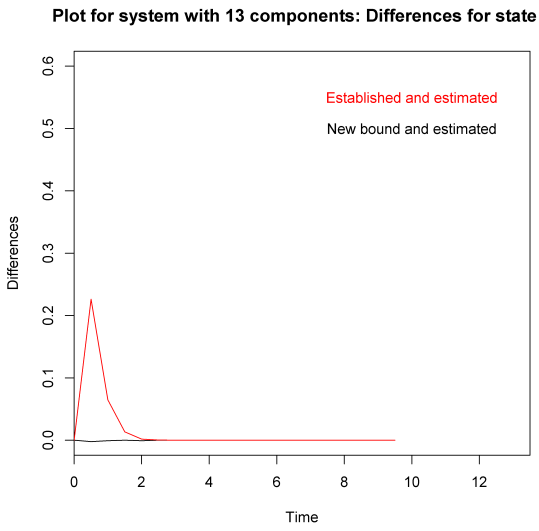

Plot for system with 13 components: Differences for state 3
Established and estimated
New bound and estimated
Differences
Time

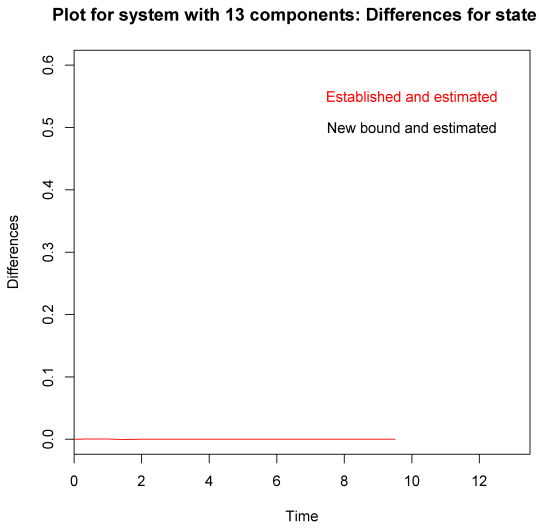

Plot for system with 13 components: Differences for state 4
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 13 components: Differences for state 5
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 13 components: Differences for state 6
Established and estimated
New bound and estimated
Differences
Time




### 4.5.4 Summary

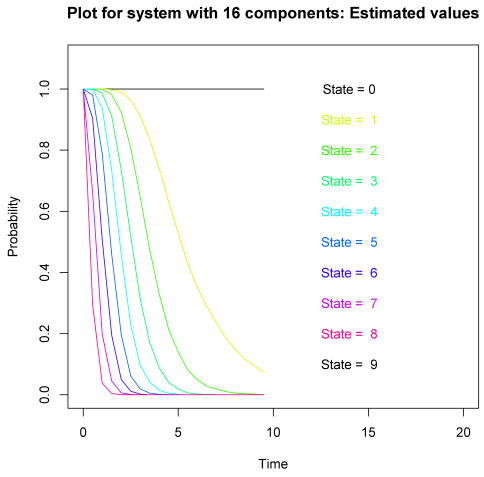
The new bound gets even faster similar to the estimated value and it seams like the new bound is equal to the estimated value after state 5. As time passes, the values of both bounds will be more similar and after some time be equal to the estimated value. The value of the new bound will faster be more similar, but also gradually be equal to the estimated value. The established bound is slower to get similar to the estimated value. It seams like the established lower bound is equal to the estimated value at state 9.

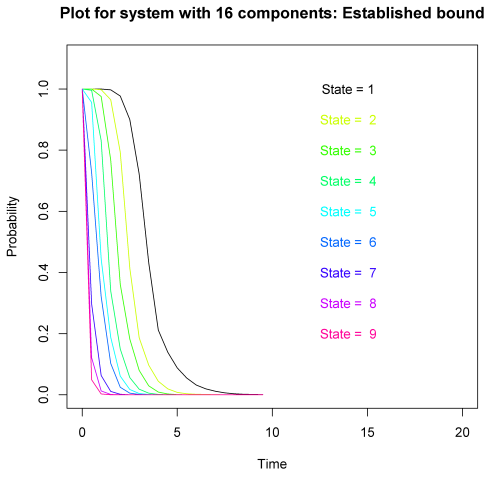
## 4.6 System with 16 components

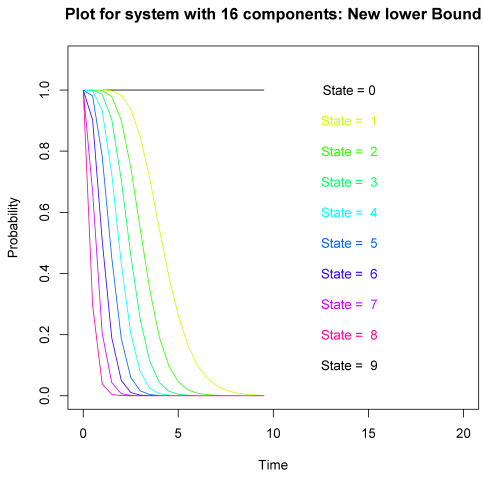
Let us now look at the system with 16 components. This system is an extension of the system with 13 components and has here added 3 more components.

### 4.6.1 Plots for the different bounds

As for the other systems, I will first show three plots where you can see how the different bounds are changing over time.



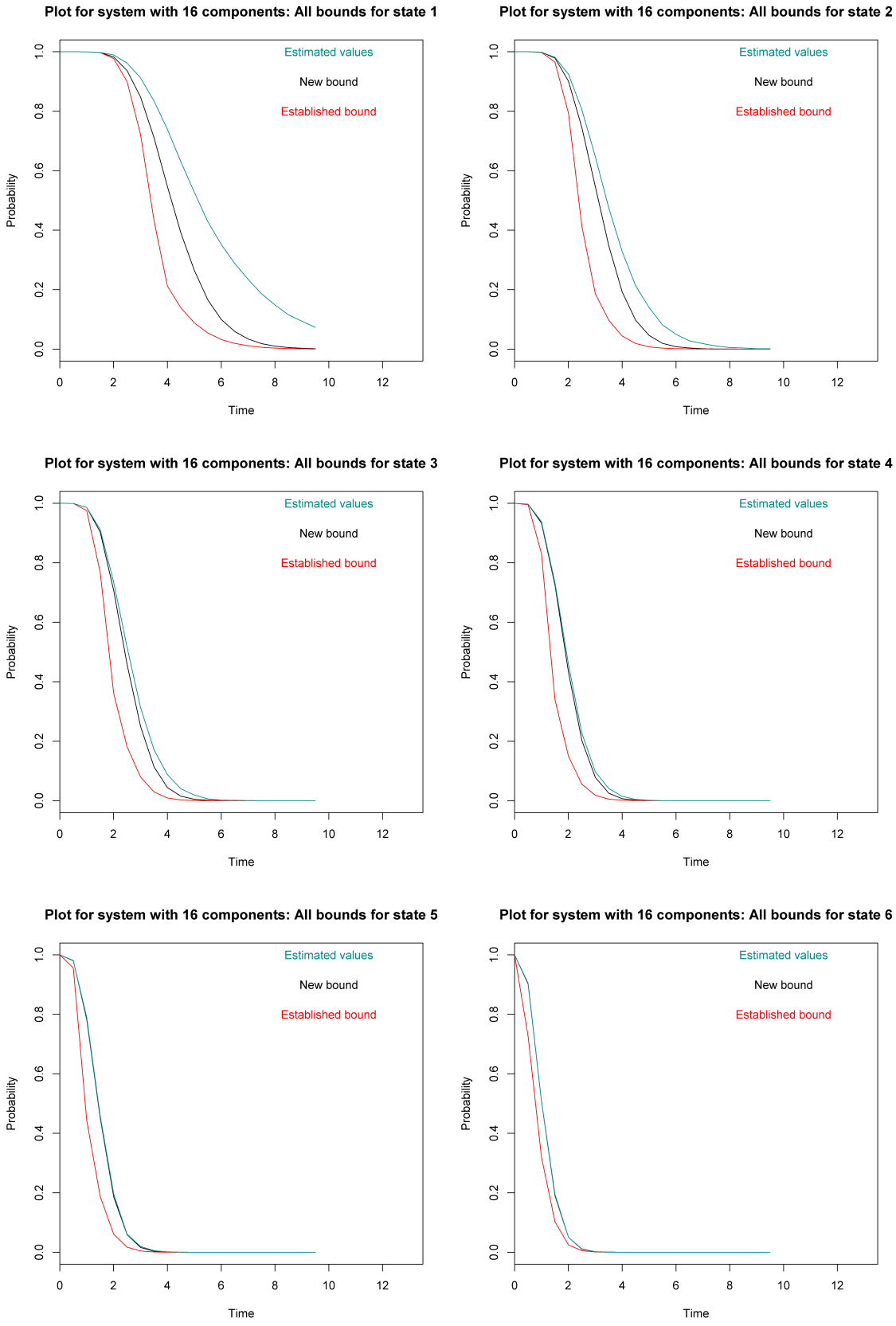



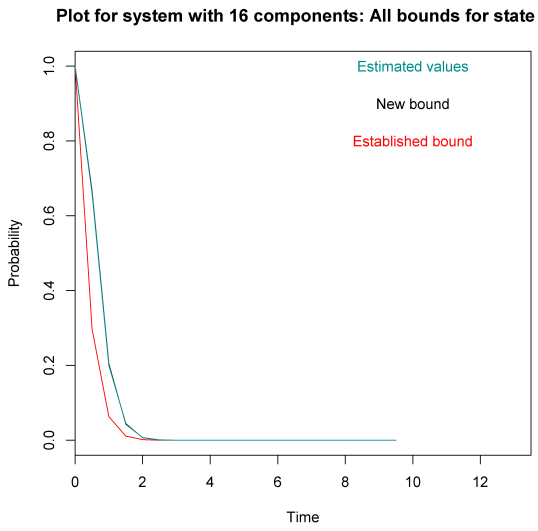



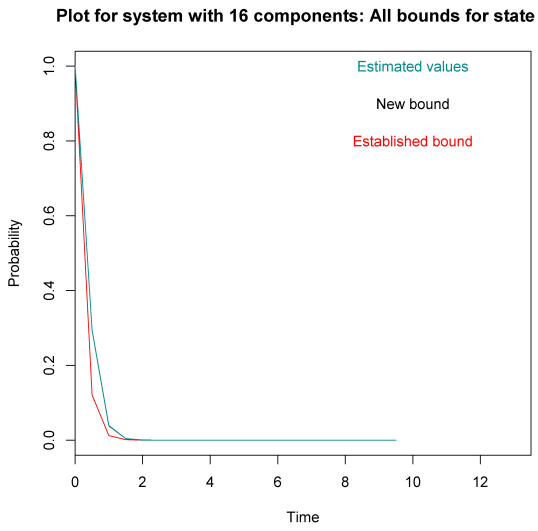

If we compare these new plots for established lower bound, and the new lower bound against the estimated value, it seams like as for the systems with 11 and 13 components that the two bounds look much more like the estimated value.

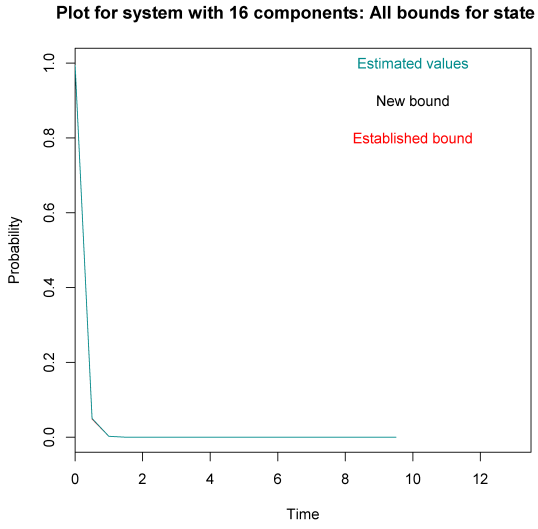
### 4.6.2 Plots one state at a time

Let us plot all the bounds at one state at a time to better see the differences.





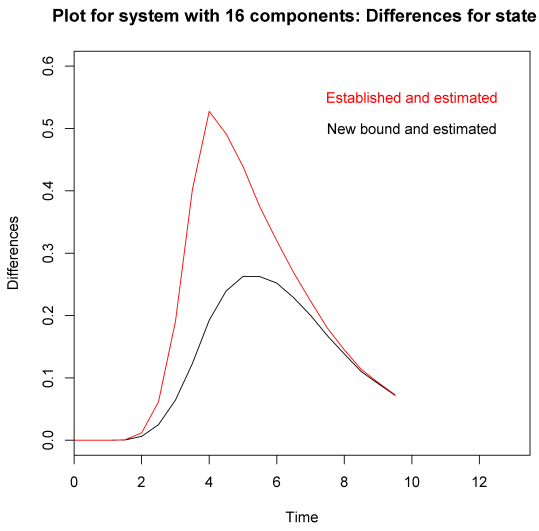



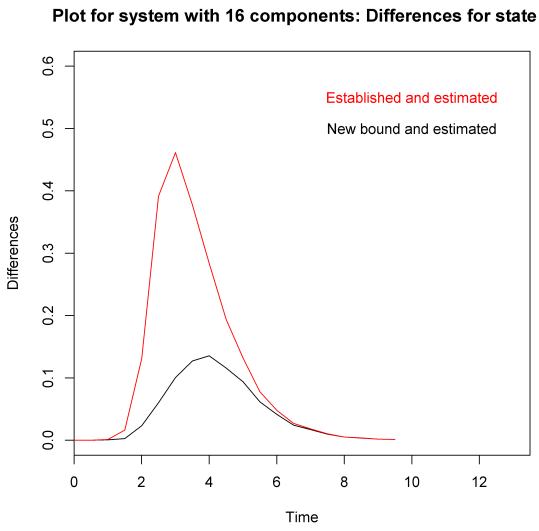



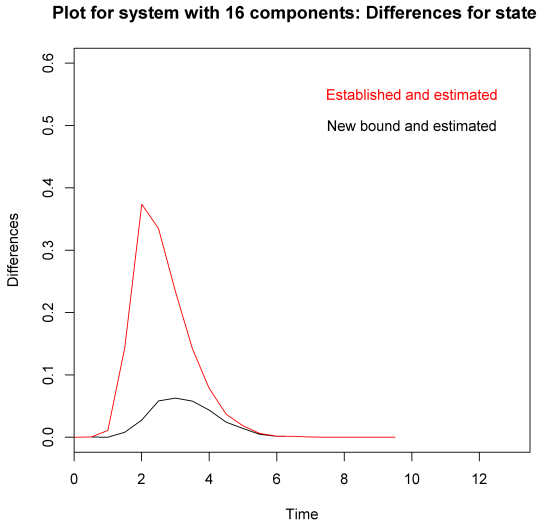

The bounds are now much closer to he estimated value. The new lower bound does here also look closer to the estimated value than what the established lower bound does.

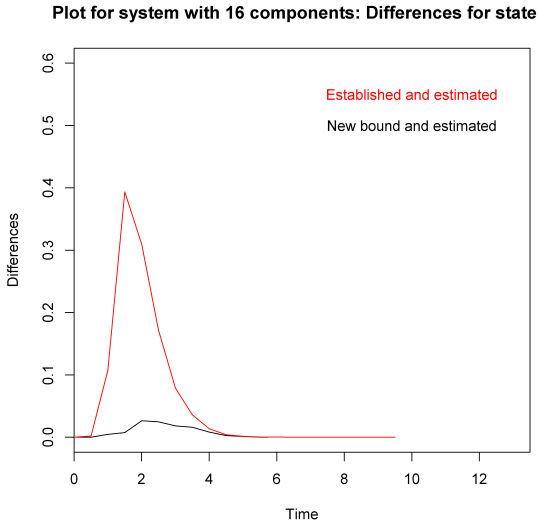
### 4.6.3 Plots of the differences

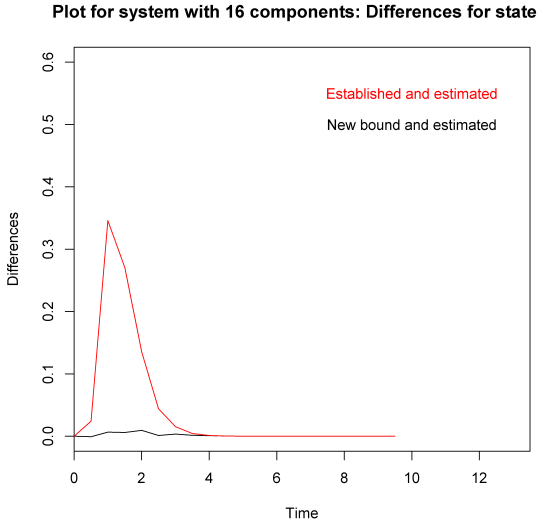
The differences are found by taking the value for the estimated value and subtract it by the value to one of the other bounds. The closer the plot is to zero, the closer is the bound to the estimated value. Let us plot the difference between the estimated value and the two others.

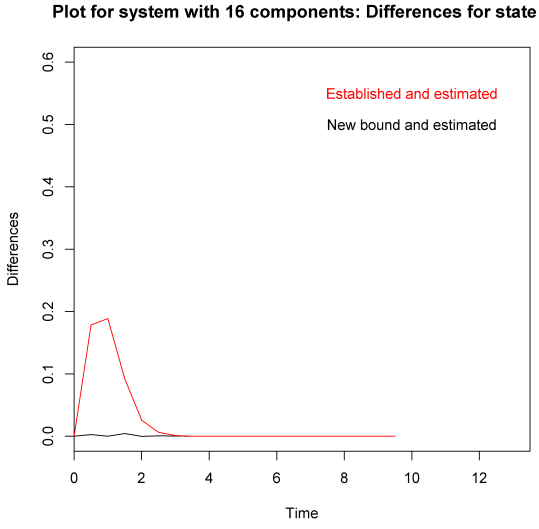

Plot for system with 16 components: Differences for state 1
Established and estimated
New bound and estimated
Differences
Time

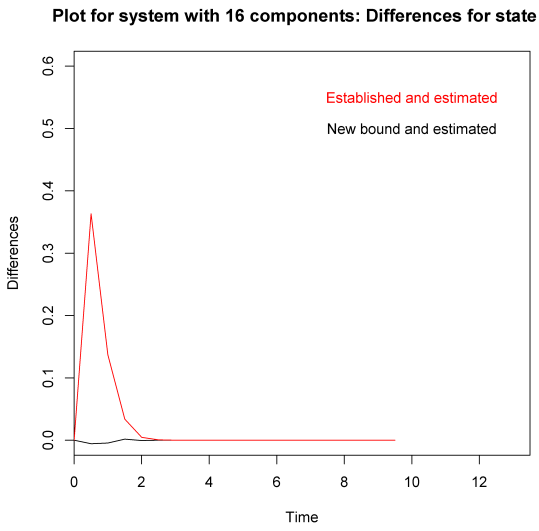

Plot for system with 16 components: Differences for state 2
Established and estimated
New bound and estimated
Differences
Time

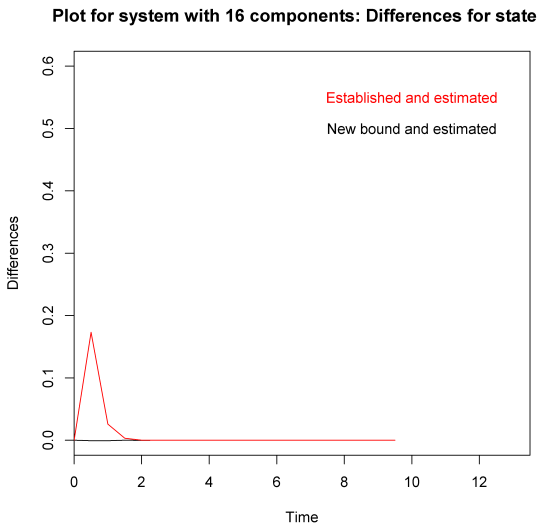

Plot for system with 16 components: Differences for state 3
Established and estimated
New bound and estimated
Differences
Time

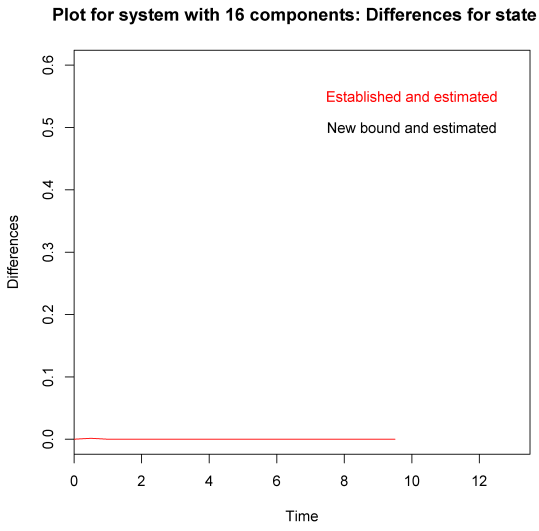

Plot for system with 16 components: Differences for state 4
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 16 components: Differences for state 5
Established and estimated
New bound and estimated
Differences
Time


Plot for system with 16 components: Differences for state 6
Established and estimated
New bound and estimated
Differences
Time



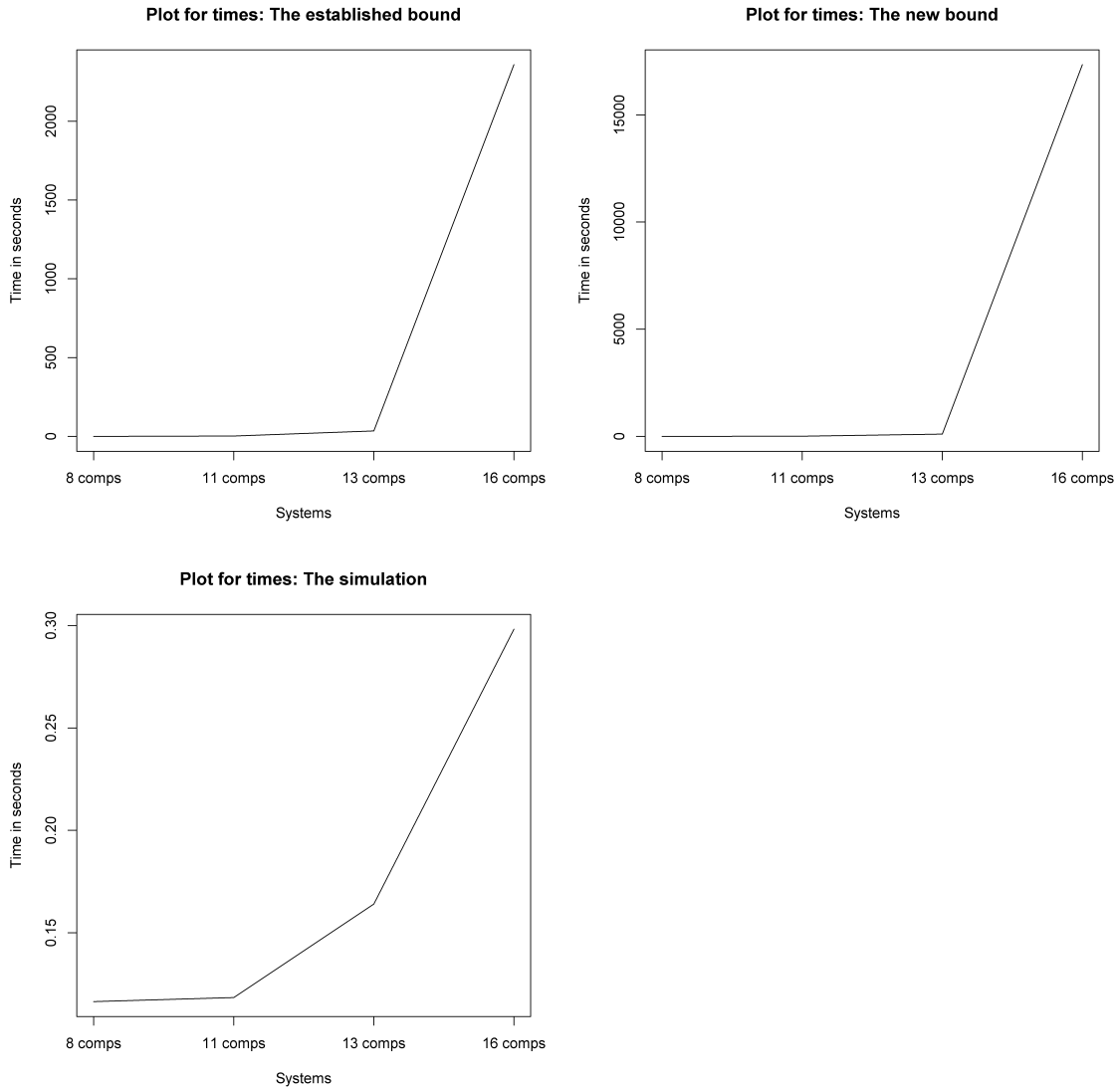




### 4.6.4 Summary

The new bound gets even faster similar to the estimated value and it seams like the new bound is equal to the estimated value after state 4. As time passes, the values of both bounds will be more similar and after some time be equal to the estimated value. The value of the new bound will faster be more similar, but also gradually be equal to the estimated value. The established bound is slower to get similar to the estimated value. It seams like the established lower bound is equal to the estimated value at state 9.

## 4.7 Times

It is also interesting to see how long time the different bounds use and how long the simulation is for the different systems. We will therefore now show three plots:




We here find the time for the established bound by adding the time it takes to find the minimal path and cut sets to the time it takes to calculate the established bound. The new bounds time is simply found by taking the time it uses to do the calculation. With simulation does it here mean the discrete event simulation. The new bound uses much more time than what the established bound does, in seconds does the new bound use 14980.09 more time than the established bound. This is approximately 4 hours and 10 minutes. This comparison may be a bit unfair since it is possible that it exists a better way to calculate the new lower bound or a better algorithm that makes the calculation time much smaller.

### 4.8 Conclusion

From the theory we would expect that the estimated value has a higher value than the two bounds. We would also expect state 0 to have a higher value than state 9. If we look at the different plots that have been shown we can see that this is the case. Let us sum up pro's and con's about the different bounds. It seems like the new bound is better in all cases. And the new bound gets faster similar to the estimated value than what the established lower bound does. It looks like that the lower bound gets better when there are more components in the system. So the best bound without thinking about calculation time is the new lower bound. The only cons about the new bound is the calculation time. The established lower bound does as well improve when having more components in a serial-connection, but it is'nt as good as the lower bound. But the established bound is faster to calculate and this is a really good thing about this bound.

## 5 Further-studies

One of the things that I would like to keep studying and do more research on is to find a more optimal way to calculate the new lower bound. I would also like to check if there is a similar way to find the minimal cut vectors for systems that are not a flow network.

# 6 Full computer code

In this section you can look at the computer code.

## 6.1 Main class

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Map;
import java.util.Random;
import java.util.Iterator;
import java.util.Set;

import no.uio.stochutils.Formula;
import no.uio.stochutils.FormulaParser;
import sim.DefaultSimulator;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

class Main {
        /**This class is the starting class and start it all**/
        private static Double[] state;
        private static List<Component> components;
        static Phi phi;
        static Calculate calc;
        static double simTime = 0;
        static int numSims = 10000;

        public static void main(String[] args) {
                /**Unmark the one you want to run**/
                //start_for_example_4();
                //startAll();
        }

        private static void start_for_example_4() {
                setStateSet4();
                components = getComponents_example_4(state);

                set_Phi_and_cutSets_example_4();

                set_p_for_example_4();
                set_q_for_example_4();


                calc.calculate_new_lower_bound();
                calculate_ls();
                calculate_established_bound();

                print_all_to_documents();
        }


        private static void setStateSet4() {
                double[] s = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0 };
                state = new Double[6];
                for (int i = 0; i < s.length; i++) {
                        state[i] = s[i];
                }
        }

```

```
60          private static List<Component> getComponents_example_4(Double[]
                state2) {
61
62                  Component x1 = new Component(state2, "X1", 0);
63                  Component x2 = new Component(state2, "X2", 1);
64
65                  x1.setFirst(true);
66                  x2.setFirst(true);
67
68                  x1.goesTo(new ArrayList<Component>(Arrays.asList(x2)));
69                  x2.goesTo(new ArrayList<Component>());
70                  return new ArrayList<Component>(Arrays.asList(x1, x2));
71          }
72
73          private static void set_Phi_and_cutSets_example_4() {
74                  phi = new Phi();
75                  phi.changeFlow(true);
76                  phi.setComponents(components);
77
78                  set_sets_for_example_4();
79
80                  find_vector();
81          }
82
83          private static void set_sets_for_example_4() {
84                  HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
                        Integer, Cut_or_path_set>();
85                  Cut_or_path_set cut1 = new Cut_or_path_set();
86                  Cut_or_path_set cut2 = new Cut_or_path_set();
87                  List<Component> cut1list = new ArrayList<Component>(Arrays.
                        asList(components.get(0), components.get(1)));
88
89                  cut1.add_component_vector(cut1list, 0, phi);
90                  cutSets.put(0, cut1);
91                  phi.set_cut_sets(cutSets);
92          }
93
94          private static void set_p_for_example_4() {
95                  calc = new Calculate(numSims, components, phi.get_cut_sets
                        (), phi);
96                  List<double[]> pi_Components = new ArrayList<double[]>();
97                  double[] pi = { 1,(double) 0.95, (double) 0.9, (double)
                        0.85, (double) 0.8, (double) 0.75 };
98                  pi_Components.add(pi);
99                  pi_Components.add(pi);
100                 calc.pi_Components = pi_Components;
101         }
102
103         private static void set_q_for_example_4() {
104                 List<double[]> qi_Components = new ArrayList<double[]>();
105                 double[] qi = { (double) 0.05,(double) 0.05, (double)
                        0.05,(double) 0.05, (double) 0.05, (double) 0.75 };
106                 qi_Components.add(qi);
107                 qi_Components.add(qi);
108                 calc.ri_Components = qi_Components;
109         }
110
111         private static void print_all_to_documents() {
112                 File file_Lj = new File("Lj.txt");
113
114                 File file_l_doubleStar = new File("l_doubleStar.txt");
115
116                 File file_l_apostrophe = new File("l_apostrophe.txt");
117
118                 File file_min_cut = new File("min_cut.txt");
119
120                 File file_min_path = new File("min_path.txt");
121
```

```
122                 try {
123                         FileWriter fw_Lj = new FileWriter(file_Lj, false);
124                         PrintWriter pw_Lj = new PrintWriter(fw_Lj);
125                         pw_Lj.append("0,1,2,3,4,5,6,7,8,9,10" + "\r\n");
126
127                         FileWriter fw_l_doubleStar = new FileWriter(
                                file_l_doubleStar , false);
128                         PrintWriter pw_l_doubleStar = new PrintWriter(
                                fw_l_doubleStar);
129                         pw_l_doubleStar.append("1,2,3,4,5,6,7,8,9,10" + "\r
                                \n");
130
131                         FileWriter fw_l_apostrophe = new FileWriter(
                                file_l_apostrophe, false);
132                         PrintWriter pw_l_apostrophe = new PrintWriter(
                                fw_l_apostrophe);
133                         pw_l_apostrophe.append("0,1,2,3,4,5,6,7,8,9,10" + "
                                \r\n");
134
135                         FileWriter fw_min_cut= new FileWriter(file_min_cut,
                                 false);
136                         PrintWriter pw_min_cut = new PrintWriter(fw_min_cut
                                );
137
138                         FileWriter fw_min_path= new FileWriter(
                                file_min_path, false);
139                         PrintWriter pw_min_path = new PrintWriter(
                                fw_min_path);
140
141                         double[] l_doubleStar = calc.get_l_doubleStar();
142                         double[] l_apostrophe = calc.get_l_apostrophe();
143                         double[] Lj = calc.get_new_lower_bound();
144                         HashMap<Double, List<
                                Minimal_path_or_maximal_cut_vector>> min_cut =
                                phi.get_maximal_cut_vectors();
145                         HashMap<Double, List<
                                Minimal_path_or_maximal_cut_vector>> min_path =
                                 phi.get_minimal_path_vectors();
146
147                         go_through_all_min_cut_or_path_and_write_out(
                                pw_min_cut, min_cut, "Cut ");
148                         go_through_all_min_cut_or_path_and_write_out(
                                pw_min_path, min_path, "Path ");
149
150
151                         String string_Lj = Arrays.toString(Lj);
152                         string_Lj = string_Lj.replaceAll("[\\] ]", "").
                                replaceAll("\\[", "");
153                         pw_Lj.append(string_Lj + "\r\n");
154
155                         String string_l_doubleStar = Arrays.toString(
                                l_doubleStar);
156                         string_l_doubleStar = string_l_doubleStar.
                                replaceAll("[\\] ]", "").replaceAll("\\[", "");
157                         pw_l_doubleStar.append(string_l_doubleStar + "\r\n"
                                );
158
159                         String string_l_apostrophe = Arrays.toString(
                                l_apostrophe);
160                         string_l_apostrophe = string_l_apostrophe.
                                replaceAll("[\\] ]", "").replaceAll("\\[", "");
161                         pw_l_apostrophe.append(string_l_apostrophe + "\r\n"
                                );
162
163
164                         pw_Lj.close();
165                         pw_l_doubleStar.close();
166                         pw_l_apostrophe.close();
```

```
167                         pw_min_cut.close();
168                         pw_min_path.close();
169                 } catch (IOException e) {
170                         // TODO Auto-generated catch block
171                         e.printStackTrace();
172                 }
173         }
174
175         private static void go_through_all_min_cut_or_path_and_write_out(
                PrintWriter pw_min_cut_or_path, HashMap<Double,List<
                Minimal_path_or_maximal_cut_vector>> min_cut_or_path, String
                path_or_cut) {
176                 List<Double> system_state = phi.get_system_states();
177                 for(int j = 1; j < system_state.size(); j++){
178                         double current_state = system_state.get(j);
179                         pw_min_cut_or_path.append("State: " + current_state
                                + "\r\n" + "\r\n");
180                         List<Minimal_path_or_maximal_cut_vector> listVect =
                                 (List<Minimal_path_or_maximal_cut_vector>)
                                min_cut_or_path.get(current_state);
181                         int i  = 1;
182                         for (Minimal_path_or_maximal_cut_vector ve :
                                listVect) {
183                                 pw_min_cut_or_path.append(path_or_cut + i +
                                        "\r\n");
184                                 i++;
185                                 HashMap<Component, Double> vector = ve.
                                        returnVector();
186                                 Set set2 = vector.entrySet();
187                                 Iterator iterator2 = set2.iterator();
188                                 while (iterator2.hasNext()) {
189                                         Map.Entry mentry2 = (Map.Entry)
                                                iterator2.next();
190                                         Component comp = (Component)
                                                mentry2.getKey();
191                                         pw_min_cut_or_path.append(comp.
                                                getName() + "  is in state: " +
                                                 mentry2.getValue() + "\r\n");
192                                 }
193                                 pw_min_cut_or_path.append("\r\n");
194                         }
195                         pw_min_cut_or_path.append("\r\n" + "\r\n");
196                 }
197         }
198
199         private static void startAll() {
200                 double maxSimTime = 10;
201                 File file_Lj = new File("Lj.txt");
202
203                 File file_p = new File("p.txt");
204
205                 File file_B = new File("B.txt");
206
207                 File file_t_s = new File("time_sim.txt");
208
209                 File file_t_B = new File("time_B.txt");
210
211                 File file_t_Lj = new File("time_Lj.txt");
212
213                 File file_t_maxmin = new File("time_maxmin.txt");
214
215
216                 /* Make new files to store values in */
217                 try {
218                         FileWriter fw_Lj = new FileWriter(file_Lj, false);
219                         PrintWriter pw_Lj = new PrintWriter(fw_Lj);
220                         pw_Lj.append("t,0,1,2,3,4,5,6,7,8,9" + "\r\n");
221
```

```
222                         FileWriter fw_p = new FileWriter(file_p, false);
223                         PrintWriter pw_p = new PrintWriter(fw_p);
224                         pw_p.append("t,0,1,2,3,4,5,6,7,8,9" + "\r\n");
225
226                         FileWriter fw_B = new FileWriter(file_B, false);
227                         PrintWriter pw_B = new PrintWriter(fw_B);
228                         pw_B.append("t,1,2,3,4,5,6,7,8,9" + "\r\n");
229
230                         FileWriter fw_t_s = new FileWriter(file_t_s, false)
                                ;
231                         PrintWriter pw_t_s = new PrintWriter(fw_t_s);
232
233                         FileWriter fw_t_B = new FileWriter(file_t_B, false)
                                ;
234                         PrintWriter pw_t_B = new PrintWriter(fw_t_B);
235
236                         FileWriter fw_t_Lj = new FileWriter(file_t_Lj,
                                false);
237                         PrintWriter pw_t_Lj = new PrintWriter(fw_t_Lj);
238
239                         FileWriter fw_t_maxmin = new FileWriter(
                                file_t_maxmin, false);
240                         PrintWriter pw_t_maxmin = new PrintWriter(
                                fw_t_maxmin);
241
242                         /* Unmark the one you want to simulate */
243                         //set_Starting_Values_For_8comps(pw_t_maxmin);
244                         //set_starting_values_for_13_comps(pw_t_maxmin);
245                         //set_starting_values_for_16_comps(pw_t_maxmin);
246                         //set_starting_values_for_system_11_comps(
                                pw_t_maxmin);
247
248                         /* Set simulation time and start simulation */
249                         for (simTime = 0.001; simTime <= maxSimTime;
                                simTime = simTime + 0.5) {
250                             start_simulation_and_calculation(pw_Lj,
                                    pw_p, pw_B, pw_t_s, pw_t_B, pw_t_Lj);
251                         }
252
253                         pw_Lj.close();
254                         pw_p.close();
255                         pw_B.close();
256                         pw_t_s.close();
257                         pw_t_B.close();
258                         pw_t_Lj.close();
259                 } catch (IOException e) {
260                         // TODO Auto-generated catch block
261                         e.printStackTrace();
262                 }
263         }
264
265         private static void set_Starting_Values_For_8comps(PrintWriter
                pw_t_maxmin) {
266                 set_state_set();
267                 components = get_components_system_8_comps(state);
268                 // set_Phi_and_let_it_find_cuts();
269                 set_Phi_and_cut_sets(8, pw_t_maxmin);
270         }
271
272         private static void set_state_set() {
273                 double[] s = { 0.0, 1.0, 2.0, 3.0 };
274                 state = new Double[4];
275                 for (int i = 0; i < s.length; i++) {
276                         state[i] = s[i];
277                 }
278         }
279
```

```
280         private static List<Component> get_components_system_8_comps(Double
                [] s) {
281             /*Makes new components and give them a name*/
282             Component x1 = new Component(s, "X1", 0);
283             Component x2 = new Component(s, "X2", 1);
284             Component x3 = new Component(s, "X3", 2);
285             Component x4 = new Component(s, "X4", 3);
286             Component x5 = new Component(s, "X5", 4);
287             Component x6 = new Component(s, "X6", 5);
288             Component x7 = new Component(s, "X7", 6);
289             Component x8 = new Component(s, "X8", 7);
290 
291             /*Set the first components*/
292             x1.setFirst(true);
293             x2.setFirst(true);
294             x3.setFirst(true);
295 
296             /*Tell the different components which components it goes to
                    next*/
297             x1.goesTo(new ArrayList<Component>(Arrays.asList(x6, x4)));
298             x2.goesTo(new ArrayList<Component>(Arrays.asList(x7)));
299             x3.goesTo(new ArrayList<Component>(Arrays.asList(x8, x5)));
300             x4.goesTo(new ArrayList<Component>(Arrays.asList(x7)));
301             x5.goesTo(new ArrayList<Component>(Arrays.asList(x7)));
302             return new ArrayList<Component>(Arrays.asList(x1, x2, x3,
                    x4, x5, x6, x7, x8));
303         }
304 
305         private static void set_starting_values_for_system_11_comps(
                PrintWriter pw_t_maxmin) {
306             set_state_set();
307             components = get_components_system_11_comps(state);
308             // set_Phi_and_let_it_find_cuts();
309             set_Phi_and_cut_sets(11, pw_t_maxmin);
310         }
311 
312         private static List<Component> get_components_system_11_comps(
                Double[] s) {
313             /*Makes new components and give them a name*/
314             Component x1 = new Component(s, "X1", 0);
315             Component x2 = new Component(s, "X2", 1);
316             Component x3 = new Component(s, "X3", 2);
317             Component x4 = new Component(s, "X4", 3);
318             Component x5 = new Component(s, "X5", 4);
319             Component x6 = new Component(s, "X6", 5);
320             Component x7 = new Component(s, "X7", 6);
321             Component x8 = new Component(s, "X8", 7);
322             Component x9 = new Component(s, "X9", 8);
323             Component x10 = new Component(s, "X10", 9);
324             Component x11 = new Component(s, "X11", 10);
325 
326             /*Set the first components*/
327             x1.setFirst(true);
328             x2.setFirst(true);
329             x3.setFirst(true);
330 
331             /*Tell the different components which components it goes to
                    next*/
332             x1.goesTo(new ArrayList<Component>(Arrays.asList(x4)));
333             x2.goesTo(new ArrayList<Component>(Arrays.asList(x5)));
334             x3.goesTo(new ArrayList<Component>(Arrays.asList(x6)));
335             x4.goesTo(new ArrayList<Component>(Arrays.asList(x7, x9)));
336             x5.goesTo(new ArrayList<Component>(Arrays.asList(x10)));
337             x6.goesTo(new ArrayList<Component>(Arrays.asList(x8, x11)))
                    ;
338             x7.goesTo(new ArrayList<Component>(Arrays.asList(x10)));
339             x8.goesTo(new ArrayList<Component>(Arrays.asList(x10)));
340             x9.goesTo(new ArrayList<Component>(Arrays.asList()));
```

```
341                 x10.goesTo(new ArrayList<Component>(Arrays.asList()));
342                 x11.goesTo(new ArrayList<Component>(Arrays.asList()));
343                 return new ArrayList<Component>(Arrays.asList(x1, x2, x3,
                        x4, x5, x6, x7, x8, x9, x10, x11));
344         }
345
346         private static void set_starting_values_for_13_comps(PrintWriter
                pw_t_maxmin) {
347                 set_state_set();
348                 components = get_components_system_13_comps(state);
349                 // set_Phi_and_let_it_find_cuts();
350                 set_Phi_and_cut_sets(13,pw_t_maxmin);
351         }
352
353         private static List<Component> get_components_system_13_comps(
                Double[] s) {
354                 /*Makes new components and give them a name*/
355                 Component x1 = new Component(s, "X1", 0);
356                 Component x2 = new Component(s, "X2", 1);
357                 Component x3 = new Component(s, "X3", 2);
358                 Component x4 = new Component(s, "X4", 3);
359                 Component x5 = new Component(s, "X5", 4);
360                 Component x6 = new Component(s, "X6", 5);
361                 Component x7 = new Component(s, "X7", 6);
362                 Component x8 = new Component(s, "X8", 7);
363                 Component x9 = new Component(s, "X9", 8);
364                 Component x10 = new Component(s, "X10", 9);
365                 Component x11 = new Component(s, "X11", 10);
366                 Component x12 = new Component(s, "X12", 11);
367                 Component x13 = new Component(s, "X13", 12);
368
369                 /*Set the first components*/
370                 x1.setFirst(true);
371                 x2.setFirst(true);
372                 x3.setFirst(true);
373
374                 /*Tell the different components which components it goes to
                        next*/
375                 x1.goesTo(new ArrayList<Component>(Arrays.asList(x4)));
376                 x2.goesTo(new ArrayList<Component>(Arrays.asList(x5)));
377                 x3.goesTo(new ArrayList<Component>(Arrays.asList(x6)));
378                 x4.goesTo(new ArrayList<Component>(Arrays.asList(x7, x11)))
                        ;
379                 x5.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
380                 x6.goesTo(new ArrayList<Component>(Arrays.asList(x10, x13))
                        );
381                 x7.goesTo(new ArrayList<Component>(Arrays.asList(x8)));
382                 x8.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
383                 x9.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
384                 x10.goesTo(new ArrayList<Component>(Arrays.asList(x9)));
385                 x11.goesTo(new ArrayList<Component>(Arrays.asList()));
386                 x12.goesTo(new ArrayList<Component>(Arrays.asList()));
387                 x13.goesTo(new ArrayList<Component>(Arrays.asList()));
388
389                 return new ArrayList<Component>(Arrays.asList(x1, x2, x3,
                        x4, x5, x6, x7, x8, x9, x10, x11, x12, x13));
390         }
391
392         private static void set_starting_values_for_16_comps(PrintWriter
                pw_t_maxmin) {
393                 set_state_set();
394                 components = get_components_system_16_comps(state);
395                 // set_Phi_and_let_it_find_cuts();
396                 set_Phi_and_cut_sets(16,pw_t_maxmin);
397         }
398
399         private static List<Component> get_components_system_16_comps(
                Double[] s) {
```

```
400                 /*Makes new components and give them a name*/
401                 Component x1 = new Component(s, "X1", 0);
402                 Component x2 = new Component(s, "X2", 1);
403                 Component x3 = new Component(s, "X3", 2);
404                 Component x4 = new Component(s, "X4", 3);
405                 Component x5 = new Component(s, "X5", 4);
406                 Component x6 = new Component(s, "X6", 5);
407                 Component x7 = new Component(s, "X7", 6);
408                 Component x8 = new Component(s, "X8", 7);
409                 Component x9 = new Component(s, "X9", 8);
410                 Component x10 = new Component(s, "X10", 9);
411                 Component x11 = new Component(s, "X11", 10);
412                 Component x12 = new Component(s, "X12", 11);
413                 Component x13 = new Component(s, "X13", 12);
414                 Component x14 = new Component(s, "X14", 13);
415                 Component x15 = new Component(s, "X15", 14);
416                 Component x16 = new Component(s, "X16", 15);
417
418                 /*Set the first components*/
419                 x1.setFirst(true);
420                 x2.setFirst(true);
421                 x3.setFirst(true);
422
423                 /*Tell the different components which components it goes to
                        next*/
424                 x1.goesTo(new ArrayList<Component>(Arrays.asList(x4)));
425                 x2.goesTo(new ArrayList<Component>(Arrays.asList(x5)));
426                 x3.goesTo(new ArrayList<Component>(Arrays.asList(x6)));
427                 x4.goesTo(new ArrayList<Component>(Arrays.asList(x7, x11)))
                        ;
428                 x5.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
429                 x6.goesTo(new ArrayList<Component>(Arrays.asList(x10, x13))
                        );
430                 x7.goesTo(new ArrayList<Component>(Arrays.asList(x8)));
431                 x8.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
432                 x9.goesTo(new ArrayList<Component>(Arrays.asList(x12)));
433                 x10.goesTo(new ArrayList<Component>(Arrays.asList(x9)));
434                 x11.goesTo(new ArrayList<Component>(Arrays.asList(x14)));
435                 x12.goesTo(new ArrayList<Component>(Arrays.asList(x15)));
436                 x13.goesTo(new ArrayList<Component>(Arrays.asList(x16)));
437                 x14.goesTo(new ArrayList<Component>(Arrays.asList()));
438                 x15.goesTo(new ArrayList<Component>(Arrays.asList()));
439                 x16.goesTo(new ArrayList<Component>(Arrays.asList()));
440
441                 return new ArrayList<Component>(
442                                 Arrays.asList(x1, x2, x3, x4, x5, x6, x7,
                                        x8, x9, x10, x11, x12, x13, x14, x15,
                                        x16));
443         }
444
445         /*Function that can find minimal cut sets for system without cyclic
                parts and set the phi*/
446         private static void set_Phi_and_let_it_find_cuts() {
447                 phi = new Phi();
448                 phi.changeFlow(true);
449                 phi.setComponents(components);
450                 find_cutSets();
451                 find_vector();
452         }
453
454         private static void find_cutSets() {
455                 phi.findCuts();
456         }
457
458         private static void find_vector() {
459                 phi.find_vectors();
460         }
461
```

```
462         private static void set_Phi_and_cut_sets(int i, PrintWriter
                pw_t_maxmin) {
463                 phi = new Phi();
464                 phi.setComponents(components);
465                 if (i == 8) {
466                         set_cut_sets_for_8();
467                 } else if (i == 11) {
468                         set_cut_sets_for_system_11_comps();
469                 } else if (i == 13) {
470                         set_cut_sets_for_system_13_comps();
471                 } else if (i == 16) {
472                         set_cut_sets_for_system_16();
473                 }
474
475                 long time = System.nanoTime();
476                 find_vector();
477                 time = System.nanoTime()-time;
478
479                 pw_t_maxmin.append(time + "\r\n");
480                 pw_t_maxmin.close();
481         }
482
483         private static void set_cut_sets_for_8() {
484                 /*Makes cut sets and a list to store them all */
485                 HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
                        Integer, Cut_or_path_set>();
486                 Cut_or_path_set cut1 = new Cut_or_path_set();
487                 Cut_or_path_set cut2 = new Cut_or_path_set();
488                 Cut_or_path_set cut3 = new Cut_or_path_set();
489                 Cut_or_path_set cut4 = new Cut_or_path_set();
490                 Cut_or_path_set cut5 = new Cut_or_path_set();
491                 Cut_or_path_set cut6 = new Cut_or_path_set();
492                 Cut_or_path_set cut7 = new Cut_or_path_set();
493                 Cut_or_path_set cut8 = new Cut_or_path_set();
494
495                 /*Add the components in the right cut sets*/
496                 List<Component> cut1list = new ArrayList<Component>(
497                                 Arrays.asList(components.get(0), components
                                        .get(1), components.get(2)));
498                 List<Component> cut2list = new ArrayList<Component>(
499                                 Arrays.asList(components.get(1), components
                                        .get(2), components.get(3), components.
                                        get(5)));
500                 List<Component> cut3list = new ArrayList<Component>(
501                                 Arrays.asList(components.get(0), components
                                        .get(2), components.get(6)));
502                 List<Component> cut4list = new ArrayList<Component>(
503                                 Arrays.asList(components.get(0), components
                                        .get(1), components.get(4), components.
                                        get(7)));
504                 List<Component> cut5list = new ArrayList<Component>(
505                                 Arrays.asList(components.get(2), components
                                        .get(5), components.get(6)));
506                 List<Component> cut6list = new ArrayList<Component>(Arrays.
                        asList(components.get(1), components.get(3),
507                                 components.get(4), components.get(5),
                                        components.get(7)));
508                 List<Component> cut7list = new ArrayList<Component>(
509                                 Arrays.asList(components.get(0), components
                                        .get(6), components.get(7)));
510                 List<Component> cut8list = new ArrayList<Component>(
511                                 Arrays.asList(components.get(5), components
                                        .get(6), components.get(7)));
512
513                 /*Add cut list to the cut and add the cut sets to the list
                        that contains all the sets*/
514                 cut1.add_component_vector(cut1list, 0, phi);
515                 cut2.add_component_vector(cut2list, 1, phi);
```

```
516                 cut3.add_component_vector(cut3list, 2, phi);
517                 cut4.add_component_vector(cut4list, 3, phi);
518                 cut5.add_component_vector(cut5list, 4, phi);
519                 cut6.add_component_vector(cut6list, 5, phi);
520                 cut7.add_component_vector(cut7list, 6, phi);
521                 cut8.add_component_vector(cut8list, 7, phi);
522                 cutSets.put(0, cut1);
523                 cutSets.put(1, cut2);
524                 cutSets.put(2, cut3);
525                 cutSets.put(3, cut4);
526                 cutSets.put(4, cut5);
527                 cutSets.put(5, cut6);
528                 cutSets.put(6, cut7);
529                 cutSets.put(7, cut8);
530                 phi.set_cut_sets(cutSets);
531         }
532
533         private static void set_cut_sets_for_system_11_comps() {
534                 /*Makes cut sets and a list to store them all */
535                 HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
                        Integer, Cut_or_path_set>();
536                 Cut_or_path_set cut1 = new Cut_or_path_set();
537                 Cut_or_path_set cut2 = new Cut_or_path_set();
538                 Cut_or_path_set cut3 = new Cut_or_path_set();
539                 Cut_or_path_set cut4 = new Cut_or_path_set();
540                 Cut_or_path_set cut5 = new Cut_or_path_set();
541                 Cut_or_path_set cut6 = new Cut_or_path_set();
542                 Cut_or_path_set cut7 = new Cut_or_path_set();
543                 Cut_or_path_set cut8 = new Cut_or_path_set();
544
545                 /*Add the components in the right cut sets*/
546                 List<Component> cut1list = new ArrayList<Component>(
547                                 Arrays.asList(components.get(0), components
                                        .get(1), components.get(2)));
548                 List<Component> cut2list = new ArrayList<Component>(
549                                 Arrays.asList(components.get(1), components
                                        .get(2), components.get(6), components.
                                        get(8)));
550                 List<Component> cut3list = new ArrayList<Component>(
551                                 Arrays.asList(components.get(0), components
                                        .get(2), components.get(9)));
552                 List<Component> cut4list = new ArrayList<Component>(
553                                 Arrays.asList(components.get(0), components
                                        .get(1), components.get(7), components.
                                        get(10)));
554                 List<Component> cut5list = new ArrayList<Component>(
555                                 Arrays.asList(components.get(2), components
                                        .get(8), components.get(9)));
556                 List<Component> cut6list = new ArrayList<Component>(Arrays.
                        asList(components.get(1), components.get(6),
557                                 components.get(7), components.get(8),
                                        components.get(10)));
558                 List<Component> cut7list = new ArrayList<Component>(
559                                 Arrays.asList(components.get(0), components
                                        .get(9), components.get(10)));
560                 List<Component> cut8list = new ArrayList<Component>(
561                                 Arrays.asList(components.get(8), components
                                        .get(9), components.get(10)));
562
563                 List<List<Component>> cuts = new ArrayList<List<Component
                        >>(
564                                 Arrays.asList(cut1list, cut2list, cut3list,
                                         cut4list, cut5list, cut6list, cut7list
                                        , cut8list));
565
566                 /*Add cut list to the cut and add the cut sets to the list
                        that contains all the sets*/
567                 cut1.add_component_vector(cut1list, 0, phi);
```

```
568                 cut2.add_component_vector(cut2list, 1, phi);
569                 cut3.add_component_vector(cut3list, 2, phi);
570                 cut4.add_component_vector(cut4list, 3, phi);
571                 cut5.add_component_vector(cut5list, 4, phi);
572                 cut6.add_component_vector(cut6list, 5, phi);
573                 cut7.add_component_vector(cut7list, 6, phi);
574                 cut8.add_component_vector(cut8list, 7, phi);
575                 cutSets.put(0, cut1);
576                 cutSets.put(1, cut2);
577                 cutSets.put(2, cut3);
578                 cutSets.put(3, cut4);
579                 cutSets.put(4, cut5);
580                 cutSets.put(5, cut6);
581                 cutSets.put(6, cut7);
582                 cutSets.put(7, cut8);
583
584                 /*Make two list of components. One that are going to be
                        switched to another component
585                  * and the other list contains the components that it is
                         going to be switched to */
586                 List<Component> change_comps = new ArrayList<Component>(
587                                 Arrays.asList(components.get(3), components
                                        .get(4), components.get(5)));
588                 List<Component> change_from_comps = new ArrayList<Component
                        >(
589                                 Arrays.asList(components.get(0), components
                                        .get(1), components.get(2)));
590
591                 /*Switch the components and add the new cut set to the list
                         with all the cut sets*/
592                 for (int i = 0; i < change_comps.size(); i++) {
593                         List<List<Component>> cuts2 = new ArrayList<List<
                                Component>>();
594                         for (List<Component> comps : cuts) {
595                                 List<Component> cutlist = new ArrayList<
                                        Component>();
596                                 boolean changes = false;
597                                 for (Component comp : comps) {
598                                         if (comp == change_from_comps.get(i
                                                )) {
599                                                 cutlist.add(change_comps.
                                                        get(i));
600                                                 changes = true;
601                                         } else {
602                                                 cutlist.add(comp);
603                                         }
604                                 }
605                                 if (changes) {
606                                         cuts2.add(cutlist);
607                                         Cut_or_path_set cut = new
                                                Cut_or_path_set();
608                                         int n = cutSets.size();
609                                         cut.add_component_vector(cutlist, n
                                                , phi);
610                                         cutSets.put(n, cut);
611                                 }
612                         }
613                         cuts.addAll(cuts2);
614                 }
615
616                 phi.set_cut_sets(cutSets);
617         }
618
619         private static void set_cut_sets_for_system_13_comps() {
620                 /*Makes cut sets and a list to store them all */
621                 HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
                        Integer, Cut_or_path_set>();
622                 Cut_or_path_set cut1 = new Cut_or_path_set();
```

```
623                 Cut_or_path_set cut2 = new Cut_or_path_set();
624                 Cut_or_path_set cut3 = new Cut_or_path_set();
625                 Cut_or_path_set cut4 = new Cut_or_path_set();
626                 Cut_or_path_set cut5 = new Cut_or_path_set();
627                 Cut_or_path_set cut6 = new Cut_or_path_set();
628                 Cut_or_path_set cut7 = new Cut_or_path_set();
629                 Cut_or_path_set cut8 = new Cut_or_path_set();
630
631                 /*Add the components in the right cut sets*/
632                 List<Component> cut1list = new ArrayList<Component>(
633                         Arrays.asList(components.get(0), components
                                .get(1), components.get(2)));
634                 List<Component> cut2list = new ArrayList<Component>(
635                         Arrays.asList(components.get(1), components
                                .get(2), components.get(6), components.
                                get(10)));
636                 List<Component> cut3list = new ArrayList<Component>(
637                         Arrays.asList(components.get(0), components
                                .get(2), components.get(11)));
638                 List<Component> cut4list = new ArrayList<Component>(
639                         Arrays.asList(components.get(0), components
                                .get(1), components.get(9), components.
                                get(12)));
640                 List<Component> cut5list = new ArrayList<Component>(
641                         Arrays.asList(components.get(2), components
                                .get(10), components.get(11)));
642                 List<Component> cut6list = new ArrayList<Component>(Arrays.
                    asList(components.get(1), components.get(6),
643                         components.get(9), components.get(10),
                                components.get(12)));
644                 List<Component> cut7list = new ArrayList<Component>(
645                         Arrays.asList(components.get(0), components
                                .get(11), components.get(12)));
646                 List<Component> cut8list = new ArrayList<Component>(
647                         Arrays.asList(components.get(10),
                                components.get(11), components.get(12))
                                );
648
649                 List<List<Component>> cuts = new ArrayList<List<Component
                    >>(
650                         Arrays.asList(cut1list, cut2list, cut3list,
                                 cut4list, cut5list, cut6list, cut7list
                                , cut8list));
651
652                 /*Add cut list to the cut and add the cut sets to the list
                        that contains all the sets*/
653                 cut1.add_component_vector(cut1list, 0, phi);
654                 cut2.add_component_vector(cut2list, 1, phi);
655                 cut3.add_component_vector(cut3list, 2, phi);
656                 cut4.add_component_vector(cut4list, 3, phi);
657                 cut5.add_component_vector(cut5list, 4, phi);
658                 cut6.add_component_vector(cut6list, 5, phi);
659                 cut7.add_component_vector(cut7list, 6, phi);
660                 cut8.add_component_vector(cut8list, 7, phi);
661                 cutSets.put(0, cut1);
662                 cutSets.put(1, cut2);
663                 cutSets.put(2, cut3);
664                 cutSets.put(3, cut4);
665                 cutSets.put(4, cut5);
666                 cutSets.put(5, cut6);
667                 cutSets.put(6, cut7);
668                 cutSets.put(7, cut8);
669
670                 /*Make two list of components. One that are going to be
                        switched to another component
671                  * and the other list contains the components that it is
                         going to be switched to */
```

```
672             List<Component> change_comps = new ArrayList<Component>(
                    Arrays.asList(components.get(3), components.get(4),
673                         components.get(5), components.get(7),
                             components.get(8)));
674             List<Component> change_from_comps = new ArrayList<Component
                    >(Arrays.asList(components.get(0), components.get(1),
675                         components.get(2), components.get(6),
                             components.get(9)));
676
677             /*Switch the components and add the new cut set to the list
                     with all the cut sets*/
678             for (int i = 0; i < change_comps.size(); i++) {
679                 List<List<Component>> cuts2 = new ArrayList<List<
                        Component>>();
680                 for (List<Component> comps : cuts) {
681                     List<Component> cutlist = new ArrayList<
                            Component>();
682                     boolean changes = false;
683                     for (Component comp : comps) {
684                         if (comp == change_from_comps.get(i
                                )) {
685                             cutlist.add(change_comps.
                                    get(i));
686                             changes = true;
687                         } else {
688                             cutlist.add(comp);
689                         }
690                     }
691                     if (changes) {
692                         cuts2.add(cutlist);
693                         Cut_or_path_set cut = new
                                Cut_or_path_set();
694                         int n = cutSets.size();
695                         cut.add_component_vector(cutlist, n
                                , phi);
696                         cutSets.put(n, cut);
697                     }
698                 }
699                 cuts.addAll(cuts2);
700             }
701
702             phi.set_cut_sets(cutSets);
703         }
704
705         private static void set_cut_sets_for_system_16() {
706             /*Makes cut sets and a list to store them all */
707             HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
                    Integer, Cut_or_path_set>();
708             Cut_or_path_set cut1 = new Cut_or_path_set();
709             Cut_or_path_set cut2 = new Cut_or_path_set();
710             Cut_or_path_set cut3 = new Cut_or_path_set();
711             Cut_or_path_set cut4 = new Cut_or_path_set();
712             Cut_or_path_set cut5 = new Cut_or_path_set();
713             Cut_or_path_set cut6 = new Cut_or_path_set();
714             Cut_or_path_set cut7 = new Cut_or_path_set();
715             Cut_or_path_set cut8 = new Cut_or_path_set();
716
717             List<Component> cut1list = new ArrayList<Component>(
718                     Arrays.asList(components.get(0), components
                            .get(1), components.get(2)));
719             List<Component> cut2list = new ArrayList<Component>(
720                     Arrays.asList(components.get(1), components
                            .get(2), components.get(6), components.
                            get(10)));
721             List<Component> cut3list = new ArrayList<Component>(
722                     Arrays.asList(components.get(0), components
                            .get(2), components.get(11)));
723             List<Component> cut4list = new ArrayList<Component>(
```

```
724                         Arrays.asList(components.get(0), components
                                .get(1), components.get(9), components.
                                get(12)));
725                 List<Component> cut5list = new ArrayList<Component>(
726                         Arrays.asList(components.get(2), components
                                .get(10), components.get(11)));
727                 List<Component> cut6list = new ArrayList<Component>(Arrays.
                        asList(components.get(1), components.get(6),
728                         components.get(9), components.get(10),
                                components.get(12)));
729                 List<Component> cut7list = new ArrayList<Component>(
730                         Arrays.asList(components.get(0), components
                                .get(11), components.get(12)));
731                 List<Component> cut8list = new ArrayList<Component>(
732                         Arrays.asList(components.get(10),
                                components.get(11), components.get(12))
                                );
733
734                 List<List<Component>> cuts = new ArrayList<List<Component
                        >>(
735                         Arrays.asList(cut1list, cut2list, cut3list,
                                cut4list, cut5list, cut6list, cut7list
                                , cut8list));
736
737                 /*Add cut list to the cut and add the cut sets to the list
                        that contains all the sets*/
738                 cut1.add_component_vector(cut1list, 0, phi);
739                 cut2.add_component_vector(cut2list, 1, phi);
740                 cut3.add_component_vector(cut3list, 2, phi);
741                 cut4.add_component_vector(cut4list, 3, phi);
742                 cut5.add_component_vector(cut5list, 4, phi);
743                 cut6.add_component_vector(cut6list, 5, phi);
744                 cut7.add_component_vector(cut7list, 6, phi);
745                 cut8.add_component_vector(cut8list, 7, phi);
746                 cutSets.put(0, cut1);
747                 cutSets.put(1, cut2);
748                 cutSets.put(2, cut3);
749                 cutSets.put(3, cut4);
750                 cutSets.put(4, cut5);
751                 cutSets.put(5, cut6);
752                 cutSets.put(6, cut7);
753                 cutSets.put(7, cut8);
754
755                 /*Make two list of components. One that are going to be
                        switched to another component
756                  * and the other list contains the components that it is
                         going to be switched to */
757                 List<Component> change_comps = new ArrayList<Component>(
758                         Arrays.asList(components.get(3), components
                                .get(4), components.get(5), components.
                                get(7),
759                                         components.get(8),
                                                components.get(13),
                                                components.get(14),
                                                components.get(15)));
760                 List<Component> change_from_comps = new ArrayList<Component
                        >(
761                         Arrays.asList(components.get(0), components
                                .get(1), components.get(2), components.
                                get(6),
762                                         components.get(9),
                                                components.get(10),
                                                components.get(11),
                                                components.get(12)));
763
764                 /*Switch the components and add the new cut set to the list
                         with all the cut sets*/
765                 for (int i = 0; i < change_comps.size(); i++) {
```

```
766                         List<List<Component>> cuts2 = new ArrayList<List<
                                Component>>();
767                         for (List<Component> comps : cuts) {
768                                 List<Component> cutlist = new ArrayList<
                                        Component>();
769                                 boolean changes = false;
770                                 for (Component comp : comps) {
771                                         if (comp == change_from_comps.get(i
                                                )) {
772                                                 cutlist.add(change_comps.
                                                        get(i));
773                                                 changes = true;
774                                         } else {
775                                                 cutlist.add(comp);
776                                         }
777                                 }
778                                 if (changes) {
779                                         cuts2.add(cutlist);
780                                         Cut_or_path_set cut = new
                                                Cut_or_path_set();
781                                         int n = cutSets.size();
782                                         cut.add_component_vector(cutlist, n
                                                , phi);
783                                         cutSets.put(n, cut);
784                                 }
785                         }
786                         cuts.addAll(cuts2);
787                 }
788
789                 phi.set_cut_sets(cutSets);
790         }
791
792         private static void start_simulation_and_calculation(PrintWriter
                pw_Lj, PrintWriter pw_p, PrintWriter pw_B, PrintWriter pw_t_s,
                PrintWriter pw_t_B, PrintWriter pw_t_Lj) {
793                 startSimulation(pw_t_s);
794                 calculate_r();
795                 calculate_p();
796
797                 /*Calculate the different bounds and write them to the
                        right files*/
798                 calculate_new_lower_bound(pw_t_Lj);
799                 String string_Lj = Arrays.toString(calc.get_new_lower_bound
                        ());
800                 string_Lj = string_Lj.replaceAll("[\\] ]", "").replaceAll("
                        \\[", "");
801                 pw_Lj.append(simTime + "," + string_Lj + "\r\n");
802                 long time = 0;
803
804                 time = time + calculate_ls();
805                 String string_p = Arrays.toString(calc.get_p());
806                 string_p = string_p.replaceAll("[\\] ]", "").replaceAll("
                        \\[", "");
807                 pw_p.append(simTime + "," + string_p + "\r\n");
808
809                 time = time + calculate_established_bound();
810                 String string_B = Arrays.toString(calc.
                        get_established_lower_bound());
811                 string_B = string_B.replaceAll("[\\] ]", "").replaceAll("
                        \\[", "");
812                 pw_B.append(simTime + "," + string_B + "\r\n");
813
814                 pw_t_B.append(time+ "\r\n");
815
816         }
817
818         private static void startSimulation(PrintWriter pw_t_s) {
```

```
819                 DefaultSimulator simulator = new DefaultSimulator(numSims,
                        simTime);
820
821                 /*Get the state distributions for the differnt states*/
822                 HashMap<Double, Formula> stateDistributions = new HashMap<
                        Double, Formula>();
823                 // "GAMMA3(1.0;0.5)"
824                 // "EXPON(2.0)"
825                 String st1 = "EXPON(2.0)";
826                 String st2 = "EXPON(2.0)";
827                 String st3 = "EXPON(2.0)";
828                 String st4 = "EXPON(2.0)";
829
830                 FormulaParser.SET_MAKERS(FormulaParser.CONSTANT_MAKER +
                        FormulaParser.STANDARD_MAKER);
831
832                 Formula dist_1_0 = FormulaParser.strToFormula(st1);
833                 Formula dist_1_1 = FormulaParser.strToFormula(st2);
834                 Formula dist_1_2 = FormulaParser.strToFormula(st3);
835                 Formula dist_1_3 = FormulaParser.strToFormula(st4);
836
837                 stateDistributions.put(0.0, dist_1_0);
838                 stateDistributions.put(1.0, dist_1_1);
839                 stateDistributions.put(2.0, dist_1_2);
840                 stateDistributions.put(3.0, dist_1_3);
841
842                 /*Set the state distributions for the components*/
843                 for (Component comp : components) {
844                         comp.set_state_distributions(stateDistributions);
845                         comp.set_start_state_and_simTime(state[state.length
                                - 1], simTime);
846                         simulator.addEventHandler(comp);
847                 }
848
849                 long time = System.nanoTime();
850                 simulator.startSimulator();
851                 time = System.nanoTime()-time;
852
853                 pw_t_s.append(time + "\r\n");
854         }
855
856         private static void calculate_p() {
857                 calc.findAllp();
858         }
859
860         private static void calculate_r() {
861                 calc = new Calculate(numSims, components, phi.get_cut_sets
                        (), phi);
862                 calc.calculate_r_s();
863         }
864
865         private static long calculate_ls() {
866                 long time = System.nanoTime();
867                 calc.findAll_l();
868                 time = System.nanoTime()-time;
869                 return time;
870         }
871
872         private static long calculate_established_bound() {
873                 long time = System.nanoTime();
874                 calc.find_established_lower_bound();
875                 time = System.nanoTime()-time;
876
877                 return time;
878         }
879
880         private static void calculate_new_lower_bound(PrintWriter pw_t_Lj)
                {
```

```
881                     long time = System.nanoTime();
882                     calc.calculate_new_lower_bound();
883                     time = System.nanoTime()-time;
884 
885                     pw_t_Lj.append(time + "\r\n");
886             }
887 
888     }
```

## 6.2 Find minimal cut sets class

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.List;
4
5  public class Find_minimal_cut_sets {
6          private HashMap<Integer, Cut_or_path_set> cutSets = new HashMap<
               Integer, Cut_or_path_set>();
7          private HashMap<Component, List<List<Component>>> path_sets = new
               HashMap<Component, List<List<Component>>>();
8          private List<Component> firstComponents;
9          private List<Component> path = new ArrayList<Component>();
10         boolean flow;
11
12         Find_minimal_cut_sets(List<Component> firstComponents, boolean flow
               , Phi system) {
13                 this.firstComponents = firstComponents;
14                 this.flow = flow;
15                 loop_path_set();
16                 minimalCut(system);
17         }
18
19         public HashMap<Integer, Cut_or_path_set> returnCutSets() {
20                 return cutSets;
21         }
22
23         private void minimalCut(Phi system) {
24                 HashMap<Component, HashMap<Integer, List<Component>>> cuts
                       = find_cuts_for_each_branch();
25                 find_cuts(cuts, 0, new ArrayList<Component>(), true, system
                       );
26         }
27
28         private void add_minimal_cut(List<Component> list, Phi system) {
29                 Cut_or_path_set cutSet = new Cut_or_path_set();
30                 cutSet.add_component_vector(list, cutSets.size(), system);
31                 cutSets.put(cutSet.getIndex(), cutSet);
32         }
33
34         private void find_cuts(HashMap<Component, HashMap<Integer, List<
               Component>>> all_cuts_sorted_after_first_comp, int compNumber,
35                         List<Component> cantHave, boolean first, Phi system
                               ) {
36                 HashMap<Integer, List<Component>> cutList =
                       all_cuts_sorted_after_first_comp.get(firstComponents.
                       get(compNumber));
37                 List<Component> first_comps_has_only_one_next =
                       check_and_find_which_first_comps_thta_has_only_one_next
                       ();
38                 find_cut_lists(all_cuts_sorted_after_first_comp, cutList,
                       compNumber, cantHave, 0, first, system);
39         }
40
41         private void find_cut_lists(HashMap<Component, HashMap<Integer,
               List<Component>>> all_cuts_sorted_after_first_comp,
42                         HashMap<Integer, List<Component>> cutList, int
                               compNumber, List<Component> cant_have_in_path,
                               int lengthLooping,
43                         boolean first, Phi system) {
44                 boolean added_last_element_in_lists_to_cant_have_in_path =
                       false;
45                 int lengthLists = 0;
46                 boolean not_entered_loop = true;
47                 boolean first_comp_has_only_one_next = false;
48                 Component nextComp = null;
49                 if(firstComponents.get(compNumber).
                       get_list_of_which_components_this_goes_to().size() ==
```

```
                    1){
50                      first_comp_has_only_one_next = true;
51                      nextComp = firstComponents.get(compNumber).
                            get_list_of_which_components_this_goes_to().get
                            (0);
52                  }
53                  for (List<Component> cut : cutList.values()) {
54                      lengthLists++;
55                      if (!check_if_list_is_in_path(cut, path)) {
56                              for (Component comp : cut) {
57                                      if(first_comp_has_only_one_next &&
                                            comp != nextComp){
58                                          cant_have_in_path =
                                                add_to_cantHaveInPath(
                                                cutList.get(cutList.
                                                size() - 2),
                                                cant_have_in_path);
59                                      }
60                                      if (!
                                            check_if_component_is_in_cantHaveInPath
                                            (comp, cant_have_in_path)) {
61                                          not_entered_loop = false;
62                                          if (compNumber == 0 &&
                                                first && cutList.get(
                                                cutList.size() - 1).
                                                containsAll(cut)) {
63                                              cant_have_in_path =
                                                    new ArrayList<
                                                    Component>();
64                                          }
65                                          cant_have_in_path =
                                                add_to_cantHaveInPath(
                                                cut, cant_have_in_path)
                                                ;
66                                          if (cutList.size() > 1 &&
                                                first) {
67                                              cant_have_in_path =

                                                    add_to_cantHaveInPath
                                                    (cutList.get(
                                                    cutList.size()
                                                    - 1),
                                                    cant_have_in_path
                                                    );
68                                          }
69                                          if (cutList.size() < 3 &&
                                                cutList.get(cutList.
                                                size() - 1).containsAll
                                                (cut)) {
70                                              cant_have_in_path =

                                                    add_to_cantHaveInPath
                                                    (cutList.get(0)
                                                    ,
                                                    cant_have_in_path
                                                    );
71                                          }
72                                          path.add(comp);
73                                          boolean enteredIf = false;
74                                          if (!cutList.get(cutList.
                                                size() - 2).containsAll
                                                (cut)
75                                                      && !cutList
                                                            .get(
                                                            cutList
                                                            .size()
                                                             - 1).
                                                            containsAll
```

```
                                    (cut))
                                    {
76                                      List<Component>
                                            sendIn = new
                                            ArrayList<
                                            Component>();
77                                      sendIn.addAll(
                                            cant_have_in_path
                                            );
78                                      find_cut_lists(
                                            all_cuts_sorted_after_first_comp
                                            , cutList,
                                            compNumber,
                                            sendIn,
                                            lengthLooping +
                                             1, false,
                                            system);
79                                      enteredIf = true;
80                                      cant_have_in_path =

                                            add_to_cantHaveInPath
                                            (cutList.get(
                                            lengthLists),
                                            cant_have_in_path
                                            );
81                                  }
82                                  if (compNumber < (
                                        firstComponents.size()
                                        - 1) && !enteredIf) {
83                                      List<Component>
                                            sendIn = new
                                            ArrayList<
                                            Component>();
84                                      sendIn.addAll(
                                            cant_have_in_path
                                            );
85                                      find_cuts(
                                            all_cuts_sorted_after_first_comp
                                            , compNumber +
                                            1, sendIn, true
                                            , system);
86                                      enteredIf = true;
87                                  }
88                                  if (!enteredIf) {
89                                      List<Component>
                                            setCut = new
                                            ArrayList<
                                            Component>();
90                                      setCut.addAll(path)
                                            ;
91                                      add_minimal_cut(
                                            setCut, system)
                                            ;
92                                  }
93
94                                  path.remove(comp);
95                                  cant_have_in_path =
                                        remove_from_cantHaveInPath
                                        (cut, cant_have_in_path
                                        );
96                                  if (cutList.size() > 1 &&
                                        first) {
97                                      cant_have_in_path =

                                            remove_from_cantHaveInPath
                                            (cutList.get(
                                            cutList.size()
                                            - 1),
```

```
                                                        cant_have_in_path
                                                        );
98                                                  }
99                                              }
100                                             if(first_comp_has_only_one_next){
101                                                     cant_have_in_path =
                                                            remove_from_cantHaveInPath
                                                            (cutList.get(cutList.
                                                            size() - 2),
                                                            cant_have_in_path);
102                                             }
103                                         }
104                         } else if (cut != cutList.get(cutList.size() - 1)
                                && not_entered_loop) {
105                             cant_have_in_path = add_to_cantHaveInPath(
                                    cutList.get(cutList.size() - 1),
                                    cant_have_in_path);
106                             added_last_element_in_lists_to_cant_have_in_path
                                    = true;
107                         }
108
109                 }
110                 if(first_comp_has_only_one_next){
111                         cant_have_in_path = add_to_cantHaveInPath(cutList.
                                get(cutList.size() - 2), cant_have_in_path);
112                 }
113                 if (added_last_element_in_lists_to_cant_have_in_path &&
                        lengthLooping == 0){
114                     if (compNumber < (firstComponents.size() - 1)) {
115                         List<Component> sendIn = new ArrayList<
                                Component>();
116                         sendIn.addAll(cant_have_in_path);
117                         find_cuts(all_cuts_sorted_after_first_comp,
                                compNumber + 1, sendIn, true, system);
118                     }
119                 } else if (not_entered_loop) {
120                         List<Component> setCut = new ArrayList<Component>()
                                ;
121                         setCut.addAll(path);
122                         add_minimal_cut(setCut, system);
123                 }
124         }
125
126         private boolean check_if_list_is_in_path(List<Component> list, List
                <Component> statePaths) {
127             for (Component comp : list) {
128                 if (statePaths.contains(comp)) {
129                     return true;
130                 }
131             }
132             return false;
133         }
134
135         private List<Component> remove_from_cantHaveInPath(List<Component>
                list, List<Component> cantHave) {
136             for (Component remove : list) {
137                 cantHave.remove(remove);
138             }
139             return cantHave;
140         }
141
142         private List<Component> add_to_cantHaveInPath(List<Component> list,
                 List<Component> cantHave) {
143             for (Component add : list) {
144                 cantHave.add(add);
145             }
146             return cantHave;
147         }
```

```
148
149         private boolean check_if_component_is_in_cantHaveInPath(Component
                comp, List<Component> list) {
150             for (Component l : list) {
151                 if (l.equals(comp)) {
152                     return true;
153                 }
154             }
155             return false;
156         }
157
158         private HashMap<Component, HashMap<Integer, List<Component>>>
                find_cuts_for_each_branch() {
159             HashMap<Component, HashMap<Integer, List<Component>>> cuts
                    = new HashMap<Component, HashMap<Integer, List<
                    Component>>>();
160             for (Component c : firstComponents) {
161                 List<List<Component>> cut = new ArrayList<List<
                        Component>>();
162                 cut.addAll(path_sets.get(c));
163                 cuts.put(c, split_cut_branches(cut, c.
                        get_comp_number()));
164             }
165             return cuts;
166         }
167
168         private List<Component>
                check_and_find_which_first_comps_thta_has_only_one_next() {
169             List<Component> listOfNextComps = new ArrayList<Component
                    >();
170             for(Component comp:firstComponents){
171                 List<Component> next = comp.
                        get_list_of_which_components_this_goes_to();
172                 if(next.size() == 1){
173                     listOfNextComps.addAll(next);
174                 }
175             }
176             return listOfNextComps;
177         }
178
179         private HashMap<Integer, List<Component>> split_cut_branches(List<
                List<Component>> cut, int firstCompNumber) {
180             HashMap<Integer, List<Component>> cuts = new HashMap<
                    Integer, List<Component>>();
181             int j = 0;
182             List<Component> firstComp = new ArrayList<Component>();
183             List<Component> first_comp_next_only_one = null;
184             int nextCompNumber = -1;
185             if(firstComponents.get(firstCompNumber).
                    get_list_of_which_components_this_goes_to().size() == 1
                     && cut.size() != 1){
186                 first_comp_next_only_one = firstComponents.get(
                        firstCompNumber).
                        get_list_of_which_components_this_goes_to();
187                 nextCompNumber = first_comp_next_only_one.get(0).
                        get_comp_number();
188             }
189             boolean first_comp_not_added = true;
190             for (List<Component> list : cut) {
191                 List<Component> setCuts = new ArrayList<Component
                        >();
192                 for (int i = 0; i < list.size(); i++) {
193                     path.add(list.get(i));
194                     if (!checkIfExists(cuts, path) && list.get(
                            i).get_comp_number() != firstCompNumber
                             && list.get(i).get_comp_number() !=
                            nextCompNumber) {
195                         setCuts.addAll(path);
```

```
196                                 } else if (list.get(i).get_comp_number() ==
                                        firstCompNumber &&
                                    first_comp_not_added) {
197                                     firstComp.addAll(path);
198                                     first_comp_not_added = false;
199                                 }
200                                 path.remove(list.get(i));
201
202                         }
203                         cuts.put(j, setCuts);
204                         j++;
205                 }
206                 if(first_comp_next_only_one != null){
207                         cuts.put(j, first_comp_next_only_one);
208                         j++;
209                 }
210                 cuts.put(j, firstComp);
211                 return cuts;
212         }
213
214         private boolean checkIfExists(HashMap<Integer, List<Component>>
              cuts, List<Component> setCut) {
215                 for (List<Component> list : cuts.values()) {
216                         if (list.contains(setCut)) {
217                                 return true;
218                         }
219                 }
220                 return false;
221         }
222
223         private void loop_path_set() {
224                 if (flow == true) {
225                         for (Component c : firstComponents) {
226                                 List<List<Component>> Path = new ArrayList<
                                        List<Component>>();
227                                 find_path_set_flow(c, c, Path);
228                                 set_minimal_path_information(Path, c);
229                         }
230                 } else {
231                         for (Component c : firstComponents) {
232                                 List<List<Component>> Path = new ArrayList<
                                        List<Component>>();
233                                 set_minimal_path_information(Path, c);
234                         }
235                 }
236         }
237
238         private void set_minimal_path_information(List<List<Component>>
              Path, Component c) {
239                 path_sets.put(c, Path);
240                 for (int i = 0; i < path_sets.get(c).size(); i++) {
241                         for (Component add_info_to_component : path_sets.
                                get(c).get(i)) {
242                                 add_info_to_component.
                                        add_minimal_path_vector_info(c, i);
243                         }
244                 }
245         }
246
247         private void find_path_set_flow(Component first, Component c, List<
              List<Component>> Path) {
248                 path.add(c);
249                 if (c.get_list_of_which_components_this_goes_to() != null)
                        {
250                         for (Component next : c.
                                get_list_of_which_components_this_goes_to()) {
251                                 if (!first.
                                        get_list_of_which_components_this_goes_to
```

```
                                        ().contains(next) || first == c) {
252                                         if (next.
                                                get_list_of_which_components_this_goes_to
                                                () != null && next.
                                                get_list_of_which_components_this_goes_to
                                                ().size() == 1) {
253                                                 Component nextsnext = next.
                                                        get_list_of_which_components_this_goes_to
                                                        ().get(0);
254                                                 if (!path.contains(
                                                        nextsnext)) {
255                                                     find_path_set_flow(
                                                            first, next,
                                                            Path);
256                                                 }
257
258                                             } else {
259                                                 find_path_set_flow(first,
                                                        next, Path);
260                                             }
261                                     }
262                             }
263                     } else {
264                             List<Component> setPath = new ArrayList<Component
                                    >();
265                             setPath.addAll(path);
266                             Path.add(setPath);
267                     }
268                     path.remove(c);
269                     return;
270
271             }
272
273     }
```

## 6.3 Default simulator class

```
1  package sim;
2  /*
       ********************************************************************************

3
4          Type:                   DefaultSimulator
5          Package:                no.uio.simutils
6
7          @author                 arne
8          @version                1.0
9          Description:    Default implementation of the simulator interface
10
11         Copyright (C) 2006 by University of Oslo. All rights reserved.
12
13         Redistribution and use in source and binary forms, with or without
14         modification, are permitted provided that the following conditions
15         are met:
16
17         1. Redistributions of source code must retain the above copyright
18            notice, this list of conditions and the following disclaimer.
19         2. Redistributions in binary form must reproduce the above
               copyright
20            notice, this list of conditions and the following disclaimer in
                  the
21            documentation and/or other materials provided with the
                  distribution.
22
23         THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS''
               AND
24         ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
               THE
25         IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
               PURPOSE
26         ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
               LIABLE
27         FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
               CONSEQUENTIAL
28         DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
               GOODS
29         OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
               INTERRUPTION)
30         HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
                STRICT
31         LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
               ANY WAY
32         OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
                OF
33         SUCH DAMAGE.
34
35 ********************************************************************************
       */
36
37 import java.util.Vector;
38
39 public class DefaultSimulator implements Simulator
40 {
41         private static boolean DISPLAY_ITERATIONS = false;
42         private static int DISPLAY_FREQUENCY = 1;
43
44         public static void SET_DISPLAY_ITERATIONS(boolean flag)
45         {
46                 DISPLAY_ITERATIONS = flag;
47         }
48
49         public static void SET_DISPLAY_FREQUENCY(int freq)
50         {
```

```
51              DISPLAY_FREQUENCY = freq;
52          }
53
54          protected static final double EPS = 1e-10;
55
56          private Vector<SimEventHandler> eventHandlers;
57          protected int numSims;                              // The number of
                simulation runs
58          protected double simTime;                           // The length of
                each simulation
59
60          public DefaultSimulator(int numSims, double simTime)
61          {
62                  eventHandlers = new Vector<SimEventHandler>();
63                  this.numSims = numSims;
64                  this.simTime = simTime;
65          }
66
67          public DefaultSimulator(int capacity, int numSims, double simTime)
68          {
69                  eventHandlers = new Vector<SimEventHandler>(capacity);
70                  this.numSims = numSims;
71                  this.simTime = simTime;
72          }
73
74          public int eventHandlerCount()
75          {
76                  return eventHandlers.size();
77          }
78
79          public void addEventHandler(SimEventHandler eventHandler)
80          {
81                  eventHandlers.add(eventHandler);
82          }
83
84          public SimEventHandler getEventHandlerAt(int index)
85          {
86                  return eventHandlers.elementAt(index);
87          }
88
89          protected void initSimulator()
90          {
91                  // NULL-method
92          }
93
94          protected void initIteration(int iteration)
95          {
96                  if (DISPLAY_ITERATIONS)
97                  {
98                          if (DISPLAY_FREQUENCY <= 1)
99                          {
100                                 System.out.println("Iteration " + iteration
                                        );
101                         }
102                         else
103                         {
104                                 if (iteration % DISPLAY_FREQUENCY == 0)
105                                 {
106                                         System.out.println("Iteration " +
                                                iteration);
107                                 }
108                         }
109                 }
110         }
111
112         public void startSimulator()
113         {
114                 initSimulator();
```

```
115                     int num = eventHandlerCount();
116
117                     for (int i = 1; i <= numSims; i++)
118                     {
119                             initIteration(i);
120
121                             SimEventQueue eventQueue = new SimEventQueue();
122
123                             for (int j = 0; j < num; j++)
124                             {
125                                     getEventHandlerAt(j).prepareToSim(
                                            eventQueue);
126                             }
127
128                             eventQueue.processAllEvents(simTime + EPS);
129                     }
130             }
131
132             public void stopSimulator(boolean displayResults)
133             {
134                     // NULL-method
135             }
136     }
```

## 6.4 Phi class (System class)

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.Iterator;
4  import java.util.List;
5  import java.util.Map;
6  import java.util.Set;
7
8  public class Phi {
9          private List<Component> components = null;
10         private HashMap<Integer,Cut_or_path_set> cutSets = new HashMap<
               Integer,Cut_or_path_set>();
11         private List<Component> first_components = new ArrayList<Component
               >();
12         private Find_minimal_cut_sets find_cut_sets;
13         private Find_minimal_path_and_maximal_cut_vectors
               find_minimal_path_and_maximal_cut_vectors;
14         private HashMap<Double,List<Minimal_path_or_maximal_cut_vector>>
               maximal_cut_vectors;
15         private HashMap<Double,List<Minimal_path_or_maximal_cut_vector>>
               minimal_path_vectors;
16         private List<Double> systemStates;
17         private double systemState;
18         private double system_minimum_state;
19         private Cut_or_path_set cut_with_lowest_state;
20         /** Is it a flow network? Yes = true and no = false. Standard is
               false. */
21         private Boolean flow = false;
22         private Integer[] stored_state_values;
23         private Integer[] stored_minimum_values;
24         private int cuts_that_has_stored_its_value = 1;
25         private double maxState;
26
27
28         public void addComponent(Component c) {
29                 this.components.add(c);
30                 setFirst();
31         }
32
33
34         public void setComponents(List<Component> c) {
35                 this.components = c;
36                 setFirst();
37         }
38
39         public void set_system_states(List<Double> states){
40                 systemStates = states;
41         }
42
43         private void initialize_stored_state_values_and_minimum_values() {
44                 stored_state_values = new Integer[systemStates.size()];
45                 stored_minimum_values = new Integer[systemStates.size()];
46                 for (int i = 0; i < systemStates.size(); i++) {
47                         stored_state_values[i] = 0;
48                         stored_minimum_values[i] = 0;
49                 }
50
51         }
52
53         public void find_vectors() {
54                 find_minimal_path_and_maximal_cut_vectors =  new
                       Find_minimal_path_and_maximal_cut_vectors(cutSets,
                       components);
55                 systemStates = find_minimal_path_and_maximal_cut_vectors.
                       get_system_states();
56                 find_max_state();
57                 maximal_cut_vectors =
```

```
                    find_minimal_path_and_maximal_cut_vectors.
                    get_maximal_cuts();
58                  minimal_path_vectors =
                    find_minimal_path_and_maximal_cut_vectors.
                    get_minimal_paths();
59                  initialize_stored_state_values_and_minimum_values();
60          }
61
62          private void find_max_state() {
63                  maxState = systemStates.get(systemStates.size()-1);
64                  systemState = maxState;
65                  system_minimum_state = systemState;
66          }
67
68          public void findCuts(){
69                  find_cut_sets = new Find_minimal_cut_sets(first_components,
                    flow, this);
70                  cutSets = find_cut_sets.returnCutSets();
71          }
72
73          public void addComponents(List<Component> c) {
74                  for (Component comp : c) {
75                          this.components.add(comp);
76                  }
77                  setFirst();
78                  findCuts();
79          }
80
81          public void removeComponent(Component c) {
82                  components.remove(c);
83                  first_components.remove(c);
84                  findCuts();
85          }
86
87          public void changeFlow(Boolean f) {
88                  flow = f;
89          }
90
91
92          public void setFirst() {
93                  for (Component c : components) {
94                          if (c.is_this_one_first()) {
95                                  first_components.add(c);
96                          }
97                  }
98          }
99
100         public void find_system_state(){
101                 systemState = maxState;
102                 Set set = cutSets.entrySet();
103                 Iterator iterator = set.iterator();
104                 while (iterator.hasNext()) {
105                         double tempSystemState = 0;
106                         Map.Entry mentry = (Map.Entry) iterator.next();
107                         Cut_or_path_set cut = (Cut_or_path_set) mentry.
                            getValue();
108                         if(cut.get_current_set_state() < systemState){
109                                 systemState = cut.get_current_set_state();
110                                 cut_with_lowest_state = cut;
111                         }
112                 }
113         }
114
115         public HashMap<Double,List<Minimal_path_or_maximal_cut_vector>>
            get_maximal_cut_vectors(){
116                 return maximal_cut_vectors;
117         }
118
```

```
119         public HashMap<Double,List<Minimal_path_or_maximal_cut_vector>>
                get_minimal_path_vectors(){
120             return minimal_path_vectors;
121         }
122 
123         public List<Double> get_system_states(){
124             return systemStates;
125         }
126 
127         public Double get_system_state(){
128             return systemState;
129         }
130 
131         public void find_new_current_state(double subsetState,
                Cut_or_path_set cut) {
132             if(subsetState < systemState){
133                 systemState = subsetState;
134                 cut_with_lowest_state = cut;
135             }
136             else if(systemState < subsetState && cut_with_lowest_state
                    == cut){
137                 find_system_state();
138             }
139             if(systemState < system_minimum_state){
140                 system_minimum_state = systemState;
141             }
142         }
143 
144         public void set_cut_sets(HashMap<Integer, Cut_or_path_set> cutSets)
                {
145             this.cutSets = cutSets;
146         }
147 
148         public HashMap<Integer, Cut_or_path_set> get_cut_sets(){
149             return cutSets;
150         }
151 
152 
153         public void tell_phi_to_store_state_value() {
154             if (cuts_that_has_stored_its_value == cutSets.size()) {
155                 store_state_value();
156                 store_minimum_value();
157                 systemState = maxState;
158                 system_minimum_state = systemState;
159                 cut_with_lowest_state = null;
160                 cuts_that_has_stored_its_value = 1;
161             } else {
162                 cuts_that_has_stored_its_value++;
163             }
164         }
165 
166         public void store_state_value() {
167             for (int i = 0; i <= maxState; i++) {
168                 if (i == systemState) {
169                     stored_state_values[i] =
                            stored_state_values[i] + 1;
170                     break;
171                 }
172             }
173         }
174 
175         private void store_minimum_value() {
176             for (int i = 0; i < systemStates.size(); i++) {
177                 if (systemStates.get(i) == system_minimum_state) {
178                     stored_minimum_values[i] =
                            stored_minimum_values[i] + 1;
179                     break;
180                 }
```

```
181                     }
182             }
183
184
185             public Integer[] get_Stored_Minimum_Value_Counts(){
186                     return stored_minimum_values;
187             }
188
189
190             public Integer[] get_Stored_State_Values(){
191                     return stored_state_values;
192             }
193
194     }
```

## 6.5 Component class

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.List;
4
5  import no.uio.stochutils.Formula;
6  import sim.SimEvent;
7  import sim.SimEventHandler;
8  import sim.SimEventQueue;
9
10 public class Component  implements SimEventHandler{
11         /** Set the states of the component. Example: {0,1,2,3} */
12         private Double[] states;
13         /** Distribution for each state */
14         private HashMap<Double,Formula> state_distributions = new HashMap<
               Double,Formula>();
15         //Set the starting state for this Component
16         private double start_state;
17         /** Set name of the component. Example: X1 */
18         private String name;
19         /** Set witch components this one is passing to */
20         private List<Component> goesTo = null;
21         /** Is this the first component? */
22         private boolean first = false;
23         /** Which component number is this? */
24         private int compNumber;
25         private double current_state;
26         private double minimum_state;
27         /** MinimalPaths this component is in */
28         private HashMap<Component, Integer>
               minimal_path_vectors_this_component_is_in = new HashMap<
               Component, Integer>();
29         // private HashMap<Integer, List<Component>>
               minimalCutsThisComponentIsIn =
30         // new HashMap<Integer,List<Component>>();
31         private List<Cut_or_path_set> cutSets = new ArrayList<
               Cut_or_path_set>();
32         private Integer[] stored_last_state_values;
33         private Integer[] stored_minimum_values;
34         private double simTime;
35
36
37         Component(Double[] s, String n, int number) {
38                 states = s;
39                 initialize_stored_last_state_values_and_minimum_values();
40                 name = n;
41                 compNumber = number;
42         }
43
44         private void initialize_stored_last_state_values_and_minimum_values
               () {
45                 stored_last_state_values = new Integer[states.length];
46                 stored_minimum_values = new Integer[states.length];
47                 for(int i = 0; i < states.length;i++){
48                         stored_last_state_values[i] = 0 ;
49                         stored_minimum_values[i] = 0;
50                 }
51
52         }
53
54         public void goesTo(List<Component> c) {
55                 goesTo = c;
56         }
57
58         public boolean is_this_one_first() {
59                 return first;
60         }
```

```
61
62          public void set_current_state(double newState){
63                  tell_cut_sets_of_the_change(newState-current_state);
64                  current_state = newState;
65                  if(current_state < minimum_state){
66                          minimum_state = current_state;
67                  }
68          }
69
70
71          private void tell_cut_sets_of_the_change(double stateDiff) {
72                  for(Cut_or_path_set cut:cutSets){
73                          cut.find_new_current_state(stateDiff);
74                  }
75          }
76
77          /**
78           * Here you can set that the component is the first component the
                 system
79           * starts with.
80           */
81          public void setFirst(boolean f) {
82                  first = f;
83          }
84
85          public void add_minimal_path_vector_info(Component c, int i) {
86                  minimal_path_vectors_this_component_is_in.put(c, i);
87          }
88
89          public HashMap<Component, Integer>
                get_minimal_path_vectors_this_component_is_in() {
90                  return minimal_path_vectors_this_component_is_in;
91          }
92
93
94          public void addCutSet(Cut_or_path_set cutSet) {
95                  cutSets.add(cutSet);
96          }
97
98          public List<Cut_or_path_set> get_cut_sets_this_component_is_in() {
99                  return cutSets;
100         }
101
102         public Double[] getStates(){
103                 return states;
104         }
105
106         public Integer get_comp_number(){
107                 return compNumber;
108         }
109
110         public List<Component> get_list_of_which_components_this_goes_to(){
111                 return goesTo;
112         }
113
114         public Double get_current_state(){
115                 return current_state;
116         }
117
118         public void set_start_state_and_simTime(double startState, double
                simTime){
119                 this.start_state = startState;
120                 this.simTime = simTime;
121         }
122
123         public void set_state_distributions(HashMap<Double,Formula>
                stateDistributions){
124                 this.state_distributions = stateDistributions;
```

```
125         }
126
127         public String getName(){
128                 return name;
129         }
130
131
132         @Override
133         public void prepareToSim(SimEventQueue eventQueue) {
134                 current_state = start_state;
135                 minimum_state = current_state;
136                 prepare_simEvent_and_add_to_queue(eventQueue);
137         }
138
139         @Override
140         public void handleSimEvent(SimEvent event, SimEventQueue eventQueue
                ) {
141                 int eventCode =  event.getEventCode();
142                 if(eventCode == 1){
143                         updateState();
144                         prepare_simEvent_and_add_to_queue(eventQueue);
145                 }
146                 else{
147                         store_last_state_value();
148                         store_minimum_value();
149                         tell_cuts_to_store_values();
150                         current_state = start_state;
151                         minimum_state = current_state;
152                         }
153         }
154
155
156         private void tell_cuts_to_store_values() {
157                 for(Cut_or_path_set cut:cutSets){
158                         cut.tell_set_to_store_values();
159                 }
160         }
161
162         private void store_last_state_value() {
163                 for(int i = 0; i <= states.length-1; i++ ){
164                         if(states[i] == current_state){
165                                 stored_last_state_values[i] =
                                        stored_last_state_values[i] +1;
166                                 break;
167                         }
168                 }
169         }
170
171         private void store_minimum_value() {
172                 for(int i = 0; i <= states.length-1; i++ ){
173                         if(states[i] == minimum_state){
174                                 stored_minimum_values[i] =
                                        stored_minimum_values[i] + 1;
175                                 break;
176                         }
177                 }
178         }
179
180         private void prepare_simEvent_and_add_to_queue(SimEventQueue
                eventQueue){
181                 Formula f = state_distributions.get(current_state);
182                 double eventTime = f.getFormulaValue();
183                 double next_event_time = eventTime + eventQueue.
                        getCurrentTime();
184                 int eventCode = 0;
185                 /*Check if the next event is before simTime. If not set
                        next event time equal simTime*/
186                 if(next_event_time < simTime){
```

```
187                         eventCode = 1;
188                 }
189                 else{
190                         next_event_time = simTime;
191                 }
192                 SimEvent simEv = new SimEvent(next_event_time, eventCode,
                        this);
193                 eventQueue.addEvent(simEv);
194         }
195
196         private void updateState(){
197                         double newState = find_next_state();
198                         set_current_state(newState);
199         }
200
201         private double find_next_state() {
202                 int number_of_states = states.length-1;
203                 if(current_state == states[0]){
204                         return states[number_of_states];
205                 }
206                 else{
207                         for(int i = number_of_states-1; i >= 0; i--){
208                                 double state = states[i];
209                                 if(current_state > state){
210                                         return state;
211                                 }
212                         }
213                 }
214                 return 0.0;
215         }
216
217
218         public Integer[] get_Stored_State_Values(){
219                 return stored_last_state_values;
220         }
221
222         public Integer[] get_Stored_Minimum_Value_Counts(){
223                 return stored_minimum_values;
224         }
225
226 }
```

## 6.6 Cut or path set class

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.List;
4  import java.util.HashMap;
5
6  public class Cut_or_path_set {
7          private Phi system;
8          private List<Component> component_vector = new ArrayList<Component
               >();
9          private Integer index;
10         private double set_state;
11         private double set_minimum_state;
12         private Integer[] stored_state_values;
13         private Integer[] stored_minimum_value_counts;
14         private List<Double> states = new ArrayList<Double>();
15         private Double maxState;
16         private int number_of_states = 1;
17         private int components_that_has_stored_its_value = 1;
18         private int size_component_vector;
19
20         public void add_component_vector(List<Component> comps, int index,
               Phi system) {
21                 component_vector.addAll(comps);
22                 add_set_to_Component();
23                 this.system = system;
24                 find_max_state();
25                 findStates();
26                 initialize_stored_state_values_and_minimum_values();
27                 this.index = index;
28                 size_component_vector = component_vector.size();
29         }
30
31         private void findStates() {
32                         calculate_state(0,0);
33         }
34
35         private void calculate_state(int compNumber, double state) {
36                 for (double compState : component_vector.get(compNumber).
                       getStates()) {
37                         state = state + compState;
38                         if (state <= maxState) {
39                                 if (compNumber < component_vector.size()-1)
                                       {
40                                         calculate_state(compNumber + 1,
                                               state);
41                                 } else if (!states.contains(state)) {
42                                         states.add(state);
43                                 }
44                         } else {
45                                 break;
46                         }
47                 }
48                 Collections.sort(states);
49         }
50
51         private void find_max_state() {
52                 maxState = 0.0;
53                 for (Component comp : component_vector) {
54                         Double[] states = comp.getStates();
55                         maxState = maxState + states[states.length - 1];
56                         number_of_states = number_of_states+(states.length
                               -1);
57                 }
58                 set_state = maxState;
59                 set_minimum_state = set_state;
60         }
```

```
61
62          private void initialize_stored_state_values_and_minimum_values() {
63                  stored_state_values = new Integer[number_of_states];
64                  stored_minimum_value_counts = new Integer[number_of_states
                        ];
65                  for (int i = 0; i < number_of_states; i++) {
66                          stored_state_values[i] = 0;
67                          stored_minimum_value_counts[i] = 0;
68                  }
69
70          }
71
72          private void add_set_to_Component() {
73                  for (Component comp : component_vector) {
74                          comp.addCutSet(this);
75                  }
76          }
77
78          public List<Component> get_Component_vector() {
79                  return component_vector;
80          }
81
82          public Integer getIndex() {
83                  return index;
84          }
85
86
87          public void find_new_current_state(double stateDiff){
88                  set_state += stateDiff;
89                  if(set_state < set_minimum_state){
90                          set_minimum_state = set_state;
91                  }
92                  tell_Phi_about_the_change();
93          }
94
95          private void tell_Phi_about_the_change() {
96                  system.find_new_current_state(set_state, this);
97          }
98
99          public double get_current_set_state() {
100                 return set_state;
101         }
102
103         public void store_state_value() {
104                 for (int i = 0; i <= maxState; i++) {
105                         if (i == set_state) {
106                                 stored_state_values[i] =
                                        stored_state_values[i] + 1;
107                                 break;
108                         }
109                 }
110         }
111
112         public double get_maxState() {
113                 return maxState;
114         }
115
116         public Integer[] get_stored_state_values() {
117                 return stored_state_values;
118         }
119
120         public void tell_set_to_store_values() {
121                 if (components_that_has_stored_its_value ==
                        component_vector.size()) {
122                         store_state_value();
123                         store_minimum_value();
124                         system.tell_phi_to_store_state_value();
125                         set_state = maxState;
```

```
126                         components_that_has_stored_its_value = 1;
127                 } else {
128                         components_that_has_stored_its_value++;
129                 }
130         }
131
132         private void store_minimum_value() {
133                 for (int i = 0; i < states.size(); i++) {
134                         if (states.get(i) == set_minimum_state) {
135                                 stored_minimum_value_counts[i] =
                                        stored_minimum_value_counts[i] +1;
136                                 break;
137                         }
138                 }
139         }
140
141         public List<Double> getStates(){
142                 return states;
143         }
144
145         public Integer[] get_Stored_State_Values(){
146                 return stored_state_values;
147         }
148
149         public Integer[] get_Stored_Minimum_Value_Counts(){
150                 return stored_minimum_value_counts;
151         }
152
153         public String get_Component_names() {
154                 String names = "";
155                 for(Component comp: component_vector){
156                         names = names + comp.getName() + ", ";
157                 }
158                 return names;
159         }
160
161         public Integer get_set_size() {
162                 return size_component_vector;
163         }
164 }
```

## 6.7 Minimal path or minimal cut vector class

```
1  import java.util.HashMap;
2
3  public class Minimal_path_or_maximal_cut_vector {
4          /** This class will contain a minimal path or maximal cut vector *
               */
5          private HashMap<Component,Double> vector = new HashMap<Component,
               Double>();
6
7          public void makeVector(HashMap<Component,Double> v){
8                  vector = v;
9          }
10
11         public HashMap<Component,Double> returnVector(){
12                 return vector;
13         }
14
15 }
```

## 6.8 Find minimal path and minimal cut vectors class

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.Iterator;
4  import java.util.List;
5  import java.util.Map;
6  import java.util.Set;
7  import java.util.Arrays;
8  import java.util.Collections;
9
10 public class Find_minimal_path_and_maximal_cut_vectors {
11         /**This class finds the minimal path and maximal cut vectors**/
12         private HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
               maximal_cuts = new HashMap<Double, List<
               Minimal_path_or_maximal_cut_vector>>();
13         private HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
               minimal_paths = new HashMap<Double, List<
               Minimal_path_or_maximal_cut_vector>>();
14         HashMap<Integer, Cut_or_path_set> cutSets;
15         HashMap<Integer, List<Cut_or_path_set>> sorted_cut_sets = new
               HashMap<Integer, List<Cut_or_path_set>>();
16         private List<Double> systemState = new ArrayList<Double>();
17         double highest_system_state = Double.MAX_VALUE;
18         List<Component> components;
19
20         Find_minimal_path_and_maximal_cut_vectors(HashMap<Integer,
               Cut_or_path_set> cutSets, List<Component> components) {
21                 this.cutSets = cutSets;
22                 this.components = components;
23                 find_highest_system_state();
24                 find_all_system_states();
25                 find_sorted_cut_set();
26                 find_maximal_cut_vector();
27                 find_minimal_path_vector();
28         }
29
30         private void find_sorted_cut_set() {
31                 Set set = cutSets.entrySet();
32                 Iterator iterator = set.iterator();
33                 while (iterator.hasNext()) {
34                         Map.Entry mentry = (Map.Entry) iterator.next();
35                         Cut_or_path_set cut = (Cut_or_path_set) mentry.
                               getValue();
36                         int size = cut.get_set_size();
37                         List<Cut_or_path_set> sets;
38                         if(sorted_cut_sets.containsKey(size)){
39                                 sets = sorted_cut_sets.get(size);
40                         }
41                         else {
42                                 sets = new ArrayList<Cut_or_path_set>();
43                         }
44                         sets.add(cut);
45                         sorted_cut_sets.put(size, sets);
46                 }
47         }
48
49         private void find_highest_system_state() {
50                 Set set = cutSets.entrySet();
51                 Iterator iterator = set.iterator();
52                 while (iterator.hasNext()) {
53                         Map.Entry mentry = (Map.Entry) iterator.next();
54                         Cut_or_path_set cut = (Cut_or_path_set) mentry.
                               getValue();
55                         if(cut.get_maxState()<highest_system_state){
56                                 highest_system_state = cut.get_maxState();
57                         }
58                 }
```

```
59          }
60
61          private void find_all_system_states() {
62                  Set set = cutSets.entrySet();
63                  Iterator iterator = set.iterator();
64                  while (iterator.hasNext()) {
65                          Map.Entry mentry = (Map.Entry) iterator.next();
66                          Cut_or_path_set cut = (Cut_or_path_set) mentry.
                                getValue();
67                          double cutState = 0;
68                          List<Component> componentVector = cut.
                                get_Component_vector();
69                          calculate_cut_set_state(componentVector.get(0), 0,
                                cutState, componentVector, true);
70                  }
71
72          }
73
74          private void calculate_cut_set_state(Component cutComp, int
                compNumber, double cutState, List<Component> componentVector,
75                          boolean previous_comp_in_last_state) {
76                  boolean lastComp = check_if_this_is_the_last_comp(cutComp,
                        componentVector);
77                  Double[] states = cutComp.getStates();
78                  for (double state : states) {
79                          cutState = cutState + state;
80                          if (cutState <= highest_system_state) {
81                                  boolean is_this_last_state =
                                        is_this_comp_in_last_state_and_previous_last_state
                                        (previous_comp_in_last_state, states,
                                        state);
82                                  if (!lastComp) {
83                                          calculate_cut_set_state(
                                                componentVector.get(compNumber
                                                + 1), compNumber + 1, cutState,
                                                 componentVector,
84                                                         is_this_last_state)
                                                        ;
85                                  } else {
86                                          add_system_state(cutState, lastComp
                                                , is_this_last_state);
87                                  }
88                          } else {
89                                  break;
90                          }
91                          cutState = cutState - state;
92                  }
93          }
94
95          private boolean check_if_this_is_the_last_comp(Component cutComp,
                List<Component> componentVector) {
96                  boolean lastComp = false;
97                  if (cutComp == componentVector.get(componentVector.size() -
                        1)) {
98                          lastComp = true;
99                  }
100                 return lastComp;
101         }
102
103         private boolean is_this_comp_in_last_state_and_previous_last_state(
                boolean last_comp_state, Double[] states, double state) {
104                 boolean is_this_last_state = false;
105                 if (state == states[states.length - 1] && last_comp_state)
                        {
106                         is_this_last_state = true;
107                 }
108                 return is_this_last_state;
109         }
```

```
110
111         private void add_system_state(double cutState, boolean lastComp,
                boolean last_comp_state) {
112             if (!systemState.contains(cutState) && cutState <=
                    highest_system_state) {
113                 systemState.add(cutState);
114             }
115             if (lastComp && last_comp_state) {
116                 systemState.removeIf(u -> u > cutState);
117                 highest_system_state = cutState;
118             }
119         }
120
121
122         private void find_maximal_cut_vector() {
123             for (double system_state : systemState.subList(1,
                    systemState.size())) {
124                 HashMap<Component, Double> start_vector = new
                        HashMap<Component, Double>();
125                 start_vector =
                        put_components_in_minimal_path_or_maximal_cut_vector
                        (start_vector, true);
126                 Set set = cutSets.entrySet();
127                 Iterator iterator = set.iterator();
128                 while (iterator.hasNext()) {
129                     HashMap<Component, Double> new_vector = new
                            HashMap<Component, Double>();
130                     new_vector.putAll(start_vector);
131                     Map.Entry mentry = (Map.Entry) iterator.
                            next();
132                     Cut_or_path_set cut_set = (Cut_or_path_set)
                            mentry.getValue();
133                     double comps_State = 0;
134                     if (system_state == systemState.get(1)) {
135                         for (Component comp : cut_set.
                                get_Component_vector()) {
136                             new_vector.put(comp, comp.
                                    getStates()[0]);
137                             comps_State = comps_State +
                                    new_vector.get(comp);
138                         }
139                         add_to_maximal_cut_vectors(
                                new_vector, system_state,
                                comps_State);
140                     } else {
141                         calculate_cut_state(system_state,
                                cut_set.get_Component_vector(),
                                0, comps_State, new_vector, 0)
                                ;
142                     }
143
144                 }
145             }
146
147         }
148
149         private void calculate_cut_state(double system_state, List<
                Component> comps, int compNumber, double comps_summed_state,
150                     HashMap<Component, Double> vector, int cut_Number)
                            {
151
152             Component comp = comps.get(compNumber);
153             for (double compState : comp.getStates()) {
154                 vector.put(comp, compState);
155                 double temp_comps_summed_state = comps_summed_state
                        + vector.get(comp);
156                 if (system_state < temp_comps_summed_state) {
157                     return;
```

```
158                         }
159                         if (compNumber == comps.size() - 1) {
160                                 check_other_cuts(vector, system_state,
                                        temp_comps_summed_state,comps.size());
161                         } else {
162                                 calculate_cut_state(system_state, comps,
                                        compNumber + 1, temp_comps_summed_state
                                        , vector, cut_Number);
163                         }
164                 }
165         }
166
167         private void check_other_cuts(HashMap<Component, Double> new_vector
                , double system_state, double comps_State, int cut_size) {
168                 for(int i = cut_size-1; i >= 1; i--) {
169                         if(sorted_cut_sets.containsKey(i)) {
170                                 for(Cut_or_path_set cut : sorted_cut_sets.
                                        get(i)) {
171                                         double state = calculate_state(
                                                new_vector, cut.
                                                get_Component_vector());
172                                         if(state <= comps_State) {
173                                                 return;
174                                         }
175                                 }
176                         }
177                 }
178                 add_to_maximal_cut_vectors(new_vector, system_state,
                        comps_State);
179         }
180
181         private double calculate_state(HashMap<Component, Double>
                new_vector, List<Component> comps) {
182                 double state = 0;
183                 for(Component comp : comps) {
184                         state += new_vector.get(comp);
185                 }
186                 return state;
187         }
188
189         private void add_to_minimal_path_vector(HashMap<Component, Double>
                vect, double system_state) {
190                 HashMap<Component, Double> sendIn = new HashMap<Component,
                        Double>();
191                 sendIn.putAll(vect);
192                 Minimal_path_or_maximal_cut_vector new_vector = new
                        Minimal_path_or_maximal_cut_vector();
193                 new_vector.makeVector(sendIn);
194                 add_comps_to_max_or_min(system_state, sendIn, new_vector,
                        this.minimal_paths);
195
196         }
197
198         private void add_to_maximal_cut_vectors(HashMap<Component, Double>
                vect, double system_state, double comps_summed_state) {
199                 if (comps_summed_state == system_state - 1) {
200                         HashMap<Component, Double> sendIn = new HashMap<
                                Component, Double>();
201                         sendIn.putAll(vect);
202                         Minimal_path_or_maximal_cut_vector newVector = new
                                Minimal_path_or_maximal_cut_vector();
203                         newVector.makeVector(sendIn);
204                         add_comps_to_max_or_min(system_state, sendIn,
                                newVector, this.maximal_cuts);
205                 }
206         }
207
208         private void add_comps_to_max_or_min(double sysState, HashMap<
```

```
        Component, Double> sendIn, Minimal_path_or_maximal_cut_vector
        new_vector,
209                     HashMap<Double, List<
                            Minimal_path_or_maximal_cut_vector>>
                            add_to_this_vector_list) {
210             List<Minimal_path_or_maximal_cut_vector> listVect;
211             if (add_to_this_vector_list.containsKey(sysState)) {
212                     listVect = add_to_this_vector_list.get(sysState);
213                     boolean exists =
                            check_if_the_comp_combination_already_exists(
                            sendIn, listVect);
214                     if (!exists) {
215                             listVect.add(new_vector);
216                             add_to_this_vector_list.put(sysState,
                                    listVect);
217                     }
218             } else {
219                     listVect = new ArrayList<>(Arrays.asList(new_vector
                            ));
220                     add_to_this_vector_list.put(sysState, listVect);
221             }
222     }
223
224     private boolean check_if_the_comp_combination_already_exists(
            HashMap<Component, Double> sendIn, List<
            Minimal_path_or_maximal_cut_vector> vect_list) {
225             boolean exists = false;
226             for (Minimal_path_or_maximal_cut_vector vector : vect_list)
                     {
227                     HashMap<Component, Double> v = vector.returnVector
                            ();
228                     int number_of_existens_of_comps_in_sendIn = 0;
229                     for (Component c : components) {
230                             if (v.get(c) == sendIn.get(c)) {
231                                     number_of_existens_of_comps_in_sendIn
                                            ++;
232                             }
233                     }
234                     if (number_of_existens_of_comps_in_sendIn ==
                            components.size()) {
235                             exists = true;
236                     }
237             }
238             return exists;
239     }
240
241     private void find_minimal_path_vector() {
242             for (double sysState : systemState) {
243                     HashMap<Component, Double> start_vector = new
                            HashMap<Component, Double>();
244                     start_vector =
                            put_components_in_minimal_path_or_maximal_cut_vector
                            (start_vector, false);
245                     HashMap<Component, Double> new_vector = new HashMap
                            <Component, Double>();
246                     new_vector.putAll(start_vector);
247                     set_comp_states_and_calculate_cut_set_state(
                            sysState, new_vector, 0);
248             }
249     }
250
251     private void set_comp_states_and_calculate_cut_set_state(double
            system_state, HashMap<Component, Double> vector, int compNumber
            ) {
252             Component comp = components.get(compNumber);
253             for (double compState : comp.getStates()) {
254                     vector.put(comp, compState);
255                     if (compNumber < components.size() - 1) {
```

```
256                                 set_comp_states_and_calculate_cut_set_state
                                        (system_state, vector, compNumber + 1);
257                         } else {
258                                 calculate_and_check_cut_set_states(
                                        system_state, vector, 0);
259                         }
260                 }
261         }
262
263         private void calculate_and_check_cut_set_states(double system_state
                , HashMap<Component, Double> vector, int cutNumber) {
264                 Cut_or_path_set set = cutSets.get(cutNumber);
265                 List<Component> comps = set.get_Component_vector();
266                 double cutState = 0;
267                 for (Component comp : comps) {
268                         cutState = cutState + vector.get(comp);
269                         if (system_state < cutState) {
270                                 return;
271                         }
272                 }
273                 if (cutState == system_state) {
274                         if (cutNumber < cutSets.size() - 1) {
275                                 calculate_and_check_cut_set_states(
                                        system_state, vector, cutNumber + 1);
276                         } else {
277                                 add_to_minimal_path_vector(vector,
                                        system_state);
278                         }
279                 }
280         }
281
282         private HashMap<Component, Double>
                put_components_in_minimal_path_or_maximal_cut_vector(HashMap<
                Component, Double> vector, boolean max) {
283                 for (Component c : components) {
284                         if (max) {
285                                 vector.put(c, c.getStates()[c.getStates().
                                        length - 1]);
286                         } else {
287                                 vector.put(c, c.getStates()[0]);
288                         }
289                 }
290                 return vector;
291         }
292
293         public List<Double> get_system_states() {
294                 return systemState;
295         }
296
297         public HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
                get_maximal_cuts() {
298                 return maximal_cuts;
299         }
300
301         public HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
                get_minimal_paths() {
302                 return minimal_paths;
303         }
304 }
```

## 6.9 Calculate class

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.HashMap;
4  import java.util.Iterator;
5  import java.util.List;
6  import java.util.Map;
7  import java.util.Set;
8
9  public class Calculate {
10 /**This class calculate the different bounds and find r_i and p_i for the
       components**/
11         public List<double[]> ri_Components = new ArrayList<double[]>();
12         public List<double[]> pi_Components = new ArrayList<double[]>();
13         private List<double[]> ri_Cuts = new ArrayList<double[]>();
14         private List<double[]> pi_Cuts = new ArrayList<double[]>();
15         private double[] r_System;
16         private double[] p_System;
17         private double[] l_apostrophe;
18         private double[] l_doubleStar;
19         private double[] established_lower_bound;
20         private double[] new_lower_bound;
21         private int numSims;
22         private List<Component> comps;
23         private HashMap<Integer, Cut_or_path_set> cuts;
24         private Phi system;
25
26         Calculate(int numSims, List<Component> comps, HashMap<Integer,
               Cut_or_path_set> cuts, Phi system) {
27                 this.numSims = numSims;
28                 this.comps = comps;
29                 this.cuts = cuts;
30                 this.system = system;
31         }
32
33         public void calculate_r_s() {
34                 calculate_Component_rs();
35                 calculate_Cut_rs();
36                 calculate_System_r();
37         }
38
39         private void calculate_Component_rs() {
40                 for (Component comp : comps) {
41                         double[] r = new double[comp.getStates().length];
42                         for (int i = 0; i < r.length; i++) {
43                                 int storedValueCount = comp.
                                       get_Stored_Minimum_Value_Counts()[i];
44                                 r[i] = ((1.0*storedValueCount) / (1.0*
                                       numSims));
45                                 comp.get_Stored_Minimum_Value_Counts()[i] =
                                       0;
46                         }
47                         ri_Components.add(r);
48                 }
49         }
50
51         private void calculate_Cut_rs() {
52                 Set set = cuts.entrySet();
53                 Iterator iterator = set.iterator();
54                 while (iterator.hasNext()) {
55                         Map.Entry mentry = (Map.Entry) iterator.next();
56                         Cut_or_path_set cut = (Cut_or_path_set) mentry.
                               getValue();
57                         double[] r = new double[cut.
                               get_Stored_Minimum_Value_Counts().length];
58                         for (int i = 0; i < r.length; i++) {
59                                 double storedValueCount = (double) cut.
```

```
                                get_Stored_Minimum_Value_Counts()[i];
60                          r[i] = (1.0*storedValueCount) / (1.0*
                                numSims);
61                          cut.get_Stored_Minimum_Value_Counts()[i] =
                                0;
62                  }
63                  ri_Cuts.add(r);
64              }
65          }
66
67          private void calculate_System_r() {
68              r_System = new double[system.
                    get_Stored_Minimum_Value_Counts().length];
69              for (int i = 0; i < r_System.length; i++) {
70                  double storedValueCount = (double) system.
                        get_Stored_Minimum_Value_Counts()[i];
71                  r_System[i] = (1.0*storedValueCount) / (1.0*numSims
                        );
72                  system.get_Stored_Minimum_Value_Counts()[i] = 0;
73              }
74          }
75
76          public void findAllp() {
77              calculate_Component_p();
78              calculate_Cut_p();
79              calculate_System_p();
80          }
81
82          private void calculate_System_p() {
83              double[] pi = new double[system.
                    get_Stored_Minimum_Value_Counts().length];
84              List<Double> compStates = system.get_system_states();
85              calculateP(pi, compStates, r_System);
86              p_System = pi;
87          }
88
89          private void calculate_Cut_p() {
90              Set set = cuts.entrySet();
91              Iterator iterator = set.iterator();
92              while (iterator.hasNext()) {
93                  Map.Entry mentry = (Map.Entry) iterator.next();
94                  Cut_or_path_set cut = (Cut_or_path_set) mentry.
                        getValue();
95                  int i = cut.getIndex();
96                  double[] pi = new double[cut.getStates().size()];
97                  List<Double> compStates = cut.getStates();
98                  calculateP(pi, compStates, ri_Cuts.get(i));
99                  pi_Cuts.add(pi);
100             }
101         }
102
103         private void calculate_Component_p() {
104             for (int i = 0; i < comps.size(); i++) {
105                 double[] pi = new double[comps.get(i).getStates().
                        length];
106                 Double[] compStates = comps.get(i).getStates();
107                 calculateP(pi, compStates, ri_Components.get(i));
108                 pi_Components.add(pi);
109             }
110         }
111
112         private void calculateP(double[] pi, Double[] states, double[] qi)
                {
113             double p = 1;
114             pi[0] = p;
115             for (int i = 1; i < states.length; i++) {
116                 double q = qi[i - 1];
117                 p -= q;
```

```
118                         pi[i] = p;
119                 }
120         }
121
122         private void calculateP(double[] pi, List<Double> states, double[]
                qi) {
123                 double p = 1;
124                 pi[0] = p;
125                 for (int i = 1; i < states.size(); i++) {
126                         double q = qi[i-1];
127                         p -= q;
128                         pi[i] = p;
129                 }
130         }
131
132
133         public void findAll_l() {
134                 calcualte_l_apostrophe();
135                 calcualte_l_doubleStar();
136         }
137
138         private void calcualte_l_doubleStar() {
139                 l_doubleStar = new double[system.get_system_states().size()
                        -1];
140                 HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
                        maximalCuts = system.get_maximal_cut_vectors();
141                 for (int i = 1; i <= l_doubleStar.length; i++) {
142                         List<Minimal_path_or_maximal_cut_vector> paths =
                                maximalCuts.get(system.get_system_states().get(
                                i));
143                         double value = 1;
144                         for (Minimal_path_or_maximal_cut_vector vect :
                                paths) {
145                                 double vectorValue = (double) 1.0;
146                                 Set set = vect.returnVector().entrySet();
147                                 Iterator iterator = set.iterator();
148                                 while (iterator.hasNext()) {
149                                         Map.Entry mentry = (Map.Entry)
                                                iterator.next();
150                                         Double compStateInPath = (Double)
                                                mentry.getValue();
151                                         Component comp = (Component) mentry
                                                .getKey();
152                                         int j = 0;
153                                         double state = comp.getStates()[j];
154                                         while (state != compStateInPath) {
155                                                 j++;
156                                                 state = comp.getStates()[j
                                                        ];
157                                         }
158                                         if (state != comp.getStates()[comp.
                                                getStates().length - 1]) {
159                                                 j++;
160                                                 vectorValue = (1 - (1.0*
                                                        pi_Components.get(comp.
                                                        get_comp_number())[j]))
                                                         * (1.0*vectorValue);
161                                         }
162                                 }
163                                 vectorValue = 1 - vectorValue;
164                                 value = (1.0*vectorValue) * (1.0*value);
165                         }
166                         l_doubleStar[i - 1] = value;
167                 }
168         }
169
170         private void calcualte_l_apostrophe() {
171                 l_apostrophe = new double[system.get_system_states().size()
```

```
                ];
172             HashMap<Double, List<Minimal_path_or_maximal_cut_vector>>
                    minimalPaths = system.get_minimal_path_vectors();
173             for (int i = 0; i < l_apostrophe.length; i++) {
174                 List<Minimal_path_or_maximal_cut_vector> paths =
                        minimalPaths.get(system.get_system_states().get
                        (i));
175                 double biggestValue = 0;
176                 for (Minimal_path_or_maximal_cut_vector vect :
                        paths) {
177                     double vectorValue = 1;
178                     Set set = vect.returnVector().entrySet();
179                     Iterator iterator = set.iterator();
180                     while (iterator.hasNext()) {
181                         Map.Entry mentry = (Map.Entry)
                                iterator.next();
182                         Double compStateInPath = (Double)
                                mentry.getValue();
183                         Component comp = (Component) mentry
                                .getKey();
184                         int j = 0;
185                         double state = comp.getStates()[j];
186                         while (state != compStateInPath) {
187                             j++;
188                             state = comp.getStates()[j
                                    ];
189                         }
190                         vectorValue = (1.0*vectorValue) *
                                (1.0*pi_Components.get(comp.
                                get_comp_number())[j]);
191                     }
192                     if (biggestValue <= vectorValue) {
193                         biggestValue = vectorValue;
194                     }
195                 }
196                 l_apostrophe[i] = biggestValue;
197             }
198         }
199
200
201
202         void calculate_new_lower_bound(){
203             new_lower_bound = new double[system.get_system_states().
                    size()];
204             state_enumeration(new double[comps.size()],0,1.0);
205         }
206
207         private void state_enumeration(double[] componentsStates,int
                compNumber, double value) {
208             Component comp = comps.get(compNumber);
209             Double[] states = comp.getStates();
210             for(int statePlace = states.length-1; statePlace >= 0 ;
                    statePlace-- ){
211                 componentsStates[compNumber] = states[statePlace];
212                 double tempValue = value*ri_Components.get(
                        compNumber)[statePlace];
213                 if(compNumber < comps.size()-1){
214                     state_enumeration(componentsStates,
                            compNumber+1,tempValue);
215                 }
216                 else{
217                     double lowestCutState =
                            find_lowest_cut_state(tempValue,
                            componentsStates);
218                     add_enumeration(tempValue,lowestCutState);
219                 }
220             }
221         }
```

```
222
223         private void add_enumeration(double value, double lowestCutState) {
224                 List<Double> systemStates = system.get_system_states();
225                 for(int statePlace = 0; statePlace < systemStates.size();
                        statePlace++ ){
226                         if(systemStates.get(statePlace) <= lowestCutState){
227                                 double tempValue = new_lower_bound[
                                        statePlace];
228                                 new_lower_bound[statePlace] =   tempValue +
                                        value;
229                         }
230                         else{
231                                 return;
232                         }
233                 }
234         }
235
236         private double find_lowest_cut_state(double tempValue, double[]
                componentsStates) {
237                 Set set = cuts.entrySet();
238                 Iterator iterator = set.iterator();
239                 double minState = Double.MAX_VALUE;
240                 while (iterator.hasNext()) {
241                         Map.Entry mentry = (Map.Entry) iterator.next();
242                         Cut_or_path_set cut = (Cut_or_path_set) mentry.
                                getValue();
243                         double tempState = 0.0;
244                         for (Component comp : cut.get_Component_vector()) {
245                                 tempState += componentsStates[comp.
                                        get_comp_number()];
246                         }
247                         if(tempState < minState){
248                                 minState = tempState;
249                         }
250                 }
251                 return minState;
252         }
253
254         public void find_established_lower_bound() {
255                 int lenght = l_doubleStar.length;
256                 double[] B_temp = new double[lenght];
257                 for (int i = lenght-1; i >= 0; i--) {
258                         if (l_apostrophe[i+1] <= l_doubleStar[i]) {
259                                 B_temp[i] = l_doubleStar[i];
260                         } else {
261                                 B_temp[i] = l_apostrophe[i+1];
262                         }
263                         if(i != (lenght-1)){
264                                 if(B_temp[i] < B_temp[i+1]){
265                                         B_temp[i] = B_temp[i+1];
266                                 }
267                         }
268                 }
269                 established_lower_bound = B_temp;
270         }
271
272
273         public double[] get_p() {
274                 return p_System;
275         }
276
277         public double[] get_established_lower_bound() {
278                 return established_lower_bound;
279         }
280
281         public double[] get_l_doubleStar() {
282                 return l_doubleStar;
283         }
```

```
284
285         public double[] get_l_apostrophe() {
286                 return l_apostrophe;
287         }
288
289         public List<double[]> get_Component_r_s() {
290                 return ri_Components;
291         }
292
293         public List<double[]> get_Cut_r_s() {
294                 return ri_Cuts;
295         }
296
297         public double[] get_new_lower_bound(){
298                 return new_lower_bound;
299         }
300
301         public double[] get_System_r_s() {
302                 return r_System;
303         }
304 }
```

# Bibliography


[1] Discrete event simulation. *Discrete event simulation— Wikipedia, The Free Encyclopedia*. [Online; accessed 20-September-2017]. URL: https://en.wikipedia.org/wiki/Discrete_event_simulation.

[2] Exponential distribution. *Exponential distribution— Wikipedia, The Free Encyclopedia*. [Online; accessed 20-September-2017]. URL: https://en.wikipedia.org/wiki/Exponential_distribution.

[3] Jørund Gåsemyr. "Note on flow network systems". Private communication. October (2017).

[4] Jørund Gåsemyr and Bent Natvig. "Improved availability bounds for binary and multistate monotone systems with independent component processes". In: *Journal of Applied Probability* 54.73 (2017), pp. 750–762. DOI: https://doi.org/10.1017/jpr.2017.32.

[5] Bent Natvig. *Multistate Systems Reliability Theory with Applications*. WILEY, 2011. ISBN: 978-0-470-69750-4.