Probabilistic Control in a Multi-Player Context

Pedro Miguel Lúcio Melgueira

Tecnhical Report 23/2012
BII grant
Acknowledgements

I would like to thank Professor Miguel Barão for orienting me in this project and supporting me while working on it. Without him this project would not have been concluded as it was.

I would also like to thank INESC-ID Lisboa for providing me with this extension to the investigation grant which was a great experience.
# Contents

1 Introduction 4

2 Controlled Markov Chains 6
   2.1 Introducing Controlled Markov Chains .................. 6
   2.2 Controlled Markov Chains With More Players and Signals .. 7

3 Optimization Algorithm 10
   3.1 Objective of Optimization ............................... 10
   3.2 The Algorithm For One Player ........................... 11
   3.3 The Algorithm For Multiple Players ...................... 12
   3.4 Stopping the Algorithm with a Single Player ............ 13
   3.5 Stopping the Algorithm with Multiple Players .......... 14

4 Development 15
   4.1 A Toolbox For Probability Calculus and Optimization ...... 15
   4.2 Changes to the library ................................... 15
      4.2.1 The MDMatrix Data Structure ....................... 16
      4.2.2 The Use of BLAS .................................. 16
   4.3 Data Structures ....................................... 17
      4.3.1 Data Structure PlayersLinkedList .................. 17
      4.3.2 Data Structure Cmc ................................ 18
   4.4 Creating a Cmc Using cmc_newCmc() ...................... 19
   4.5 Implemented Algorithms ................................. 20
      4.5.1 Optimize Controlled Markov Chain .................. 21
      4.5.2 Calculate Kulback-Leibler Divergence ............... 21
      4.5.3 Auxiliary Functions .............................. 22
      4.5.4 Output Results .................................. 23

5 Test Cases 26
   5.1 Test Case 0 - One Player, One Signal ..................... 26
   5.2 Test Case 1 - One Player, One Signal ..................... 28
   5.3 Test Case 2 - Two Players .............................. 30
   5.4 Test Case 3 - Three Players ............................ 32
Chapter 1

Introduction

This report is about the project developed during the extension of an integration to investigation grant made available by the Control of Dynamic Systems Group - Inesc-ID Lisboa. The project was developed under the supervision of Professor Miguel José Simões Barão.

The project aimed to use the library developed during the first part of the grant. The library, which is called a Toolbox for Probability Calculus and Optimization, performs probability calculus using multidimensional matrices, called MD matrices, to represent probability distributions and conditional probability distributions. Because of its multidimensional feature, every operation that is possible to perform with the library, such as products like $P(X, Y|Z, W)P(Y, Z) = P(X, Y, Z|W)$, must be represented using a very dynamic method. Because of this dynamic method it becomes simple for the user to input large quantities of data to perform calculations with it.

Every time there is a mention of the library in this report refer to [1] for reference.

This part of the project dealt with the implementation of some operations that optimize the actions done by players in a multi-player context. This multi-player context, along with the players themselves are represented using a model called controlled Markov chain.

Chapter 2 is about controlled Markov chains. It explains what they are and what are the different models used in this project.

The main algorithm for the implemented algorithms is first specified in [3] and was later detailed in [2]. Chapter 3 explains not only how the operation of the algorithm is done but it also explains how the mathematical expression of said operation can be rewritten so the expression is easier to understand while implementing it in the library. One other thing specified in chapter 3 are the conditions that stop the algorithm.

Chapter 4 talks about the development of the project, mainly about the data structures that were developed and used.

Finally chapter 5 is about some test cases that were done to test the
developed project and draws some conclusions regarding the results of the algorithm.
Chapter 2

Controlled Markov Chains

2.1 Introducing Controlled Markov Chains

To understand a controlled markov chain one should first understand markov chains. A markov chain is a mathematical model that can be represented has a graph. Each node of that graph represents one state in a finite set of possible states and each edge of the graph represents the probability of transition from one state to another state. A simple example is in figure 2.1. As it is observable, each transition depends only in the current state.

![Markov Chain Diagram](image)

Figure 2.1: A markov chain.

To represent the transitions between states, be that in the markov chain model or controlled markov chain model, probability distributions are used. For a markov chain, assume \( x_t \) is one possible state at time period \( t \). The distribution

\[
P(x_{t+1} | x_t),
\]

represents the probability of changing to state \( x_{t+1} \) knowing that the
current state of the system is \( x_t \).

In a controlled Markov chain the transitions between states depend not only in the current state but also in the input from one or more players (also named controllers or agents). The graph for a simple example of a controlled Markov chain with one player can be represented like in figure 2.2.

![Figure 2.2: A controlled Markov chain.](image)

To represent a controlled Markov chain with one player the distribution that represents the transition will be

\[
S(x_{t+1}|x_t, u_t),
\]

where \( u_t \) represent the input from the player. This distribution is referred to as system. Now becomes important to also represent the distribution for the player itself. That will be

\[
C(u_t|x_t).
\]

The distribution represents the probability of the controller outputting the signal \( u_t \) knowing the current state is \( x_t \).

With this in mind the controlled Markov chain is represented graphically like figure 2.3, where there is a system that changes its state according to an input of a player and a player that outputs certain signals according to the current state of the system.

### 2.2 Controlled Markov Chains With More Players and Signals

It is possible to define a system with more then one player or more then one signal. The distribution \( S \) will simple be defined has

\[
S(x_{t+1}|x_t, u^1_t, \ldots, u^n_t),
\]
Figure 2.3: A controlled markov chain with one player.

where \( n \in \mathbb{N} \) is the total number of players in the system.

For the same system there are now two ways of representing the player or players. The first way is representing one single player that outputs many signals. That approach has the representation

\[
C(u^1_t, \ldots, u^n_t | x_t).
\]

One other way is representing many players with each one having a single output,

\[
C_1(u^1_t | x_t), \ldots, C_n(u^n_t | x_t).
\]

In the second method, if the signals are independent then it is possible to calculate the product between them and obtain one player with all of the signals joint in one distribution,

\[
C(u^1_t, \ldots, u^n_t | x_t) = \prod_{k=1}^{n} C^k(u^k_t | x_t).
\]

A graphical representation for a controlled markov chain with two separated players is shown in figure 2.4.

In chapter 5 there are a few test cases that optimize a controlled markov chain with multiple players then calculate the product between those player and optimizes the chain with a single player. This is done to compare how the algorithm behaves in terms of time and to compare the results.
Figure 2.4: A controlled Markov chain with two players.
Chapter 3

Optimization Algorithm

This chapter deals with interpreting the algorithm described in [3] and [2] that is used to optimize the players in a controlled Markov chain.

3.1 Objective of Optimization

When talking about optimization it is necessary to know some target condition or optimal value. For example, when maximizing a function \( f : \mathbb{R} \rightarrow \mathbb{R} \) it is known that if that function has a maximum then that will be the value of \( x \) so that \( f(x) > f(y) \), with \( y \) being any other value in the function’s domain different than \( x \).

In the case of controlled Markov chains there are two things that are interpreted as targets for optimization. They are the distribution of the optimal system \( S'(x_{t+1}) \) and the distribution of the optimal player or players \( C'(u_k) \). Both distribution represent the desired distributions for the system and the players.

As stated in [2], using the product rule of probabilities and assuming independence from the players of a controlled Markov chain, the following distribution can be written

\[
P(x_{1:T}, u_{0:T-1}^1, \ldots, u_{0:T-1}^n | x_0) = \prod_{t=0}^{T-1} \prod_{k=1}^{n} S(x_{t+1}|x_t, u_{t}^1, \ldots, u_{t}^n) C_k(u^k_t | x_t).
\]

The distribution represents the joint distribution of the state and actions over a period \( T \).

The same can be done with the optimal system and the optimal players. The distribution is

\[
Q(x_{1:T}, u_{0:T-1}^1, \ldots, u_{0:T-1}^n) = \prod_{t=0}^{T-1} \prod_{k=1}^{n} S'(x_{t+1}) C'_k(u^k_t).
\]
The algorithm finds the distribution for the players, \( C^k \) so that the Kullback-Leibler divergence \( D(P \| Q) \) gets minimized. The Kullback-Leibler divergence is defined as

\[
D(P \| Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}.
\]

### 3.2 The Algorithm For One Player

To optimize the player an explicit expression is used. The expression was found in [3] and is written as

\[
C(u_t \mid x_t) = \exp \left( \log C'(u_t) - D(S_{t+1} \| S'_{t+1}) - E_{S_{t+1}}[- \log \gamma_{t+1}(x_{t+1})] - \log \gamma_t(x_t) \right)
\]

where

\[
\gamma_t(x_t) = \sum_{u_t} \exp \left( \log C'(u_t) - D(S_{t+1} \| S'_{t+1}) - E_{S_{t+1}}[- \log \gamma_{t+1}(x_{t+1})] \right)
\]

(3.2)

is a normalization constant.

The above expression is used to calculate one player. This solution however, for one player, is not used once but it is used iteratively. The algorithm executes a number of \( T \) times. However it doesn’t start with \( t = 0 \) but rather with \( t = T - 1 \) with each iteration decrementing the value of \( t \) until it is equal to 0. It is noticeable that the values of \( S, S' \) and \( C' \) are equal for every \( t \). The only think that changes in each iteration are the values of \( \gamma_{t+1}(x_{t+1}) \), which is the normalization constant of the previous iteration.

When the algorithm starts, that is, when \( t = T - 1 \) there is no \( \gamma_{t+1}(x_{t+1}) \) calculated. So the value for this distribution get initialized equaling 1 for every \( \gamma_{t+1}(x_{t+1}) \). Like this log \( \gamma_{t+1}(x_{t+1}) \) is equal to 0 and this doesn’t affects the calculation.

The value of \( T \) can be set as a parameter for the algorithm but there is no need for that. This is because with each iteration the results will converge to one solution. Section 3.4 talk about how to stop the algorithm.

To work better with the library the expression 3.1 got rewritten. First, some of the parts inside the exponential function are represented like

\[
A = \log C'(u_t) - D(S_{t+1} \| S'_{t+1}).
\]

Also the expected value inside the exponential function has a similar calculation to the cross entropy between \( S_{t+1} \) and \( \gamma_{t+1}(t+1) \), the expression is now represented like
\[ C(u_t | x_t) = \exp\left( A + H(S_{t+1}, \gamma_{t+1}(x_{t+1})) - \log \gamma_t(x_t) \right). \]

One final step is to place the normalization constant dividing the exponential. So the expression becomes

\[ C(u_t | x_t) = \frac{\exp\left( A + H(S_{t+1}, \gamma_{t+1}(x_{t+1})) \right)}{\gamma_t(x_t)} \]  

(3.3)

where

\[ \gamma_t(x_t) = \sum_{u_t} \exp\left( A + H(S_{t+1}, \gamma_{t+1}(x_{t+1})) \right) \]  

(3.4)

is a normalization constant.

The expression 3.3 and 3.4 are written representing a single signal. But in the case of chains with multiple signals represented by only one player the expression is basically the same and can be written like

\[ C(u_1^t, \ldots, u_n^t | x_t) = \frac{\exp\left( A + H(S_{t+1}, \gamma_{t+1}(x_{t+1})) \right)}{\gamma_t(x_t)} \]  

(3.5)

where

\[ \gamma_t(x_t) = \sum_{u_1^t} \exp\left( A + H(S_{t+1}, \gamma_{t+1}(x_{t+1})) \right) \]  

(3.6)

is a normalization constant. The value of \( A \) is calculated like

\[ A = \log C'(u_1^t, \ldots, u_n^t) - D(S_{t+1} \| S_{t+1}'). \]  

(3.7)

The distributions for the system is of course written like, \( S(X_{t+1} | x_t, u_1^t, \ldots, u_n^t) \).

### 3.3 The Algorithm For Multiple Players

For controlled markov chains with multiple players the same expression 3.3 is used. But before it is used something called an equivalent system must be calculated.

An equivalent systems \( S_k \) is what is perceived by a player \( C_k \). A graphical representation is in 3.1. To calculate this equivalent system the following formula is used,

\[ S_k(x_{t+1} | x_t, u_k^t) = \sum_{u_1^t, \ldots, u_k^t} S(x_{t+1} | x_t, u_1^t, \ldots, u_k^t) \prod_{u_1^t, \ldots, u_k^t} C_k(u_k^t | x_t). \]

The algorithm to optimize a controlled markov chain with two players, as stated in [2], is this:
Figure 3.1: Equivalent system.

- Set controller 1 to some initial candidate controller.
- Using controller 1, compute the equivalent system $S_2$.
- Optimize controller $C_2(u_2^t \mid x_t)$ with respect to the equivalent system $S_2$ using the equation 3.3.
- Using controller 2, compute the equivalent system $S_1$.
- Optimize controller $C_1(u_1^t \mid x_t)$ with respect to the equivalent system $S_1$ using the equation 3.3.
- Repeat until convergence is achieved.

For this project this algorithm generalized for $n$ players. The algorithm can be written like:

- Set every controller to some initial candidate controller.
- Using every controller except one controller $C_k$ compute the equivalent system $S_k$. $S_k$ will be the system perceived by controller $C_k$.
- Optimize controller $C_k(u_k^t \mid x_t)$ with respect to the equivalent system $S_k$ using the equation 3.3.
- Repeat for every controller.
- Repeat the algorithm until the stopping conditions are achieved.

3.4 Stopping the Algorithm with a Single Player

It is said that the algorithm should stop when convergence is achieved. Converging, in this context, means that the last calculated player is very similar to the second last calculated player. In turn, being similar means that the difference between a value $C_t(u_t = a \mid x_t = b)$ and $C_{t+1}(u_{t+1} =$
\(a \mid x_{t+1} = b\), with \(a\) and \(b\) belonging to the state space of the corresponding variables, is very small.

To detect if the calculated player are converging an approached similar to the euclidean distance is used. First suppose two calculated players, \(M_{i,j} = C_t(u_t \mid x_t)\) and \(N_{i,j} = C_{t+1}(u_{t+1} \mid x_{t+1})\). \(M_{i,j}\) and \(N_{i,j}\) are their corresponding matrices, \(i\) and \(j\) are indexes that represent the columns and rows of the matrices. Comparing each element of a matrix to that of a point in an euclidean space, the euclidean distance can be calculated like

\[
\alpha = \sqrt{\sum_i \sum_j (M_{i,j} - N_{i,j})^2}.
\]

If \(\alpha < \epsilon\), with \(\epsilon\) begin a small value close to zero, the algorithm stops.

### 3.5 Stopping the Algorithm with Multiple Players

When talking about multiple players, convergence is achieved when every player converges. This means the euclidean distance approached is used with every player in the system. But after calculating the distance for every player the result is a vector \(A = \alpha_n\), with \(n\) being the number of players. So now something as to be done with the vector to decide if the algorithm can stop or no.

The algorithm uses a percentage \(\delta\) that represents the minimum number of players that needs to converge before the algorithm can stop. If \(\delta = 0\), then none of the players need to converge before the algorithm stops, but that doesn’t make much sense. If \(\delta = 1\), then every player needs to converge for the algorithm to stop.
Chapter 4

Development

The code implemented in this project was developed in a Linux environment, Ubuntu, and was written using the C programming language. The project used the library developed in the first part of this grant called a Toolbox for Probability Calculus and Optimization.

The code in the library was modified to correct a few bugs left in the first submission. The code is now compiled using the C99 standard as opposed to the C89 standard in the first submission. Aside from that, the code also suffer two major modifications which will be described in section 4.2.

4.1 A Toolbox For Probability Calculus and Optimization

This library was develop to work with probability distribution and in both and efficient and easy way to perform various calculations with those distributions. These calculations are mostly products of the form $P(X \mid Y) P(Y) = P(X, Y)$ or divisions like $\frac{P(X, Y)}{P(Y)} = P(X \mid Y)$.

The library also allows the calculation of information theory measures like the entropy, cross entropy, mutual information, Kullback-Leibler divergence among other functions and algorithms.

One of the main features of the library is the fact that it works with distributions with an undetermined number of random variables.

4.2 Changes to the library

Two major changes were made to the library. This section justifies those changes.
4.2.1 The $MDMatrix$ Data Structure

One of those modifications was changing how the data structures were used. In the first submission there is a data structure called $MDMatrix$ that is used to represent the data of a probability distribution or simply the data of a multidimensional matrix. In the first submission the $MDMatrix$ structure was always declared statically. This lead to the bad use of conventions.

```c
typedef struct MDMATRIX_STRUCT {
    double *array;
    int array_size;

    MDVar *vars;
    int vars_size;

    int prodist;
} MDMatrix;

void fun(MDMatrix mdm);
```

Figure 4.1: Old $MDMatrix$ declaration code and a generic function.

The code for the $MDMatrix$ was written like in figure 4.1. Also in figure 4.1 there is a signature of a function that has an $MDMatrix$ as a parameter. Analyzing that code it is observable that every time an $MDMatrix$ gets passed to a function by parameter some of the variables in the $MDMatrix$ in the argument get assign a value that can be later changed without changing the original value. While that happens the pointer in the argument get to point to the same address as the original pointer.

So by changing a variable in the argument the original doesn’t change. But by changing the data pointed by the pointer in the argument the original will change. Although the code was written so this situation would never happen this is still seen as code not following conventions.

To follow the conventions the declaration of the data structure $MDMatrix$ was simply changed so the list } $MDMatrix$ is now } *$MDMatrix$; This let to some alteration of the code so the symbol $m1.array_size$ would be changed to $m1->array_size$ among other less important things.

4.2.2 The Use of BLAS

One other thing that was changed in the library was the use of another library called BLAS (Basic Linear Algebra Subprograms). In the beginning of the project it was intended that the code developed was to be able to handle very specific probability distributions. That means that the distributions would have a predefined structure and a predefined number of variables.
The BLAS library is prepared to handle data formatted in the form of either a real scalar, a vector or a matrix, this terms being linear algebra term and not computer science term where vector and matrix are ways of organizing memory spaces. The way BLAS works with matrices is by using a process called vectorization. Vectorization of a matrix allows writing a linear algebra matrix in a single vector. By working with the value in a single vector BLAS is able to perform linear algebra operations like dot products.

However, BLAS can’t exactly handle the operations performed with probability distributions. Even if many of the operations on the distributions are systematic or comparable to those of linear algebra, it becomes very hard to use the somewhat limited functions of BLAS.

Adding to that the library quickly took a turn towards implementing multidimensional operations. Great part of that project was the development of data structures and a method that made possible to have mathematical structures similar to matrices but with a theoretical infinite number of dimensions. The way this was achieved was first inspired in the vectorization process used by BLAS, but again, BLAS was very limited in terms of many dimensions.

As the first project progressed the use of BLAS got reduced to scalar operations, that either multiply or divide every element of a matrix by a scalar, and exponential and logarithmic operations. The rest of the project basically relied on the develop method that related two multidimensional objects to perform the operations following the rules of probability products and the like.

During the second part of the grant, that is this part, it was decided that it would no longer be necessary to use BLAS. This made the compilation process much more simple, it made the general complexity of the project also more simple and the operations that BLAS use to perform are know done using standard C programming without any special method or third party libraries.

4.3 Data Structures

4.3.1 Data Structure PlayersLinkedList

The data structure PlayersLinkedList is a double linked list implemented in the C programming language. Each element of the list is an MDMatrix.

During the execution of both algorithms that optimize a controlled markov chain with each iteration of the algorithms a new player (or players depending on the system), gets calculated. It becomes necessary, or at the very least convenient, to save every player calculated.

Since the algorithm stops in a dynamic way there is not way of knowing how many players are going to be calculated during the execution of the
algorithm. Because of that the calculated players could not have been stored
in more simple data structures like arrays.

To go around this problem the double linked list was implemented. Each
time a players gets calculated it gets inserted next to the head of the list.
Like this the complexity of insertion is always constant.

4.3.2 Data Structure Cmc

To work with the controlled markov chains a data structure was develop.
That data structure holds not only the distributions that represent the sys-
tem but also some variables that contain the information about the number
of players and signals in that system and other information relevant to the
calculations.

```c
typedef struct CMC
{
    int o_xt_id;
    int o_xt_1_id;
    MDMatrix system;
    MDMatrix optimal_system;
    int n_uts;
    int n_players;
    int *o_uts_id;
    PlayersLinkedList *players;
    MDMatrix *optimal_players;
    int iterations;
    double max_convergence_error;
    double min_converged_players;
} *Cmc;
```

Figure 4.2: Declaration of the data structure that represents a controlled
markov chain

The declaration using the C programming language is in figure 4.2. First
of all, note that two MDMatrix data structures called system and optimal_system.
They represent the probability distribution that represents the
system and the distribution of the optimal system respectively. Mathemat-
ically they are written like $S(x_{t+1}|x_t, u_1^t, \ldots, u_n^t)$ and $S'(x_{t+1})$.

The pointer *optimal_players points to an array of MDMatrix data
structures. In this array are the various optimal players for the controlled markov
chain. The order in which they are placed in the array is of no importance.

The variable n_players stands for and holds the number of players. The
variables n_uts stands for and also holds the number of signals. The signals
are represented in this report using the notation $u_t$ hence $ut$ in the name of
the variable.

Adding to the system and optimal system there are two variables that are \( o_{xt} \) and \( o_{xt+1} \). In the Toolbox for Probability Calculus and Optimization, there is a data structure that represents the random variables of a probability distribution. The data structure is called \( MDVar \). It is a simple data structure, it only has three integer variables. One of those variables is the \( id \). The id is an integer identifier that has to be unique for one variable.

Suppose that in some project it would be intended to represent two probability distributions \( P(X, Y \mid Z) \) and \( Q(W \mid X) \). This would mean that there would be two \( MDMatrix \) data structures and each would have a few \( MDVar \) data structures. The \( id \) of the \( MDVar \) that will represent the \( X \) in the distribution \( P \) will be the same as the one that represents \( X \) in the distribution \( Q \).

The \( id \) of the \( MDVar \) data structures for the variables \( x_t \) and \( x_{t+1} \) are stored in \( o_{xt} \) and \( o_{xt+1} \) respectively.

Likewise, the pointer \( o_{uts} \) will point to an array that has the size of the number of signals and each element of the array will have the original \( id \) of the signal or signals. In the situation where there are many signals, the order in which the \( ids \) are stored is important. That order is simply the same order in which the signals are placed in the \( MDMatrix \).

The algorithm, as it was said before, is an iterative algorithm. Therefore there is a finite number of cycles in which the algorithm gets executed. That number is stored in the variable \( iterations \).

In description of the algorithm there is an \( \epsilon \) that represent an error value used to detect if the algorithm should stop at its current iterations or not. This value is stored in the variable \( max_{convergence \_error} \).

There is also a \( \delta \) value that represents the minimum number of players that as to converge when optimizing a controlled markov chain with multiple players. That value is stored in the variable \( min_{converged \_players} \).

The last aspect of this data structure to talk about is the pointer \( \times \) and the data structure \( PlayersLinkedList \). This will be discussed in the following sections.

### 4.4 Creating a Cmc Using \texttt{cmc\_newCmc()} 

```c
Cmc cmc_newCmc(MDMatrix sys, MDMatrix osys, MDMatrix *op, int xt_id, int xt_1_id, int n_players, int n_uts, double mce, double mcp);
void cmc_freeCmc(Cmc c);
MDMatrix *getLastPlayers(Cmc cmc);
```

Figure 4.3: Function to create a new Cmc.
To create a $Cmc$ data structure to work with the algorithm it is recommended that the function $cmc\_newCmc$ is used. This function is an algorithm that takes a few $MDMatrix$ along with other values and returns controlled markov chain ready to be used with the algorithm.

The most important aspect of this function is the fact that it saves the $MDMatrix$ data structures with a few alterations. Those alterations are done to the $ids$ of each random variable of the distributions. The $ids$ of the variables representing the state and next state of the controlled markov chain, that is the $x_t$ and $x_{t+1}$ respectively, are changed to the value 0 and 1 respectively.

The $ids$ of the signals are also changed. Each $id$ of a different signal gets a negative value assigned, starting with $-1$.

The reason to change the ids is due to the fact that it becomes very convenient for the algorithms to identify what is a signal and what is a state while using the different distributions.

One other reason is that some iterative calculations may need to differentiate between a variable at time $t$, $t+1$, $t+2$, etc. This would mean that in each iteration of the optimization algorithms each calculated player would have a different $ids$ for it’s variables.

In this project this possibility is not used in the final submitted project but it was explorer in early stages of the project.

Speaking about the parameters for the function $cmc\_newCmc()$, $sys$ and $osys$ will be copied with the changes described above to the $MDMatrix$ data structures $system$ and $optimal\_system$ respectively.

The parameter $*op$ represents the optimal player or players. This is, therefore, a pointer to an array of $MDMatrix$ data structures that each represent an optimal player. For the situation where there is one player, either with one or many signals, the array will have only one element which is the optimal player to be used for that calculation.

The parameter $xt\_id$ and $xt\_1\_id$ contain the $id$ of the random variables that represent $x_t$ and $x_{t+1}$ respectively.

Parameter $n\_players$ is the number of players and $n\_uts$ the number of signals.

The parameters $mce$ and $mcp$ are the values for $max\_convergence\_error$ and $min\_converged\_players$.

4.5 Implemented Algorithms

This section deals with the different algorithms implemented in this project. Those are the algorithms used to optimize a controlled markov chain and output results among others.
4.5.1 Optimize Controlled Markov Chain

This two functions are the ones where the algorithms for optimization are implemented. There is one for the case where there is only player and other for the case where there are many players.

In both cases the result of that optimization is stored in the data structure $\text{Cmc}$ that was passed to the function by argument. In the data structure the players are keep in the double linked lists in the array $\ast \text{players}$. The first element in any of the lists is the last player calculated.

**Optimize CMC With One Player**

```c
void optimizeCMC_SP(Cmc cmc, int log_base);
```

When a controlled markov chain has only one player, regardless of that player having one or many signals, this function should be used. The parameter $\text{Cmc}$ is, of course, supposed to be initialized with the function $\text{cmc\_newCmc()}$. The parameter $\text{log\_base}$ is the base of the logarithms and also the base for the exponential functions used in the different calculations.

**Optimize CMC With Multiple Players**

```c
void optimizeCMC_MP(Cmc cmc, int log_base);
```

This function is used to optimize a controlled markov chain with multiple players. In here it is necessary for each player to have only one signal.

The signature of the function and what each parameter is for is similar to the one for one players.

4.5.2 Calculate Kulback-Leibler Divergence

```c
MDMatrix calcKLD(Cmc cmc, double log_base);
```

This function calculates two distributions and returns an $\text{MDMatrix}$ containing the Kullback-Leibler divergence between them. The distributions are

$$
P(x_{t+1}, u_{t}^{1}, \ldots, u_{t}^{n} \mid x_1) = \prod_{k=1}^{n} S(x_{t+1} \mid x_t, u_{t}^{1}, \ldots, u_{t}^{n}) C_{k}(u_{t}^{k} \mid x_t),$$

and

$$
Q(x_{t+1}, u_{t}^{1}, \ldots, u_{t}^{n}) = \prod_{k=1}^{n} S'(x_{t+1}) C'_{k}(u_{t}^{k}).
$$
The distribution $P$ is calculated performing the product between the system’s probability distribution $S$ and every one of the last calculated players $C^k_t$. The distribution $Q$ is calculated by performing the product between the optimal system’s distributions $S'$ and the optimal players $C'_k$.

The resulting $MDMatrix$ is indexed by the random variable $x_t$ and contain the Kullback-Leibler divergence between $P$ and $Q$ when $x_t$ is equal to some value in it’s state space.

### 4.5.3 Auxiliary Functions

This functions are used by the two algorithms above for calculations. They should not be used individually.

#### Calculate $A$

```c
MDMatrix calcA(MDMatrix sys, MDMatrix osys, MDMatrix op, double log_base);
```

This function calculates the value of $A$, 3.7, in the expression 3.5. The argument $sys$ is the system of the controlled markov chain, $osys$ is the optimal system and $op$ is one optimal player.

Parameter $log\_base$ is the base used in the base of the logarithm and exponential functions used in these calculations.

#### Calculate Explicit Expression

```c
MDMatrix calcExplicitExpression(MDMatrix sys, MDMatrix prev_it_n, MDMatrix A, double log_base);
```

This function calculates the result of the explicit expression 3.5. Like in other functions, the parameter $sys$ represents the system of the controlled markov chain. The parameter $prev\_it\_n$ represents the normalization of the previous iteration. Mathematically this is $\gamma_{t+1}(x_{t+1})$ from 3.5.

The parameter $A$ is what gets calculated by the function $calcA$ described above.

Parameter $log\_base$ is the base used in the base of the logarithm and exponential functions used in these calculations.

#### Calculate Equivalent System

```c
MDMatrix calcEquivalentSystem(Cmc cmc, int index);
```

This function calculates the equivalent system as describe in 3.3. The parameter $index$ represents the index in the linked lists *players* from which to use the players when calculating the equivalent system.
Check If Algorithm For Multiple Players Can Stop

```cpp
bool optimizaCMC_MP_stop(Cmc cmc);
```

This function just receives a controlled markov chain a checks if it’s current values are sufficient to make the algorithm stop. This functions is used during the optimization algorithms and is called at every iteration.

The algorithm used for this is the one described in 3.5. The function is only used when optimizing a controlled markov chain with multiple players.

### 4.5.4 Output Results

The functions specified in this section deal with outputting the calculated information to the `stdout`. The objective is to get the results displayed on a terminal quickly for a fast reading while testing.

Such algorithms as the ones implemented in this project would normally be implemented with other projects. Therefor these output functions would probably be disregarded in that situation.

Something important about these functions is that they all receive only a controlled markov chain by argument. The functions also suppose the chain they receive has been through an optimization algorithm an actually has some results to display.

The functions make use of all the attributes in the data structure and display them in the screen. They also displays every player calculated and not only the last one.

There are three functions to display results, `outputResults_CommonOutput()`, `outputResults_SP()` and `outputResults_MP()`. The first one is actually common for both the other, hence the name, and it is actually called at the beginning of them. The other two function are the ones that should be called by the user to see the results. These other two functions are for displaying the results of a chain with a single player or multi-players, respectively.

The examples used below should be viewed as arbitrary examples. It is not intended of this part of the report to offer real examples for interpretation. That is done in the chapter 5.

**Output Results Common**

```cpp
void outputResults_CommonOutput(Cmc cmc);
```

This function, which is called at the beginning of the other two, is used to output the basic information about a chain. Bellow is an example about what the function writes when it is used for chains with multi-players.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Convergence Error</td>
<td>0.000100</td>
</tr>
<tr>
<td>Minimum Converged Players</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
Number of Players: 3;
Number of signals: 3;

Calculation Results:
Number of Iterations: 6;

The output is very self-explanatory. The first two line are the errors to stop the algorithm. The next two are the number of players and signals and finally the number of iteration used by the algorithms before convergence was achieved.

Output Results Single Player

void outputResults_SP(Cmc cmc);

This function is the one used to output the results of a single player chain. It’s output, without the common part, is:

********
Players:
********

Iteration: 0;
0 0.187912 0.812088
1 0.993374 0.006626

Iteration: 1;
0 0.244516 0.755484
1 0.978868 0.021132

(...)

Iteration: 10;
0 0.229323 0.770677
1 0.984169 0.015831

Iteration: 12;
0 0.229324 0.770676
1 0.984169 0.015831

The function actually displays every iteration, not only the first and the last two. But to keep the text small iteration from 3 through 10 were cut.

This output is from a chain where the system has two states and the single player has a signals that also has two states. The way to read the output is, in iteration 12, if the current state is 0, then the player should emit signal 1. If the state is 1, the player should emit signal 0.

With that said the rows are the states, the columns are the signals.

A note on the side. It is interesting to see how in the first two interactions the two calculated players are so different but in the last two they are almost the same.
Output Results Multiple Players

```c
void outputResults_MP(Cmc cmc);
```

This function is the one used to output the results of a multilayer player chain. It’s output, without the common part is shown bellow.

In here it is observable that this is an example of a chain with 3 states and 3 players. An also, each player has two signals. Each iteration displays the information of the players separately but still keeps the convention of rows for states and columns for signals.

```
******
Players:
******

( ... )

Iteration: 5;
Player with signal 10:
0 − 0.999768 0.000232
1 − 0.002074 0.997926
2 − 0.000245 0.999755
Player with signal 20:
0 − 0.000313 0.999687
1 − 0.999005 0.000995
2 − 0.000461 0.999539
Player with signal 30:
0 − 0.000398 0.999602
1 − 0.780503 0.219497
2 − 0.011512 0.988488
```

This time, also for convinience, some parts of the outout were cut. In here there is only one iteration.
Chapter 5

Test Cases

The following test cases were made to test if the algorithms implemented were correct by analyzing their results.

The tests were made up and were not adapted from any real problem. There are tests that are about controlled markov chains with one player and one signals, one player but many signals and many players each with one signal.

The tests were submitted in this project along with this report in the same order they are presented in here. In the project itself there are other test cases that were used but were not inserted in this report because they didn’t have anything interesting that the other test cases had not.

The information about each test is displayed in the form of matrices as opposed to graphs so the test cases are legible.

5.1 Test Case 0 - One Player, One Signal

This first test case is about the most simple controlled markov chain that can be optimized. The chain only has two states, one player and one signal with two outputs. In this test case a single player will be generated and that player will display what the optimal player solution is for the given system regarding a given optimal system and an optimal player.

The test is intended to check if the results are compatible with what a human would think is correct. This is possible to due only for very small chains like this one. If the results are indeed compatible with human reasoning then it is safe to assume that the developed code is working correctly.

First of all, the system for this test case is in figure 5.1. Like it was said, the system has two states.

Now both an optimal system and an optimal player are need to execute the algorithm. The optimal system should reflect the desired state. That means if the desired state is state $x_{t+1} = 1$, then the distribution of the optimal system should have a very high probability for this state. Figure 5.2
\[ S(x_{t+1} \mid x_t, u_t) = \]

<table>
<thead>
<tr>
<th>( u_t = 0 )</th>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t = 0 )</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>( x_t = 1 )</td>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( u_t = 1 )</th>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t = 0 )</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>( x_t = 1 )</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Figure 5.1: System for test case 0.

reflects this idea. The optimal player should reflect which signals the resulting distribution favors. For now the distribution should not favor any signal so the distribution is uniform. Figure 5.2 also reflect this.

\[ S'(x_{t+1}) = \quad C'(u_t) = \]

<table>
<thead>
<tr>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( u_t = 0 )</th>
<th>( u_t = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Figure 5.2: Optimal system and optimal player test case 0.

Executing the algorithm with this data the result is as follows.

Test Case 0.
Single Player.
Parameters:
Maximum Convergence Error: 0.000001;
Minimum Converged Players: 1.000000;
Number of Players: 1;
Number of signals: 1;
Calculation Results:
Number of Iterations: 13;

First note the line Minimum Converged Players: 1.000000. This is only important for chains where there are multiple players, so this information will be disregarded in this test case. The error for maximum convergence is very small so it is safe to assume the results will be very precise.

Now comes the interpretation of the 13 iterations.

<table>
<thead>
<tr>
<th>Iteration: 0; 0</th>
<th>0.187912</th>
<th>0.812088</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.993374</td>
<td>0.006626</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration: 1; 0</th>
<th>0.244516</th>
<th>0.755484</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.978868</td>
<td>0.021132</td>
</tr>
</tbody>
</table>

(…)

Iteration: 10;
Again some parts were cut because they are not important in this report. The parts that are important are more or less the beginning and end of the iterations. As it is visible the first players getting calculated have somewhat different values. The last players are practically similar save for a very small value in the second row.

Interpreting the results, with the fact that the desired state is $x_{t+1} = 1$ in mind, the output is read like this:

- If the current state is 0, ($x_t = 0$). Then the player should emit signal 1.
- If the current state is 1, ($x_t = 1$). Then the player should emit signal 0.

The “should” in the sentences above reflects the fact that if the current state is 0 the most probable signal the player is going to emit is 1. Theoretically, if a decision had to be made between emitting signal 0 or 1, given the distribution of the calculated player, one should choose to emit signal 1.

When confronting the results with the controlled markov chain one can verify that the results make sense.

To end this test case, the optimal system will be changed. Now the most desired state is $x_{t+1} = 0$. The just showing the last player calculated below one can also see that the results make sense. Player calculated in 9 iterations:

5.2 Test Case 1 - One Player, One Signal

Test case 1 is also a single player and single signal test, the difference is that the system has four states and the signal has 3 possible outputs.

First the distribution for the system is in figure 5.3. The optimal player for this test will also have a uniform distribution so it is not going to be represented in here. The optimal system will be changed so the algorithm
\[ S(x_{t+1} | x_t, u_t) = \]

<table>
<thead>
<tr>
<th>( u_t = 0 )</th>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
<th>( x_{t+1} = 2 )</th>
<th>( x_{t+1} = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t = 0 )</td>
<td>0.2</td>
<td>0.8</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( x_t = 1 )</td>
<td>0.0</td>
<td>0.9</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>( x_t = 2 )</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>( x_t = 3 )</td>
<td>0.0</td>
<td>0.7</td>
<td>0.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( u_t = 1 )</th>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
<th>( x_{t+1} = 2 )</th>
<th>( x_{t+1} = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t = 0 )</td>
<td>0.0</td>
<td>0.3</td>
<td>0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>( x_t = 1 )</td>
<td>0.0</td>
<td>0.7</td>
<td>0.0</td>
<td>0.3</td>
</tr>
<tr>
<td>( x_t = 2 )</td>
<td>0.6</td>
<td>0.2</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>( x_t = 3 )</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( u_t = 2 )</th>
<th>( x_{t+1} = 0 )</th>
<th>( x_{t+1} = 1 )</th>
<th>( x_{t+1} = 2 )</th>
<th>( x_{t+1} = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t = 0 )</td>
<td>0.9</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( x_t = 1 )</td>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td>( x_t = 2 )</td>
<td>0.0</td>
<td>0.6</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>( x_t = 3 )</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Figure 5.3: System for test case 0.

gets executed four times. Each time the favored state of the system will be changed. First the favored state will be 0, then it will be 1 and so forth. The outputted players are bellow.

Favored state 0
Iteration: 19;
0 - 0.063767 0.007339 0.928893
1 - 0.324948 0.420704 0.254348
2 - 0.048707 0.920542 0.030750
3 - 0.191814 0.715077 0.093109

Favored state 1
Iteration: 7;
0 - 0.923123 0.066441 0.010436
1 - 0.708988 0.278357 0.012655
2 - 0.313788 0.060473 0.625739
3 - 0.715436 0.012545 0.272019

Favored state 2
Iteration: 6;
0 - 0.011553 0.987325 0.001121
1 - 0.665876 0.260554 0.073570
2 - 0.036407 0.014812 0.948781
3 - 0.025909 0.171853 0.802238

Favored state 3
Iteration: 18;
The first thing to notice is that for each favored state the number of iterations that were needed to calculate the player until convergence is achieved was different. The second thing is that it is noticeable without really analyzing all values that the calculated player are very different for each favored state. Which is a good thing.

Analyzing in detail one can notice that, again, the calculated players make sense regarding the system and optimal system. Some quick notes, suppose the favored state is $x_{t+1} = 1$, if the current state is $x_t = 1$, then the signal to emit is 0. This is observable in the first row of the second calculated player. This is also consistent with the system.

One other interesting thing is to noticed that if the current state is $x_t = 1$ and the favored state is $x_{t+1} = 0$ the transition probabilities are all 0 in the system for every signal. However the calculations show a slight preference in choosing signal $u_t = 1$, as it is observable in the second row of the first calculated player.

To explain this behavior one can look at the bigger picture and admit that perhaps this is due to the fact that if signal $u_t = 1$ is the one that is emitted, then it is probable that the system will change to state $x_{t+1} = 3$. If that happens then there will be a great probability that the next transition will change to state $x_{t+1} = 0$.

## 5.3 Test Case 2 - Two Players

This test case will show algorithm for multiple players and a single player with many signals. To do this first the test case is going to be presented with multiple players, each with a signal, along with a system, optimal system and optimal players. This controlled markov chain is going to be optimized. Then, the various players are going to be multiply so that they originate a single player with many signals. By doing that it will be possible to optimize that chain and in the end it will be possible to compare the results and draw some conclusions out of them.

First, to present the system, optimal system and optimal players. For both problems the system and optimal system will be the same. Figure 5.4 shows the system.

The optimal system will favor state $x_{t+1} = 0$. The optimal players will all have a uniform distribution. The results will be shown in the form of a mathematical matrix so they are more legible. They are in figure 5.5 for multiple player and figure figure 5.6 for a single player.
\[ S(x_{t+1} \mid x_t, u_t^1, u_t^2) = \]

\[
\begin{array}{|c|ccc|}
\hline
u_t^1 = 0, u_t^2 = 0 & x_{t+1} = 0 & x_{t+1} = 1 & x_{t+1} = 2 \\
\hline
x_t = 0 & 0.0 & 0.3 & 0.7 \\
x_t = 1 & 0.8 & 0.2 & 0.0 \\
x_t = 2 & 0.0 & 0.7 & 0.3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|ccc|}
\hline
u_t^1 = 1, u_t^2 = 0 & x_{t+1} = 0 & x_{t+1} = 1 & x_{t+1} = 2 \\
\hline
x_t = 0 & 0.1 & 0.9 & 0.0 \\
x_t = 1 & 0.9 & 0.1 & 0.0 \\
x_t = 2 & 0.0 & 0.9 & 0.1 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|ccc|}
\hline
u_t^1 = 0, u_t^2 = 1 & x_{t+1} = 0 & x_{t+1} = 1 & x_{t+1} = 2 \\
\hline
x_t = 0 & 0.0 & 0.6 & 0.4 \\
x_t = 1 & 0.6 & 0.1 & 0.3 \\
x_t = 2 & 0.8 & 0.0 & 0.2 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|ccc|}
\hline
u_t^1 = 1, u_t^2 = 1 & x_{t+1} = 0 & x_{t+1} = 1 & x_{t+1} = 2 \\
\hline
x_t = 0 & 0.0 & 0.5 & 0.5 \\
x_t = 1 & 0.4 & 0.6 & 0.0 \\
x_t = 2 & 0.5 & 0.3 & 0.2 \\
\hline
\end{array}
\]

Figure 5.4: System for test case 2.

\[
C^1(u_t^1 \mid x_t) \quad u_t = 0 \quad u_t = 1 \\
\begin{array}{|c|cc|}
\hline
x_t^1 = 0 & 0.296748 & 0.703252 \\
x_t^1 = 1 & 0.086492 & 0.913508 \\
x_t^1 = 2 & 0.999150 & 0.000850 \\
\hline
\end{array}
\]

\[
C^2(u_t^2 \mid x_t) \quad u_t = 0 \quad u_t = 1 \\
\begin{array}{|c|cc|}
\hline
x_t^2 = 0 & 0.784987 & 0.215013 \\
x_t^2 = 1 & 0.999981 & 0.000019 \\
x_t^2 = 2 & 0.000000 & 1.000000 \\
\hline
\end{array}
\]

Figure 5.5: Results for multiple players in test case 2.

The first thing to notice from both outputs is that they give similar results, which is good. They are:

- If current state is \( x_t = 0 \), signal \( u_t^1 = 1 \) and signal \( u_t^2 = 0 \).
- If current state is \( x_t = 1 \), signal \( u_t^1 = 1 \) and signal \( u_t^2 = 0 \).
- If current state is \( x_t = 2 \), signal \( u_t^1 = 0 \) and signal \( u_t^2 = 1 \).

Both methods presented very similar results. The algorithm with two player took 32 iteration to end while the algorithm with one player took 34. This would perhaps lead to infer that the bigger the system gets the more iterations it takes to optimize it regardless of the number of player. But the next test case contradicts this.
$$C(u^1_t, c^2_t | x_t) =$$

<table>
<thead>
<tr>
<th>$u^1_t, u^2_t$</th>
<th>0, 0</th>
<th>1, 0</th>
<th>0, 1</th>
<th>1, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_t = 0$</td>
<td>0.227322</td>
<td>0.465414</td>
<td>0.137719</td>
<td>0.169545</td>
</tr>
<tr>
<td>$x_t = 1$</td>
<td>0.104858</td>
<td>0.892466</td>
<td>0.002665</td>
<td>0.000012</td>
</tr>
<tr>
<td>$x_t = 2$</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.998375</td>
<td>0.001625</td>
</tr>
</tbody>
</table>

Figure 5.6: Results for single player in test case 2.

<table>
<thead>
<tr>
<th>$u^1_t = 1, u^2_t = 1, u^3_t = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_t + 1 = 0$</td>
</tr>
<tr>
<td>$x_t = 0$</td>
</tr>
<tr>
<td>$x_t = 1$</td>
</tr>
<tr>
<td>$x_t = 2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$u^1_t = 1, u^2_t = 1, u^3_t = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_t + 1 = 0$</td>
</tr>
<tr>
<td>$x_t = 0$</td>
</tr>
<tr>
<td>$x_t = 1$</td>
</tr>
<tr>
<td>$x_t = 2$</td>
</tr>
</tbody>
</table>

Figure 5.7: Special case in the distribution.

5.4 Test Case 3 - Three Players

This last test case is composed of a system that has three states and three signals. Like test case 2, the system will be first optimize for three separate players and then it will be optimize for one player with three signals.

Like in the other test cases the optimal players all have a uniform distribution.

To represent the system and the results it would take a large amount of number. Because of that is was decided that that information would not been shown in this report. To see the results simply execute the test case in the submitted project.

The results are similar in both optimization. This proves once again that algorithm is working correctly. But what is interesting to notice is that it only took 5 iterations to calculate the player in the situation where there is only one player and multiple signals and 6 iteration in the situation where there are multiple signals.

These results lead to conclude that the size of the system doesn’t exactly matter when optimizing it. The why of a big system taking less iterations to optimize then a smaller system could be due to a number of factors. One possible explanation is the fact that there is a part of the system where the distribution in like in figure 5.7.

This means that when the signals are $u^1_t = 1, u^2_t = 1, u^3_t = 0$ then if
the state $x_t = 0$ the transition will be for sure $x_{t+1} = 1$, if the transition is $x_t = 1$ the transition will be $x_{t+1} = 2$, and so fourth. This can be compared as a cycle in these two situation where the signals have that configuration.

To test is this is the cause, each value of the distribution in the test case were simply change to $\frac{1}{3}$.

By doing so the results changed, of course, but what it important is that the number of iterations went up. Now to optimize this system for multiple player it took 16 iteration. For one player with many signals it took 11.
Chapter 6

Final Remarks

This project dealt with controlled Markov chains. It started by showing what the relation between controlled Markov chains and regular Markov chains were and, of course, talked about the players.

The players were introduced as either being a single player with one signal, a single player with multiple signals or multiple player with a single signal each. Something that would be interesting would be to explore what would happen if there were say two players each with two signals. Mathematically this would have the distributions below.

\[ S(x_{t+1} \mid x_t, u_1, u_2, u_3, u_4), \]
\[ C_a(u_1, u_2 \mid x_t), \]
\[ C_b(u_3, u_4 \mid x_t). \]

This idea appeared too late so it was not tested and it is probable that the algorithms can’t handle the data formatted like this. On the other hand a model like this may not be good for anything and it’s study would be unnecessary. But again, further studies on the matter had to be taken.

This project shown that the library developed in the first part of this grant, the Toolbox for Probability Calculus and Optimization, is very versatile. Every implementation in this project was done using that library which showed how easy it is to used the library to solve problems. Like this it is proven that the project from the first grant was a success in it’s objective of offering a solution to work with probability distributions with \( n \in \mathbb{N} \) random variables and to implement various algorithms regarding such objectives has optimization.

The algorithms implemented were very interesting in their objective to optimize controlled Markov chains. This optimization can be used to make decisions in situation that can be formalized with this kind of chains.

In this project a different way to rewrite the expressions shown in [2] was proposed. That way, as it is said in chapter 3, it was possible to implement
the algorithms using the library in a fairy simple way. Methods to stop the algorithms were also proposed. Finding a way to do just that was very interesting since the article [2] didn’t mention a specific method.

The various test cases that were submitted in this project along with the ones that are actually in the report showed not only that the algorithms are working but also showed what is there to expect of the results in a verity of situations.

The last remark about this project is that there was one situation that didn’t got to be explored. That was the situation where the players in a controlled markov chain communicate through a noisy channel. In this situation the equivalents systems, that is, what a player perceives as being the system along with the other players would be very different then the equivalent systems described in chapter 3. It would be interesting to see the results of the optimization of this kind of chains compared to the ones presented in this report.
Bibliography


