OPTIMIZATION OF BATCH PROCESSES BY ARTIFICIAL INTELLIGENCE ALGORITHM

By

Abdelrahman Saeed Ibrahim Abdelkader

A Thesis Submitted to the Faculty of Engineering at Cairo University
in partial fulfillment of the requirements for the degree of

Master of Science

In

Chemical Engineering

Faculty of Engineering

Cairo University

2019

OPTIMIZATION OF BATCH PROCESSES BY ARTIFICIAL INTELLIGENCE ALGORITHM

By

Abdelrahman Saeed Ibrahim Abdelkader

A Thesis Submitted to the Faculty of Engineering at Cairo University
in partial fulfillment of the requirements for the degree of

Master of Science

In

Chemical Engineering

Prof. Ahmed Soliman

Professor, Department of Chemical Engineering, Faculty of Engineering, Cairo University

Prof. Seif Eldeen Fateen

Professor, Department of Environmental Engineering, Faculty of Engineering, Zewail University

Faculty of Engineering

Cairo University
2019

Engineer's Name:	Abdelrahman Saeed Ibrahim Abdelkader
Date of Birth:	10 Oct. 1989
Nationality:	Egyptian
E-mail:	abdoelrock@hotmail.com
Phone:	+20/ 1016317097
Address:	Egypt – Alexandria City
Registration Date:	2015
Awarding Date:	2019
Degree:	Master of Science
Department:	Chemical Engineering

Supervisors: Prof. Ahmed Soliman
Prof. Saif Eldeen Fateen

Examiners:

Prof. Mohamed Mustafa Saleh	(External examiner)
Prof. Mahmoud Refaay	(Internal examiner)
Prof. Ahmed Soliman	(Thesis Main Advisor)
Prof. Seif el deen Fateen	(Advisor)

Title of Thesis:

Optimal Control of Batch Processes Via An Enhanced Robust Q-Learning Method.

Keywords:

Optimal control; Artificial intelligence; Q-learning; Batch processes; Machine Learning

Summary:

In optimizing the batch reactor, we face a problem which is that if something goes wrong during the optimum path and we won't know whether to continue on the old optimum path or calculate a new one. The new proposed method solves this issue by mapping the states to the optimum action which automatically determines the new optimum path.

بسم الله الرحمن الرحيم

Acknowledgements

All grace and thanks to **Allah** for his protection and guidance.

I would like to thank my supervisors for their patience and support throughout the project and for their technical guidance.

Special thanks to **Dr. Jie Zhang** from Newcastle university for his remarks and guidance.

Finally, the utmost pleasure is to thank my parents, my wife, my brother and sisters. I wouldn't have done it without your support.

And this work is dedicated to my beloved baby daughter **Durra**.

Table of Contents

List of figures

List of tables

Abstract

Dynamic optimization of nonlinear chemical systems -- such as batch reactors -- should be applied online, and the suitable control taken should be according to the current state of the system rather than the current time instant.

Recent state of the art methods applies the control based on the current time instant only. This is not suitable for most cases, as it is not robust to possible changes in the system. This thesis proposes an enhanced robust Q-Learning method to conduct robust online optimization of a batch reactor.

In this thesis the Q-Learning method is applied on a simple batch reactor, in order to show the effectiveness of the newly proposed method its results are compared to the enhanced robust Q-learning method where there is a significant improvement was obtained.

The main advantage of the Q-learning method or the proposed method is that it can accommodate for unplanned changes during the process by changing the control on the process.

We try to maximize the final product obtained or meet certain specifications of the products (i.e. Minimize side products).

This could be done using several ways, the advantage of our proposed method that it can overcome sudden changes during the reaction.

1. INTRODUCTION

1.1. Batch process

Many important industries use batch and semi-batch processes and are of significant position in the chemicals industry. Examples of these industries include variation of chemicals, pharmaceutical products like in the production of API (Active pharmaceutical ingredients), cosmetics industry, and certain types of polymers for both step growth and chain growth reactions are manufactured in batch operations [1].

we use batch operations usually when the making sizes or the amount dealt with are little examples of batch operations are batch distillation, batch reactors and batch crystallization. for example, when there are no mass production batch operations are used or when the waste needs to be treated by the batch operations such as batch distillation is little. also, its used when isolation is vital for aims of sterility or safety, and when the constituents involved are hard to handle. for instance, when the purity of the product is a vital issue batch process is used and when there are constraints on the temperature of the process. There has been a new attention in batch processing as a result of the market requirements and the flexibility in the production of materials using batch processes. [2]

- **Batch process optimization significance:**

The optimization of batch processes has appealed attention in latest years because, in the aspect of increased competition, it is the best option for dropping fabrication expenses, refining manufactured goods value, meeting safety necessities and ecological rules. Typically optimization finds the best design and operation parameters to get the maximum profit while maintaining the specified constraints and meeting market and regulations demands [3]

- **Batch Process operation types:**

Batch processes, all the reactants are loaded in a container firstly and handled according to a pre-known path of control throughout which no compounds are added or removed. [4]

Semi-batch processes, a reactant may be loaded with no product deduction, or a product may be removed with no reactant loading, or a mixture of both. [5]. A batch reactor is shown in figure 1-1 [6].

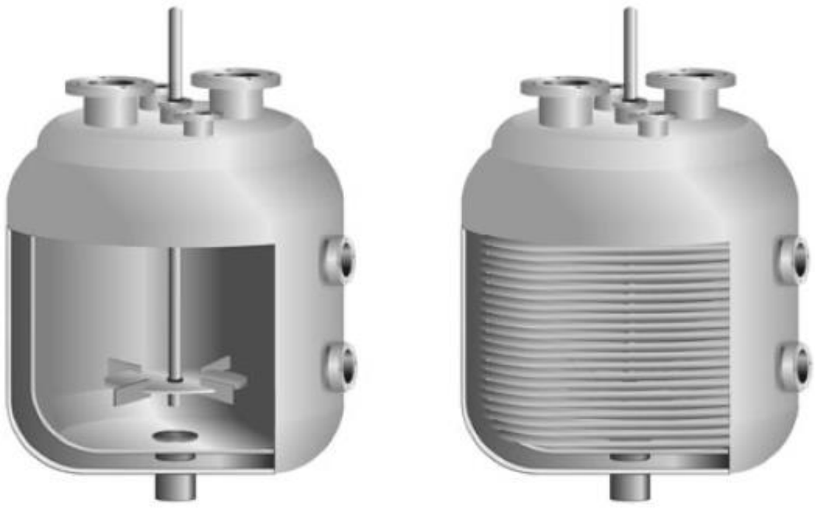


Figure 1-1 Batch Reactor

(Source:[6] page: 21)

1.2. Optimal Control and Dynamic Programming

1.2.1 Optimal control:

Optimization is a very wanted technique that we use daily in our life such as in finding the shortest route to a certain destination while avoiding heavy traffic or when purchasing an item, we find the best item that satisfy our needs while being reasonable in price. The topic of optimization is very broad it can be classified according to various ways or means so it could be solved algebraically or using geometric techniques, its target or the desired optimum could be single or multiple, its nature could be deterministic which means no probability or uncertainty is found or it could be stochastic or probabilistic. these all types of optimization could be solved using various techniques one of them is the calculus of variations that was the foundation branch of dynamic optimization that we will use in solving the optimization problem of batch reactor as we will discuss later it's noted that the calculus of variations is one trivial part of the large image of the optimization field, and it formulates the base for our study of optimal control schemes. [7]

Moreover, optimization can be categorized as static optimization and dynamic optimization as shown in figure 1-3:

A. **Static Optimization** is about finding the optimum in a plant in steady state conditions, which means the system variables are not varying with time which is also called non-transient process. The plant is then defined by algebraic equations that describe the system. Methods used are normal calculus, Lagrange multipliers, simplex method, linear and nonlinear programming and numerical methods. [8]

We optimize the system using a single optimum value. Example: finding the optimum thickness of insulation for cost reduction whereas the thickness increase the heat loss decrease but the cost of insulation increase and the optimum is the lowest point in the total cost curve as shown in figure 1-2. [9]

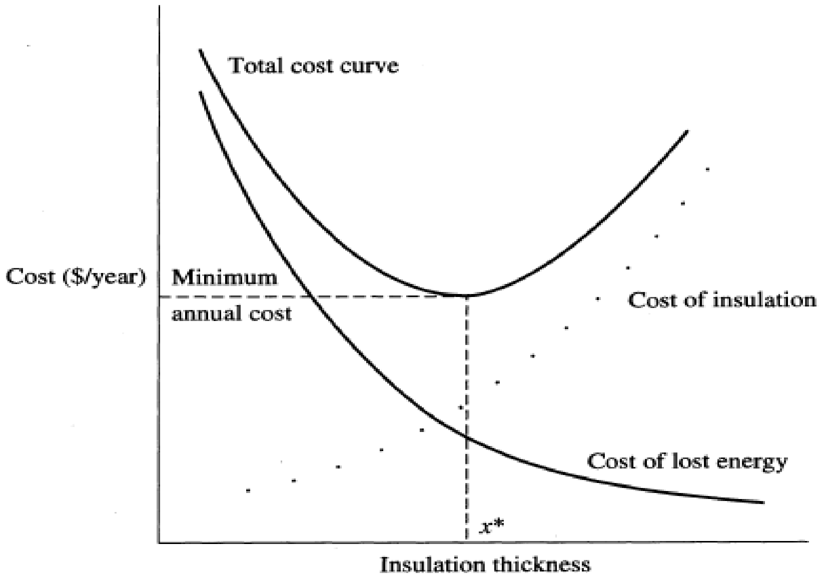


Figure 1-2 Optimum insulation thickness

(Source:[9] page: 10)

B. **<u>Dynamic Optimization:</u>** Deals with the optimization of plants under dynamic circumstances, which means the system variables are varying with respect to time, it's also called transient processes and thus the time is in the system description which is basically differential algebraic equations. Methods used are search techniques, dynamic programming, calculus of variations and Pontryagin principle. This branch of optimization is called optimal control.

Some of the solution methods can be done using IPO (Interior point optimization), SQP (Sequential quadratic programming), and GE (Genetic algorithm). But the disadvantage of these methods are that they can’t find the global optimum and that they don’t respond to changes in the system.[10]

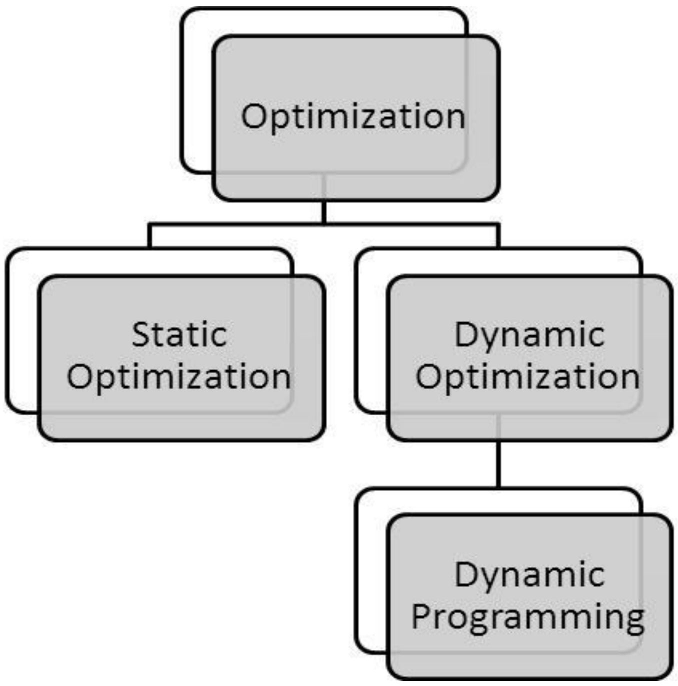


Figure 1-3 Types of optimization

The main aim of dynamic optimization is to find the optimum path or trajectory that maximizes or minimizes a certain value or equation called the performance index while at the same time maintaining a certain constraint whether the constraints are on the optimum path or on the system. Discussing figure 1-4 [11] , the optimum path we are trying to find is denoted as u*(t) (* shows optimal condition) the system is describes by state x and the output y is the resulting state from optimum value u, also y could be called the next state. [12]

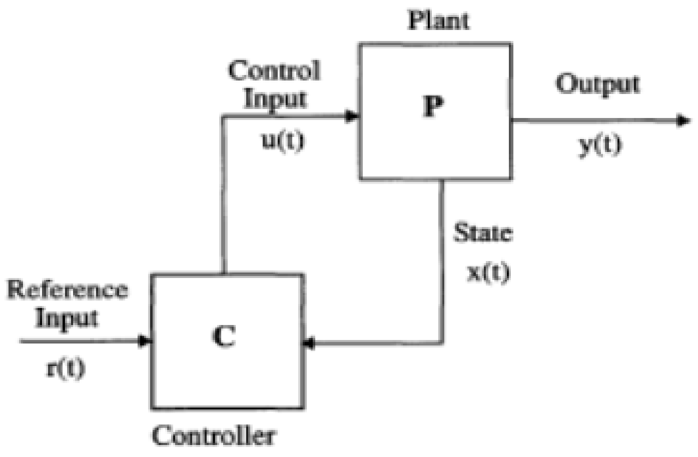


Figure 1-4 Plant control

(Source:[11] page: 3)

1.2.1.1. Dynamic optimization problem consists of:

1. A mathematical model or differential equations describing the plant or system that will be optimized.
2. A scalar value or an equation that we need to maximize or minimize.
3. Constraints on the optimum path and the plant or system itself.

Traditional control design methods which are called classical control have been effectively used on systems that are simple and uses controllers such as P (Proportional), PI (Proportional and Integral) and PID(Proportional Integral Derivative) and study their response and abilities in controlling a process. [13]

In modern control theories we use the concept performance index that is basically our target to reach and it may take several forms as described below

1.2.1.2. Examples for the performance indices:

1. Performance Index for minimum time problem [14]:

We basically want to reduce the time to reach a final or terminal state, we will deal with this kind of performance index in the third case study. The matching performance index (PI) is

$$J = \int_{t_0}^{t_f} dt = t_f - t_0 = t^* \qquad \textbf{(1-1)}$$

2. Performance Index for minimum fuel consumption problem [15]:

Assume a rocket or a space craft to be launched. Let u(t) be the thrust of a rocket engine and the thrust is directly proportional to the fuel consumption so as the thrust increase the fuel depletion increase. so to decrease the fuel consumption we have to optimize the thrust its performance index is:

$$J = \int_{t_0}^{t_f} |u(t)| dt \qquad \textbf{(1-2)}$$

For several controls when we have multiple factors affecting the thrust, we may write it as

$$J = \int_{t_0}^{t_f} \sum_{i=1}^{m} R_i \, |u(t)| dt \qquad \textbf{(1-3)}$$

Where R:is a weighting factor.

3. Performance Index for minimum power consumption problem [16]:

The total power consumption in an electrical grid system is equal to: $\sum_{i=1}^{m} \boldsymbol{R_i u_i^2}$ 1-4) (where,ri is the resistance of the ith loop and u_i is the current in the ith loop) Then, to minimize the total power spent, we have a performance measure as

$$\boldsymbol{J = \int_{t_0}^{t_f} \sum_{i=1}^{m} R_i u_i^2 dt} \qquad \textbf{(1-5)}$$

Or in general,

$$\boldsymbol{J = \int_{t_0}^{t_f} u' R\, u(t) dt} \qquad \textbf{(1-6)}$$

Where, R is a positive definite matrix similarly, we can work on reduction of the integral of the squared error of a chasing system where we try to minimize the distance between the current position and a desired position. We then have,

$$\boldsymbol{J = \int_{t_0}^{t_f} X'\, Q\, X(t) dt} \qquad \textbf{(1-7)}$$

Where, X_d (t) is the wanted value, x_a (t) is the real value, and x (t) = x_a (t) - X_d (t), is the inaccuracy. Here, Q is a weighting matrix.

4. Performance Index in general [17]:

Combine the above equations, we have a performance index in Broad shape that can be used for all of the purposes as:

$$J = X'(t_f)\, F\, x(t_f) + \int_{t_0}^{t_f} [X'\, Q\, X(t) + u' R\, u(t)]\, dt \qquad \textbf{(1-8)}$$

Or

$$J = S(x(t_f), t_f) + \int_{t_0}^{t_f} V(x(t), u(t), t) dt \qquad \textbf{(1-9)}$$

Where, R is a positive definite matrix, and Q and F are positive semi definite matrices.

So to summarize this section dynamic optimization problem is to locate the optimal control u*(t) (* indicates optimal cost) which causes the linear plant as shown in the equation below.[15]

$$(\dot{x}(t) = Ax(t) + Bu(t)) \qquad \textbf{(1-10)}$$

To find the optimum path x*(t) that optimizes (minimizes or maximizes) a performance index

$$J = X'(t_f)\, F\, x(t_f) + \int_{t_0}^{t_f} [X'\, Q\, X(t) + u' R\, u(t)]\, dt$$

Or which causes the nonlinear system described by the equation below

$$\dot{x}(t) = f(x(t), u(t), t) \qquad \textbf{(1-11)}$$

To give the state x*(t) that optimizes the general performance index

$$J = S(x(t_f), t_f) + \int_{t_0}^{t_f} V(x(t), u(t), t)dt$$

With several constraints on the action variable u(t) or the state variables x(t)The end time tf may be constant, or open ,and the final target may be completely or partially constant or variable.

1.2.2 Dynamic Programming:

As mentioned before if we have a dynamic or transient system or plant and a cost function or a performance index that we try to maximize it could be solved in various ways one is which the dynamic programming (DP). The method is named dynamic programming since it is a method depends on compute "programming" and appropriate for "dynamic" plants. [18]

The idea behind dynamic programming is to divide or perform discretization on the system or the plant where it will consist of smaller problems in which at each step we will choose an action or control from a number of allowed controls. The main idea of DP depends on a simple instinctive idea named principle of optimality.

1.1.2.1. Principle of Optimality:

An illustrating example is to set a basic multi period choice optimization process as shown in figure 1-5 [19] Here, let the optimizing cost function performance index for the section AC be J_{AC} and for the section CB be J_{CB}.

$$J_{AB} = J_{AC} + J_{CB} \qquad \textbf{(1-12)}$$

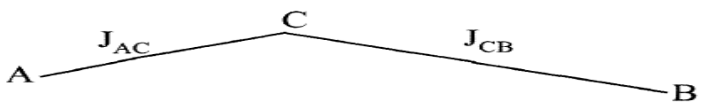


Figure 1-5 Dynamic programming illustration

(Source:[19] page: 20)

Then the optimizing cost performance index for the complete section AB is J_{AB} That is, if we have the optimum performance index for the section CB and we found the optimum performance index for AC then we can easily find the optimum total performance index AB by adding those of AC and CB, we by this can divide a problem into smaller sub problems and find the optimum total of the whole system by solving the smaller sub problems. This noticeable appearing characteristic is named the principle of optimality (PO) and declared in [18][20]

To be more obvious Dynamic programming is an algorithm design technique that be able to be used when the result of a problem may be viewed as the product of a series of actions.

Table 1-1 Advantages and Disadvantages of dynamic programming

Advantages of dynamic programming	Disadvantages of dynamic programming
• Providing insight to the problem due to the breaking down a complex problem into a serious of simpler problems.	• Consumes large memory. • Difficult to implement. • They don't respond to changes in the system.
• Global optimization.	• Curse of dimensionality occurs with a problem having multiple states. Will be very difficult to solve.

1.3. Problem Formulation for batch reactors

Let's consider the case of a single value performance index or scalar performance index and single action to be found as the optimum path as temperature. Since DP can be applied to systems with a large number of control variables it's easily applied in our case of single action or control variable. Let us therefore consider the system described by the differential equation:[21]

$$\frac{dX}{dt} = f(x, u) \qquad \textbf{(1-13)}$$

With x(0) given, where x is an (n x 1) state vector and u is a scalar control, bounded by:

$$u_{min} \leq u \leq u_{max} \qquad \textbf{(1-14)}$$

The performance index is a scalar function of the state at the final time t_f given by:

$$I(x(0), t_f) = \varphi(x(t_f) \qquad \textbf{(1-15)}$$

This problem is a fixed time problem and the final time is known. The problem is to find the optimum control u in the given allowed time that will maximize the performance index in Equation (1-15). her we will discretize the action or the control variable over the interval $t_k \leq t \leq t_{k+1}$ so that we have a constant control over each interval: $u(t) = u(t_k)$ to be more clear we need to find the set of control values u(0), u(t),......u(t_f-1) that maximizes the performance index in Equation (1-15).

1.3.1. Batch Reactor Example: Using Dynamic Programming [22]

We chose the nonlinear batch with the second order reaction scheme is A → P → C, and the problem is to find the best temperature policy to maximize the yield of component P after a batch time of 1 h. The equations describing the reactor are:

$$\frac{d[A]}{dt} = -4000\, e^{\left(-\frac{2500}{T}\right)} [A]^2 \qquad \textbf{(1-16)}$$

$$\frac{d[A]}{dt} = 4000\, e^{\left(-\frac{2500}{T}\right)}[A]^2 - 6.2 * 10^5\; e^{\left(-\frac{-5000}{T}\right)}[P] \qquad \textbf{1-17)}$$

<u>Where:</u>

[A] is A mole fraction,

[P] is P mole fraction,

t is the time,

T is the temperature in (K).

With initial conditions $[A]_0=1,[P]_0=0$

With constraints on the temperature ($298< T < 398$)

The performance index to be maximized is the final concentration of P. $I([P]_f)$

Where the final time =1.0 hr

Dynamic programming: Solution key idea

1. We divide the time interval in to equal time steps.
2. We generate states using actions or controls, where a state is a value describing the system in our case the mole fraction is the state and an action is a value that causes the system to change from a condition to another in our case is the temperature.
3. We start from the states from the beginning of the last step, and solve for all possible actions which in our case is from 298 K till 398 K. The maximum final mole fraction and the action that caused it to reach it will be chosen.
4. We proceed to the step before it and solve again till the beginning of the last step with all the possible actions, the action for the final step that produce the closest state to the just generated state will be chosen. Then we see which will produce the highest final mole fraction and store the action for this step.
5. Repeat till you reach the initial point and then reduce the region of states generated and repeat the steps.

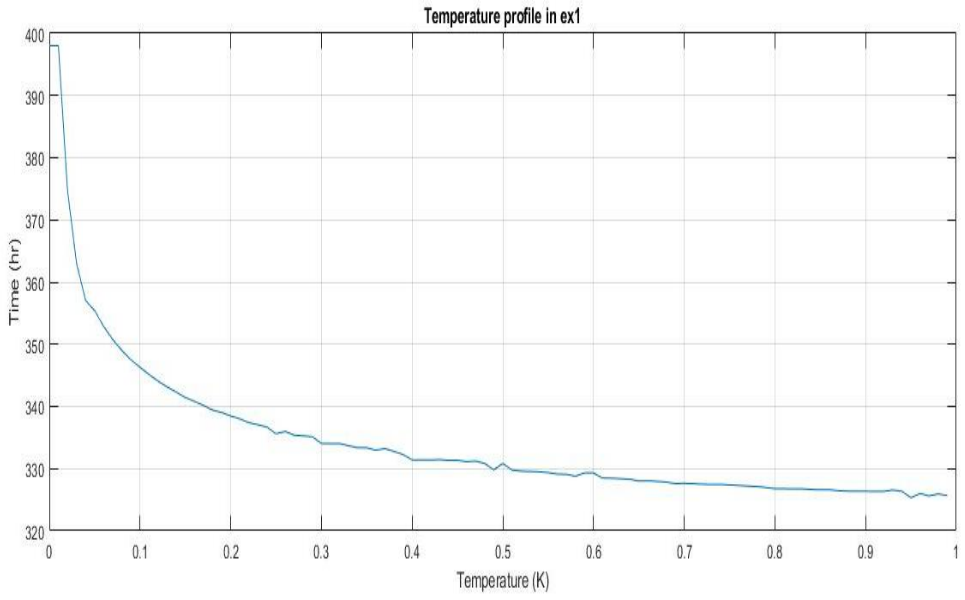


Figure 1-6 Solution of dynamic programming example

Results: The final mole fraction of P was: 0.6107 and the optimum path for the temperature that will make us reach this maximum is shown in Figure 1-6. The Matlab solution code is shown in the appendix.

2. LITERATURE REVIEW

2.1. Overview

In this section we will review five different subjects but related to our work the subjects are neural networks, support vector machines, reinforced learning, Q-learning and finally Markov decision process.

In the first part the neural networks the intelligent technique will be overviewed and then its description will be discussed and finally its capabilities will be reviewed.

In the Second part the support vector machines (SVM) main features will be discussed also separating the data methods will be presented, finally the advantages and disadvantages of the support vector machines will be listed.

In the third part is about the reinforced learning method, at first the method will be explained, then the parts of reinforced learning will be mentioned, finally the reinforced learning concepts will be illustrated by various examples.

In the fourth part the method that our thesis is about is explained, where the algorithm is discussed and illustrated by a simple example.

In the fifth and final part we will discuss the Markov decision process, the algorithm will be listed alongside with its concepts and definition.

2.2. Neural Networks

2.2.1. Overview

Many discussions have been aroused to compare the human mind to artificial neural networks (ANN), but the reality is that the ANN is no near comparison to the human mind as our mind is capable of doing impressive things and perform activities that neither the ANN nor the artificial intelligence have come near to reach. Our mind can do very complex procedures in a matter of a millisecond with high precision and excellence, few of these activities is mentioned below that shows us the greatness of the human mind [23]:

- In baseball game the catcher catches the ball that comes at him at great velocity without knowing what is that velocity is and what is the accurate direction is, this is performed by him in less than a second.
- In tennis the player predicts the trajectory of a ball that travels at very high velocity and responds to the direction of the ball with quick body movement to strike it back with high precision.
- Being able to catch a dropped item in a very quick matter to prevent it from crashing.
- The amazing human memory that can recall thousands of names and assign each name to the corresponding face and figure.
- Doing acrobats and ballet and keeping balance while doing them.
- The ability to learn more than one language and know how to speak, write and understand others speaking it.
- Composing music and writing songs and alot of other daily activities we do for granted.

The truth about the brain is that it learns these capabilities with time and need knowledge and practice. Infants for example can find some of these tasks nearly impossible such as walking or speaking but as they progress in life they learn how to perform the capabilities and much more complex ones, That become very easy to do even without putting efforts in it.

The branch of artificial intelligence (AI) was encouraged by the learning capabilities of the human mind, Scientists tries to mimic human behavior in to a robot or in process control. But some the tasks that we take for granted by the human mind such as pumping the blood by the heart through the veins or breathing or maintaining sugar in the body seems impossible to do by AI.

These human activities or even animals activities have been the center of interest by the scientists since early 1940, That's why they developed ANN to help perform they tasks.

2.2.2. Artificial Neural Network Description

ANN is a technique used in the AI field to be taught instead of usual programming to do various tasks most commonly approximation. It analyses data given to it and learns from it to be able to predict a new set of data when a related date is being entered to the NN. NN learns from the data by finding the patterns in it and the relationship among the data. Where it classifies the data in to three sets, which are training, testing and validation.

Training data set where it matches the given outputs to the inputs and finds a relation that describes the input - output data pairs. while it uses the testing data set to find whether the relation obtained earlier is doing well or needs improvements. Finally the validation data set to do the final check and validate the model.

Rosenblatt in 1959 [24] invented the perceptron a single neuron NN his aim was to mimic the human neuron found in the human brain which. This invention later was taught to play chess and defeated the world champion in chess back at that time.

When seeing the human neuron, the perceptron is a resemblance of the human neuron. So in the real neuron shown in figure 2-1 an electrical signal is produced from the axons and passes through the dendrite where it's received at the synapses which regulates the signal where it is responsible for the decision making process [25]. This is the same in the perceptron which receives a numerical value and it regulates its value through something called weights and transfer function.

Figure 2-2 and figure 2-3 illustrates an example of an artificial neuron in which the links to the left of the weights stand for the inputs that may approach from the links of supplementary perceptrons. once the various inputs, are added, an activation or transfer function is applied to match the inputs to assure that the conditions of the problem being answered [26].

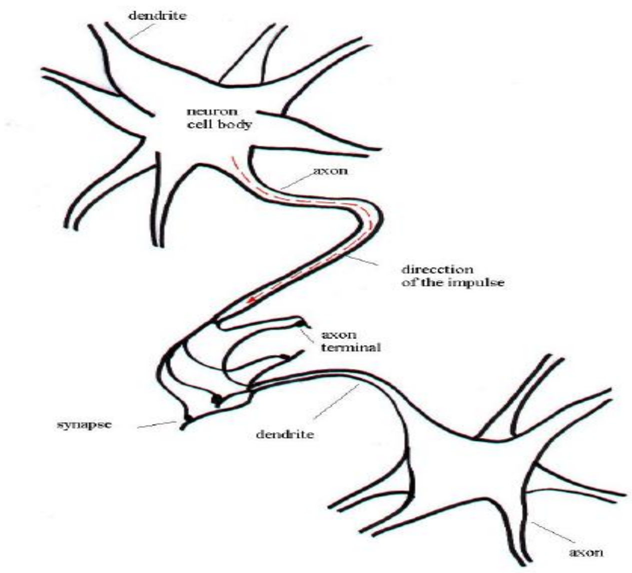


Figure 2-1 Natural neuron of human mind

(Source:[27] page: 7)

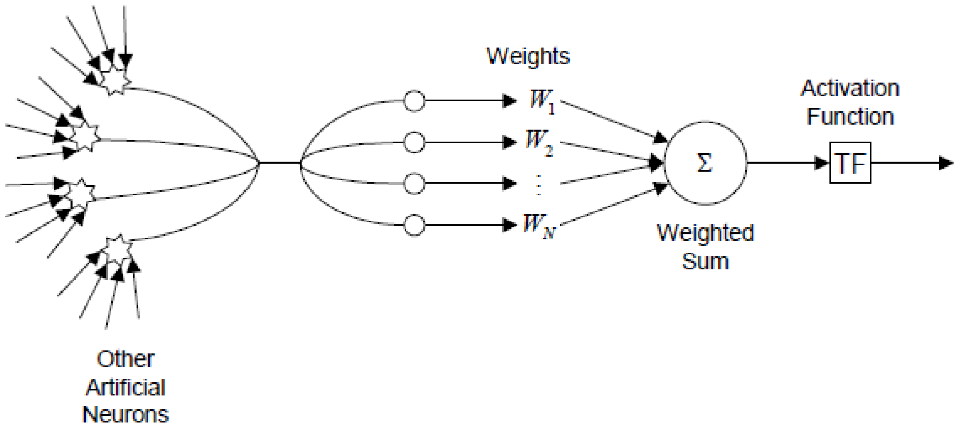


Figure 2-2 Artificial neural network

(Source:[27]   page: 16)

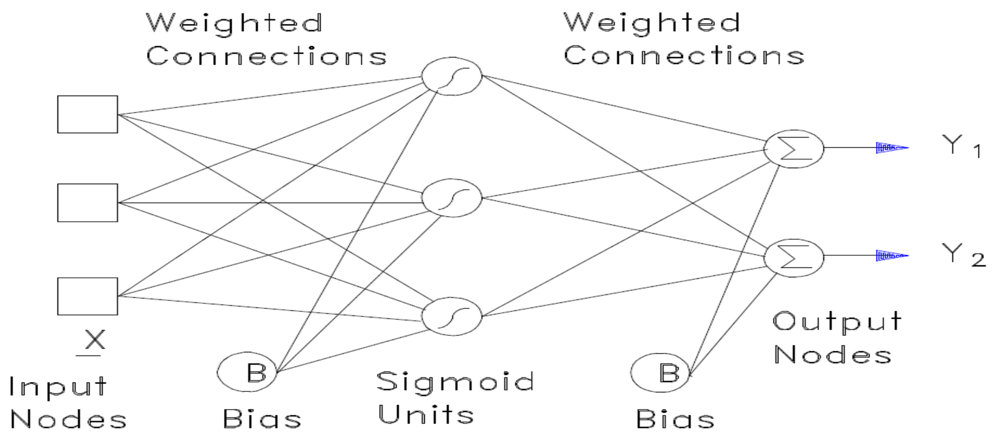


Figure 2-3 Artificial neural network detailed

(Source:[28] **page:3)**

Transfer functions is diverse it could be continous or discrete. Also it can be classified in to linear, sigmoid, tangent sigmoid. it's up to the NN designer to choose the appropriate transfer function, [29] contains extra information about transfer functions.

2.2.3. Artificial Neural Networks Capabilities

NN have various strong capabilities it could be classified in to four man categories, firstly is the approximation, NN is considered a universal approximator which can resemble any mathematical function. in approximation the input - output pairs are entered to the network and it finds the relation between them so when an unused input is entered it predicts its corresponding output.

Secondly it can be used in pattern and image recognition where Google images uses a type of neural networks call convolution NN where it finds the images that are near or matching the current image

Thirdly, Its used in finding the prediction of a series for example if we entered a series of numbers that are related with a certain sequence it will predict the next value or next few values

Finally it's used in pattern classification for instance if we entered to a trained NN the characteristics of a tumor it can find whether the tumor is cancerous or not depending on its characteristics [30].

a few references are listed in [31]–[34] that discusses the capabilities and design of NN. A chain of potential applications for NN were presented. branches included where: chemical engineering, economics, aerospace, warfare, electronics, medicine and alot more applications [35].

2.3. Support vector machines

2.3.1. Overview

Support vector machines (SVM), is a type of techniques that has been widely accepted in artificial intelligence branch of science. its main idea is that for instance we have two types of data. and a new data sample is presented and we want to know whether this data belongs to the first group or the second. The SVM answers this problems and assigns the new data to its correct classification. This is known as pattern classification

Cortes &Vapnik [36] established SVMs see figure 2-4 for double classification. Their method may be approximately outlined as follows:

First of all let's consider the above example where two sets of data are presented the SVM separates the data according to their classes by using something called the hyper plane which is a plane is placed in the position that maximizes the margin or distance between the two classes. the data on the sides of the border is named support vectors.

But commonly the data is not clear and sometimes lays on the wrong side of the hyper plane these are called overlapping classes they are moved down the hyper plane to decrease their effect on the classification problem solution. This movement places the support vector on a margin that is known for the soft margin**.**

Sometimes the support vectors are nonlinear which mean we can't split the data using a linear hyper plane instead we use a larger dimension to split the data correctly this is named as kernelling.

The algorithm used to solve this problem and perform the separation is a quadratic optimization problem. SVM is the program that performs all of these duties [37].

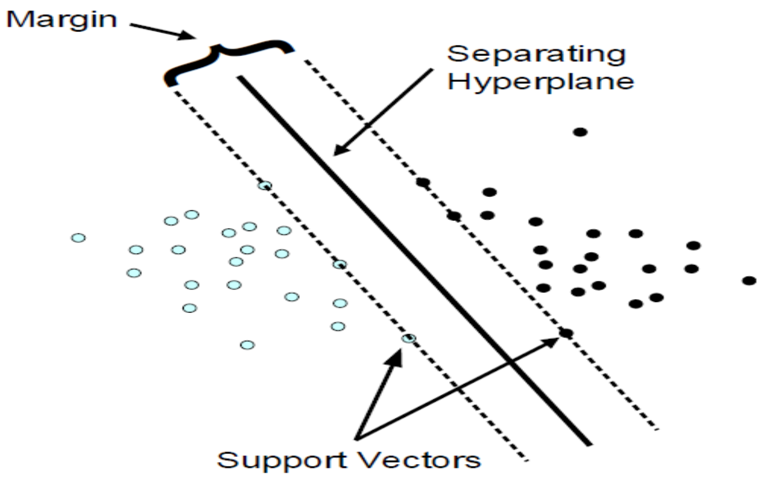


Figure 2-4 Support vector machine

(Source:[38]**)**

2.3.2. Hyper planes

Hyper planes is the main technique used in the SVM, the simplest is the line separating linear data. Where the larger the distance between the hyper plane and the data the better it will be in finding the correct classification of the data.

We need to allocate the hyper plane to find the best separation between data this is done by increasing the margin between the hyper plane and the data [38][39].

A dataset will often look more like the untidy globes below in figure 2-5 which emphasis a linearly non separate dataset.

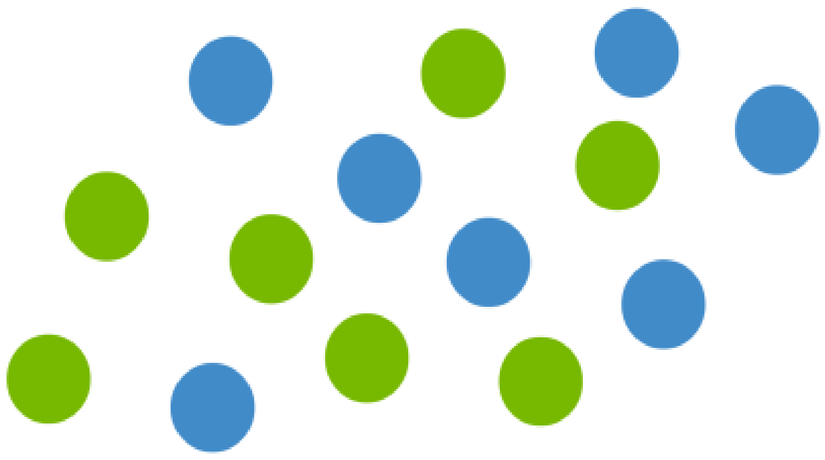


Figure 2-5 Non separate data set

(Source:[40]**)**

In order to categorize a dataset like the one overhead it's essential to change from a two dimension vision of the data to a three dimension vision. Clarification of this is feasible with additional basic example. Visualize that our two groups of colored globes above are laying on a sheet and this sheet is raised unexpectedly, releasing the globes into the atmosphere. While the globes are up in the atmosphere, you use the sheet to distinct them. This 'releasing' of the globes models the mapping of data into a greater dimension. This is identified as kernelling.

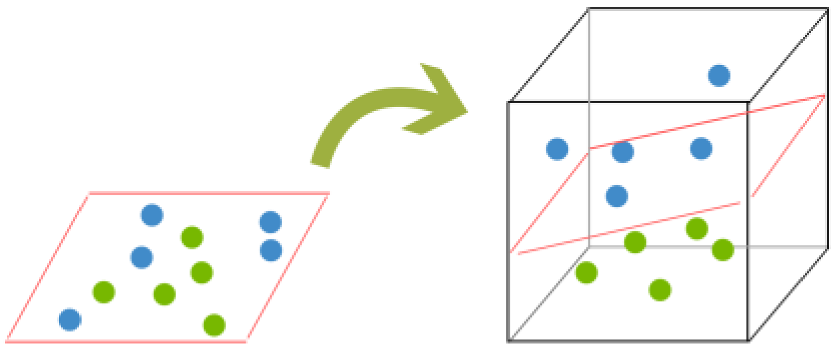


Figure 2-6 Higher dimension transformation to separate the data

(Source:[40]**)**

Since we are currently in three dimensions, our hyper plane can no more be a line. It have now to be a plane as presented in the example above in figure 2-6. The idea is that the data will carry on to be matched into higher and higher dimensions until a hyper plane can be shaped to separate it[41].

2.3.3. Pros & Cons of Support Vector Machines

Pros

- Precision.
- Works fine on lesser cleaner datasets.
- It can be extra effective since it uses a subset of training samples.

Cons

- Isn't suitable to greater datasets as the training interval with SVMs can be great.
- Fewer effective on noisier datasets with overlapping samples.

2.3.4. SVM Uses

SVM is used for text categorizing responsibilities such as classification task, spotting junk and sentiment analysis. It is more over usually used for picture recognition tasks, carrying out mainly well in aspect-based recognition and color-based classification. SVM also shows an important part in many branches of handwritten digit recognition, such as mail automatic facilities.

2.4. Reinforced Learning

2.4.1. Overview

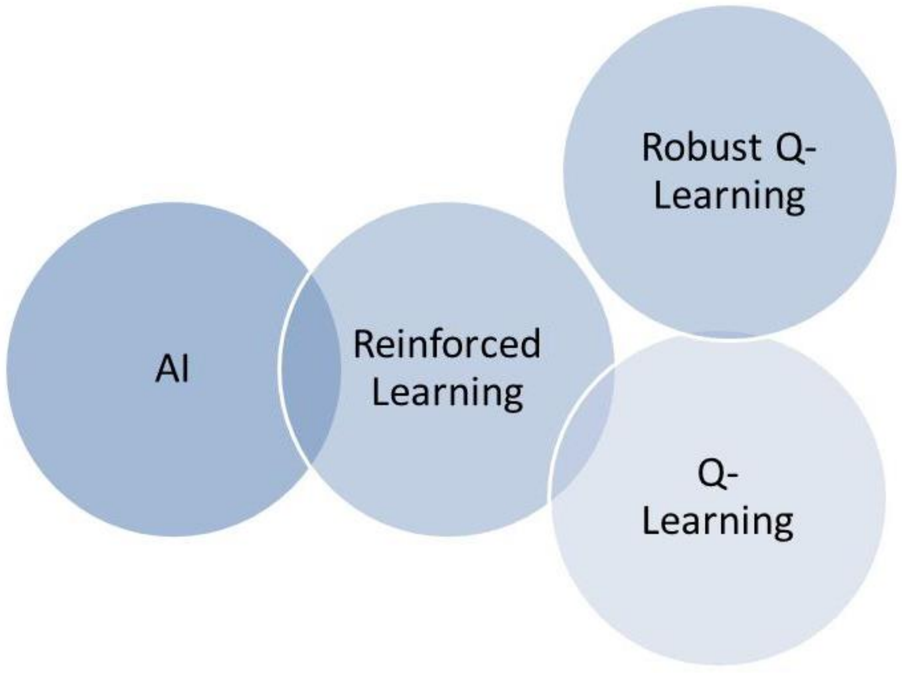


Figure 2-7 Artificial intelligence overview

Many present problems are hard to solve due to its non linearity some of them include flight control, process control and aerospace applications these problems require an appropriate computer software to be solved.

Some of these problems are still not solvable not as a result of small memory or slow computers but because there is no appropriate method to solve these problems. let's consider the case that the computer can learn to solve the problem by trial and error that would solve the issue as the computer can be taught and then allowed to solve these complex problems online.

Reinforced learning is a method that can be used to teach the computer to solve complex problems by trial and error, it uses two branches of science, one of which is the dynamic programming discussed before and the other is supervised learning. that we will discuss now [42]. The overview of artificial intelligence is shown in figure 2-7.

Supervised learning in summery is a method that teaches a function approximator like neural network or a support vector machine to model the relation between its inputs and outputs. However, Supervised learning require an input - output pairs to teach its technique how to model them in other words supervised learning needs questions and its corresponding correct answers. For instance, if we have an image of a tank, supervised learning won't be able to answer the question of whether if there is a tank in the picture or not. It needs at first to be taught using several images containing tanks and other not containing tanks. and from these images it will learn to answer the question [43].

Some people gets confused between neural networks and Reinforcement learning the fact is neural networks aren't kind of a reinforced learning nor the reinforced learning is a substitute for it. Reinforced learning as stated before it sums up the areas of dynamic programming and supervised learning to give us strong machine learning systems. it's shown in figure 2-8 [44].

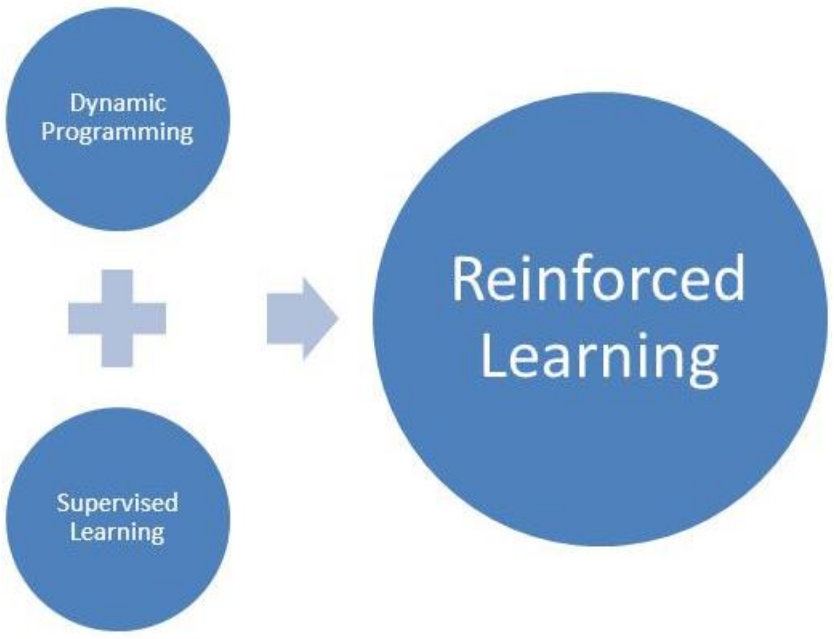


Figure 2-8 Reinforced learning overview

Recently RL had an increased interest as a result of its general solution to most problems so no need to reestablish a theory for every area in science it can work on all areas. RL finds the solution by giving the computer a target to achieve and then let's the computer learn by trial and error till it learns to achieve this goal. Scientists are eager to try reinforced learning in solving problems that were impossible to solve before.

Let's consider an illustrating example of how to teach a controller to ride a bike using reinforced learning techniques the target we want to teach the contoller isn't to crash the bike. So as we said before it will learn by trial and error and in this case the controller have to possible control actions which are turn left or turn right in the first trial the bike will start moving forward while applying turning left or right randomly so it will continue till it reach the state of being tilted to the right 45 degrees then it will choose randomly to turn left, the bike will crash resulting in something called penalty or negative reward that means that it should avoid moving left when tilted 45 degrees to the right

Now another trial begins the bike continue to turn right and left again while moving forward and it reaches the same state before of being tilted 45 degrees to the right so it already learnt not to turn left at this position so it will turn left. The bike will crash again to the ground thus receiving a negative reward. This time it will learn that being in the state of being tilted to the right 45 degrees is wrong so the system will learn to avoid this state.

So it will continue with the trials until the system after a lot of crashes it will eventually learn how to ride the bike and keep its balance without crashing [45].

2.4.2. Reinforcement Learning Problem classification

In reinforced learning there is a communication between various components there is the agent who communicates with the environment and sends the description of the environment to the reinforcement function where it uses the value function to apply the reinforced learning algorithm. As mentioned in the paragraph above the reinforced learning problem could be classified in to three main categories that are discussed in detail next[46].

A. The Environment

In Reinforced learning a matching occur between the states and the actions so the environment states has to be fully observable this assumption is necessary for the reinforced learning method. The agent as mentioned before receives the states from the environment. An example for the states that is received is the automated drones where its location and speed has to be known also its vertical height and this is achieved with appropriate sensors.

Another example is the batch reactor where the concentration of all the species must be known to be able to perform the reinforced learning, also its conditions such as temperature and sometimes pressure.

B. The Reinforcement Function

The reinforced function is obtained to describe the objective of the reinforced learning problem the reinforced function varies from a problem to another and it's up to the designer to obtain a suitable reinforced function that correctly describe the system being used. the reinforced function receives from the agent a scalar value which could be positive or negative this value is named as the reward, it obtains it through interaction with the environment [20].

The reinforced function can be classified according to the reward type whether it's immediate or late. in immediate rewards problems the rewards have to be maximized instantly so the reinforced function will have to craft the reward as positive at each instant and try to maximize that reward. On the other hand in the late reward problem the reinforced function will put the reward equal to zero on the whole interval except at the terminal state where it have to be equal to a positive scalar reward that is related to the final state and this is the case in the reinforced learning problem of the batch reactor.

Another example is the game of backgammon shown in figure 2-9. The target is to achieve victory this is considered a late reward problem there is 1020 different states that could be achieved by moving the pieces in the board and each piece have a number of admissible controls or actions. the reward function is equal to zero every where on the board only until a defeat or victory occurs it will be positive for a victory and negative for a defeat (penalty). the system will learn how to win by trial and error and using the reinforced function [47].

Figure 2-9 Backgammon game
(Source:[48]**)**

Consider the case of the inverted pendulum cart in figure 2-10 it's described as following there is a cart that have limited space to move in its allowed to move left or right at various velocities and mounted above it is an inverted pendulum the target of this problem is to balance the pendulum over the cart which is not hinged by moving the cart left or right [49].

This is also a late reinforced function problem where the reward is equal to zero unless the pendulum falls it will receive a penalty in the form of a negative scalar and as long as the pendulum is upright it will receive a reward in the form of a positive scalar.

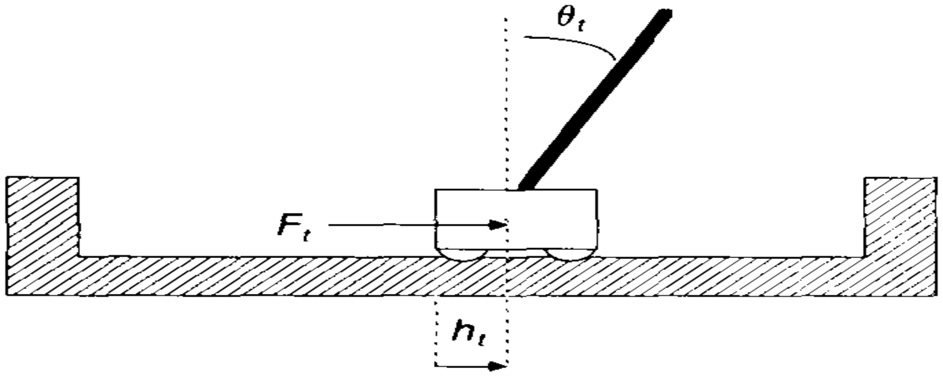


Figure 2-10 Inverted pendulum problem

(Source:[34] **)**

Another problem is named car on a hill shown in figure 2-11 the target of this reinforced learning problem is to make the car reach its target on the hill but the problem is that it have limited momentum so if it used its full thrust forward it won't be able to reach the top of the hill so it has to move backwards on the other side of the hill and gain extra momentum for sliding down due to gravity in this case it will be able to reach its goal. [50].

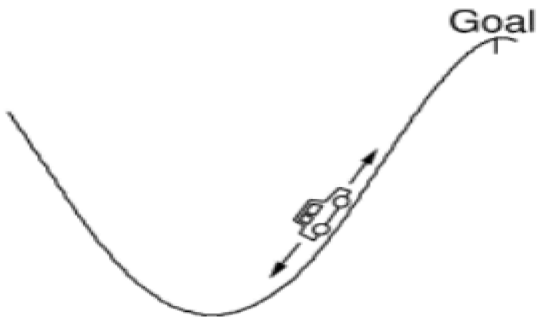


Figure 2-11 Car on a hill problem

(Source:[51] **page:4)**

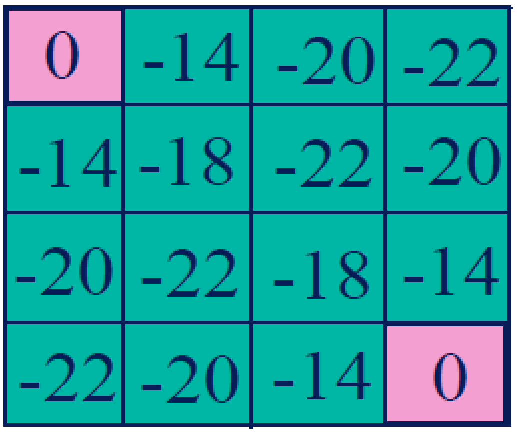
The allowable actions to the car controller are forward, backward, and do nothing. The reinforced function reward will be equal to zero everywhere it will only have one positive value at the goal target. if it reaches the top of the hill it will receive the reward. This problem will require numerous trials to teach the controller how to make the car reach the top.

C. The Value Function

The value function is a function that values the amount of received rewards, it maps the reinforcements received with the states and actions the mapping can be done using various techniques such as neural networks or support vector machines the goal is to achieve the maximum of that value function so when we receive a reward its stored in a variable that is increased or decreased according to the future received rewards the mapping importance is to know the corresponding state and action that made us reach that reward so that we will be able to recall these sequence of states and actions that achieved the maximum of the value function.[52]

Consider a square that's divided into sixteen smaller squares and let's assume we start at any of the small squares and our target is to reach the top left corner or the bottom right corner. there are four possible actions up, down, left and right. As we already know the reinforced learning technique starts its trials with random action the result of the value function for these random action is shown in figure 2-12.

Where approximately there is 22 moves are done before reaching the target square, reward in this case is equal to zero everywhere but for the target squares that are located on the top left corner and the bottom right corner.

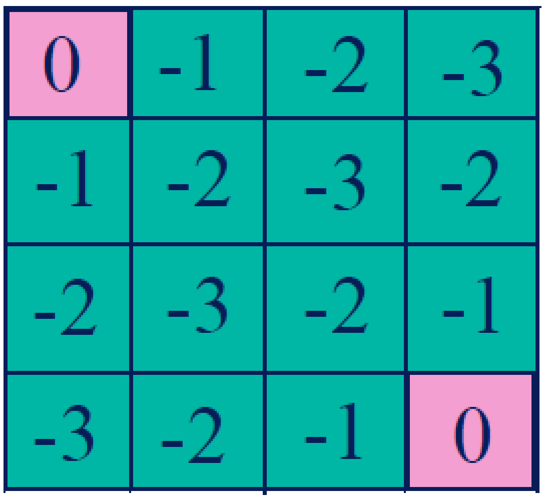


0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

Figure 2-12 Value function for the random policy

(Source:[53] **page:4)**

The optimal police which means the best moves to reach the target for all the initial states (Random small square start) is shown in figure 2-14 and its corresponding value function is shown in figure 2-13. this optimal policy is achieved by maximizing the reward of the value function it's noticed that if the initial starting location was the bottom left it will take only three moves and hence the value function equal to -3.



0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

Figure 2-13 Optimum value function
(Source:[53] **page: 5)**

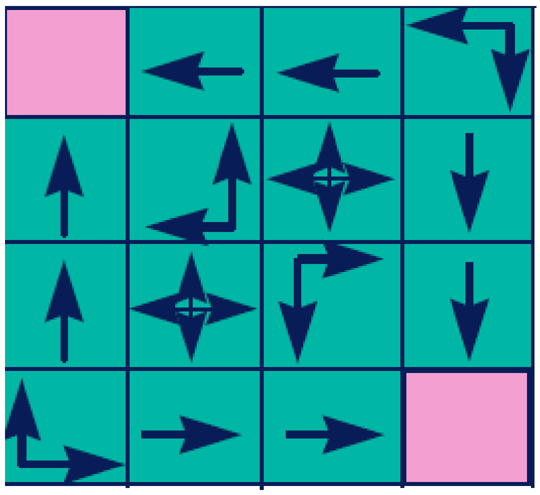


Figure 2-14 Optimal policy
(Source:[53] **page: 5)**

2.5. Q learning

2.5.1. Overview

Q-learning is a method to solve reinforced learning problems it's done by approximating a function called Q-function using values named Q-values. this function contains variables named four tuples which are four variables (x_t,u_t, r_t,x_{t+1}),

where:

x_t: state of the system at time t,

u_t: the control variable or action done.

r_t: The reward received which could be late or immediate.

x_{t+1}:The next system state that resulted from the previous state as the result of an action.

Q-learning works on finding the optimal policy that is at its best is near optimal. Q-learning [54], [55] is an additional extension to old-style dynamic programming.

Q-learning could be classified in to offline and on-line Q-learning. In offline Q-learning the system gathers the four tuples first and then input them into the Q-learning algorithm where the four tuples at first observe the environment or the system and record its states (x_t) and then applies an action (a_t) which result in a preceding state (x_{t+1}) and receives the reward at that instant (r_t) which could be equal to zero or a positive scalar or a negative scalar value. [56]

In online Q-learning there is no set of four tuples obtained but the system tries to reach the optimal policy as soon as possible by only recieving the current state and the next state and the instantaneous reward. In this research we use off line method Q-learning as we care only about the final state so we are not interested in reaching the optimal policy immediately but we are concerned with the optimal policy after a certain time.

Function approximation is necessary for the solution of the Q-learning problem as there is a need to match the Q-values to the states and actions and the optimum policy also is needed to be mapped with the states. An old fashioned way is to use the look up tables which is not used any more a more recent approach is to use the NN or SVM.

Q-learning finds the solution through iteration called "value iteration" where the value function in (2-1) is solved several time to find the best solution where the rewards are added to the maximum of the Q-values obtained from the NN or SVM for the next states and the next actions.

$$\boldsymbol{Q(x_t, u_t) = r(x_t, u_t) + \gamma^{max}_{u_{t+1}}(Q((x_{t+1}, u_{t+1}))} \qquad \textbf{(2-1)}$$

Where:

r:Reward: determines whether the action taken is bad or good

γ:Discount factor : determines how immediate is the reward wether it's effect should be soon or late. its value between 0 & 1.

t:step, **u:**Action, **x:**State, **Q:**Q-value.

2.5.2. Q learning : Example Left or right

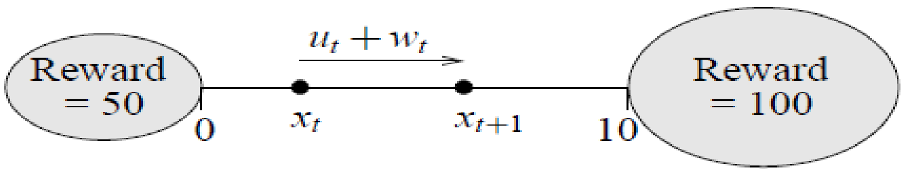


Figure 2-15 Left or right problem

(Source:[57] **page:518)**

- A ball moves in the interval [0,10] as shown in figure 2-15. Two control actions are available. One tends to move the ball to the right (u = 2) while the other to the left (u = −2). As long as the ball is inside the interval, just zero rewards are observed. When the ball leaves the interval, a final state is obtained. If the ball goes out on the right side then a reward of 100 is gained while it is twice less if it goes out on the left. [29]
- Even if going out on the right may lastly results in a better reward, the action is not essentially equal to 2 everywhere as the value of the reward signal gained after t steps is multiplied by a factor $\gamma^{t-1} = 0.75^{t-1}$.
- **Solution: key idea**
  1. We generate fourtuples (xt ,ut , rt ,xt+1), Starting at random initial state and applying a random action (ut) till the final state is reached.
  2. Set initial Q-values equal to zero or to the rewards.
  3. Map the actions and states to the Q-values using a NN or SVM name it QV1.
  4. We find the maximum Q-values for the next states using all possible actions via the previously trained NN or SVM name them QV0.
  5. We apply Q-function algorithm [QV1= r + QV0]
  6. We repeat steps 3 till 5
  7. We simulate the results by applying the test actions on all the states via the NN or SVM.
- The Q-values for the first action which is left is higher from 0 till 2.7 and the second action which is right is higher from 2.7 till 10 which means the method will choose left if x<2.7 and will choose right if x > 2.7 as shown in figure 22 the matlab solution code is in the appendix.

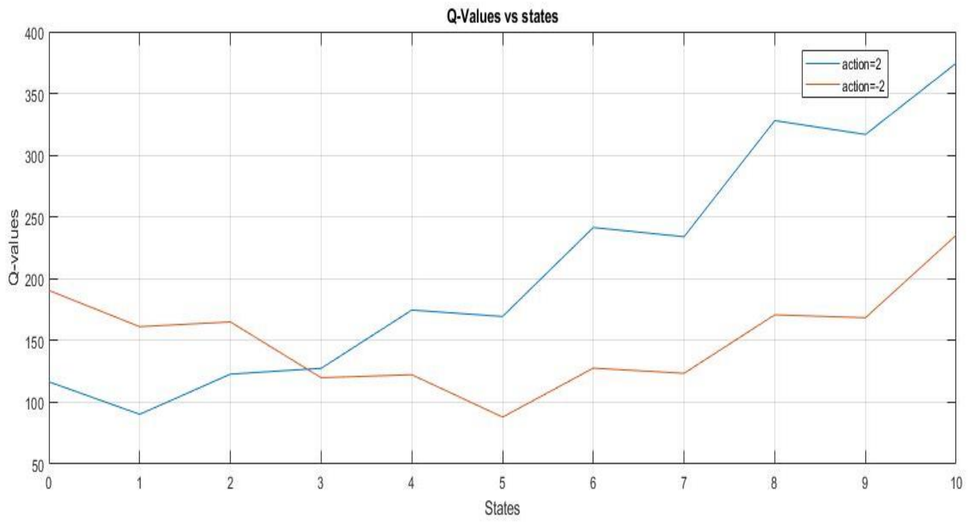


Figure 2-16 Q-values versus the states for the left or right problem

2.6. Markov Decision Process

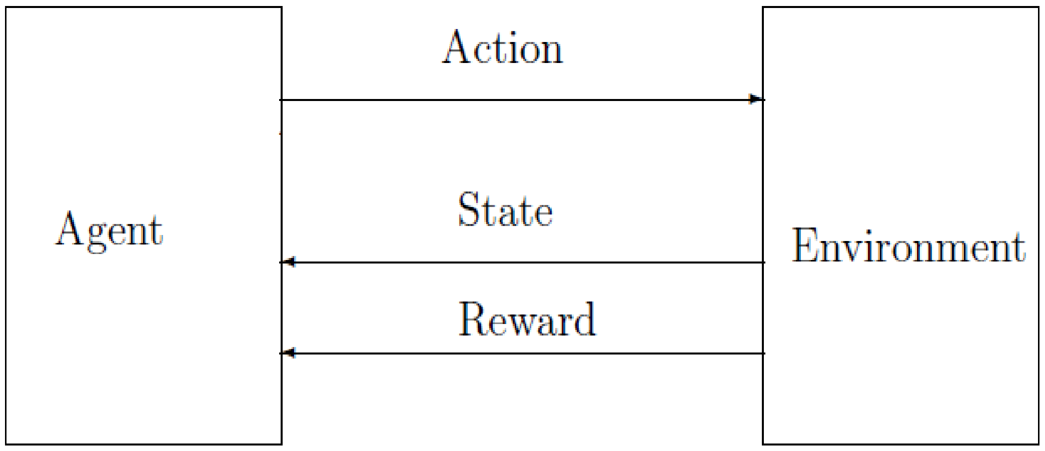


Figure 2-17 Agent interaction with environment

(Source: [58] **chapter:4, page: 1)**

Let's consider an agent that interacts with an environment as shown in figure 2.17 . it have the following characteristics first the states are being sent to the agent from the environment could be deterministic or probabilistic, there is a reward received with each action sent to the environment and this action results in the environment to transform from its initial state to a different preceding state [59].

According to the previous paragraph the agent can make prediction to what the environment will reach if a specific or certain action is applied to it. [60]

2.7. State of the art (Gap & Aim)

2.7.1. Current state of the art

Dynamic optimization of nonlinear chemical systems -- such as batch reactors -- should be applied online, and the suitable control taken should be according to the current state of the system rather than the current time instant. Recent state of the art methods apply the control based on the current time instant only. This is not suitable for most cases, as it is not robust to possible changes in the system. This paper proposes an enhanced robust Q-Learning method to conduct robust online optimization of a batch reactor.

The Chemical industry can be classified into continuous and batch processes. In continuous processes there is a set point for operation (the most desirable point) and the plant is usually continuously operated at this set point. This is called steady state operation.

From the economic point of view continuous operations should not be used for small volume, high value production operations. These kind of operations are better suited for batch processes. Batch processes include batch reactors which are classified into batch and semi batch operations. In batch operations all the reactants are loaded initially in the reactor vessel and adjustment of the temperature profile or the final time can be done to optimize the final product specifications. In semi-batch operations the feed is added by a flow rate which can be adjusted.[3]

Batch processes are known to have a dynamic nature, so it is possible to change the product specifications; and they are also suitable in a facility that produces multiple products. Batch processes are used in the production of speciality chemicals, pharmaceuticals, food and consumer products and biotechnology products [2].

Batch processes are optimized to meet production specifications (environmental, safety and quality) and to increase the profit. In industry usually heuristic-based optimization is done based on the experience and knowledge gained from previous batches. In academia, optimal control is the branch of science that scientifically deals with this kind of problem. One of its methods is dynamic programming. The dynamic programming algorithm was invented in 1957 by Richard Bellman who defined the Bellman principle of optimality[18].

Dynamic programming can be solved in different ways, one of which is iterative dynamic programming. Luus [22] have showed that by using iterative dynamic programming for a batch process global optimum can be found for several nonlinear systems; but it has several limitations. The most important limitation is that the optimal actions is a function of time only and is valid only for a fixed initial point; therefore, the optimal actions is not a robust action. Our aim in this paper is to devise an enhanced q-learning algorithm that overcomes this limitation by finding the policy which is the actions mapped with the states.

Since the development of detailed mechanistic reliable models is generally very time consuming and effort demanding for batch processes, there is an approach for developing improved data-driven control and optimization approaches [61]–[63] . These approaches require the availability of online concentration-specific measurements such as chromatographic and spectroscopic sensors, which are not yet readily available in production. Technically, the main operational difficulty in batch-process improvement lies in the presence of run-end outputs such as final quality, which cannot be measured during the run. The drawback of these methods is thatthey require a significant amount of data that should cover a wide range of the system state space. In addition, these methods map the relation between the optimal actions with time, which means that if anything went unplanned or wrong during the process this action would not make the product reach the desired specifications (i.e. not a robust actions)

Machine learning strategies have been used to provide control methods for chemical reactors and processes [64]–[66]. Those strategies include methods such as reinforced learning (RL),which includes Action-dependent HDP (ADHDP) also known as Q-learning,and the closely related adaptive critic designs (ACD) [67].

Balakrishnan and Viega first introduced the adaptive critic designs that is typically consists of two neural networks. firstly, the critic neural networks which maps the relation between the states and the co-states (Lagrange multipliers) and secondly the action neural network that maps the actions to the states. This kind of ACD was named DHDP (Dual heuristic dynamic programming) [33],

Adaptive critic designs (ACD) have several advantages. They can solve nonlinear problems, provide solutions for various initial conditions, require less computational load (compared to traditional dynamic programming methods) and can be applied in real time control problems. In general, Adaptive critic designs require a model.

The Q-learning approach optimize the actions of an agent based on responses from its environment, where it aims to maximize or minimize a scalar reward. A distinct difference between Q-learning and the other traditional ACD approaches is that there is no prescribed behavior or training model in the Q-learning approach. Recently Q-learning techniques have been introduced to solve control problems [68]The rewards, states, actions are basic concepts in the reinforced learning literature[69].

In recent years, many research efforts have been done to develop RL techniques to solve control problems without the need for a model. RL-based algorithms were developed to find the solution to the optimal tracking problem for both nonlinear discrete-time systems and nonlinear continuous-time systems.

Q-learning have successfully designed controller which can solve the model-free optimal tracking control problem of general non affine nonlinear systems. Q-learning is a model-free control design that uses off-policy data. On-policy and off-policy learning schemes are two main RL frameworks for control design. The off-policy learning– used in this paper – has several advantages and is regarded as more practical and efficient[70].

2.7.2. Gap in current state of the art

In all the known dynamic optimization methods the optimization method doesn't respond to sudden changes (no intervention) such as the dynamic programming or the IPO (Interior point optimization) and also need to be solved every time the initial value changes. Only the Q-learning respond to changes and can solve for any initial value but doesn't have consistent results and also haven't been applied on batch reactor optimization before.

The current state of the art deals with discrete state, discrete action markov decision processes, an advantage of our method that it is considered continuous state, discrete action deterministic markov decision process. with the addition of a mapping step between the states and the optimal actions, which is considered a new step for markov decision process.

In this Thesis, we propose an enhancement for the Q-Learning algorithm. The main enhancement is the search for the optimal next action, inside the process of searching the current best action. In other words, we simultaneously search for the optimal two successive actions (i.e. u_t& u_{t+1}) using exhaustive search while simultaneously searching for best values for . u_t& u_{t+1}.

It's shown in the equation below:

$$Q(x_t, u_t) = \mathrm{r}(\mathrm{x_t}, \mathrm{u_t}) + \max_{\mathrm{u_{t+1}}}(\mathrm{Q}(\mathrm{x_{t+1}}, \mathrm{u_{t+1}}) \qquad \textbf{(2-2)}$$

where:

Q is the value function

r is the rewards function

γ is the discount factor valued between [0,1]

x is the state of the system

u is the control action, t is the time instant

In the proposed algorithm, approximation methods such as neural networks or support vector machines map rewards to states and optimal actions, and also map optimal actions to states. Thus, the optimal actions is a robust action as it depends on the current state rather than the current time. Hence the optimal actioncan be applied online and can accommodate unplanned changes. To test the power and flexibility of the proposed method, we present simple multi-reaction batch reactors example solved with the original Q-learning algorithm and the proposed enhanced Q-learning algorithm.

We try to maximize the final product obtained or meet certain specifications of the products (i.e. Minimize side products).This could be done using several ways, the advantage of our proposed method that it can overcome sudden changes during the reaction with better consistency and robustness.

3. METHODOLOGY

3.1. Proposed enhanced q-learning algorithm

Our proposed approach generates data to be dealt as an input for the algorithm, The data obtained is from the process or simulation of the process using a model described by differential equations and after the data been generated -there is no need for the model. The inputs and output are listed in Tables 3-1 & 3-2 respectively.

Table 3-1 Inputs to the algorithm

Short name	Full name	Type	Dimensions
NumS	Number of states	Scalar	-
NumA	Number of actions	Scalar	-
NumN	Number of total iterations	Scalar	-
TA	Test actions	Vector	-
OA	Optimal actions	Vector	-
Q_0	Initial Q value	Vector	-
SMx	States matrix	2D matrix	(States,NumS)
NS3D	Next state 3D matrix	3D matrix	(States+1,NumS,NumA)

Table 3-2 Output from the algorithm

Short name	Full name	Type	Dimensions
Mdl	Model	Support vector machine mapping function or a Neural network	-

Where the algorithm uses the formula mentioned earlier that enhances the results. Its output is a mapping function that will be used to predict the optimum policy.

3.3. Data preparation for the learning algorithm

1. Consider the following differential equations model describing a batch rector:

$$\frac{dX}{dt} = f(x,u) \qquad \textbf{(3-1)}$$

2. Add an additional equation describing the time step which is equal to a constant with initial value equal to zero. The rest initial states x(0) is random.
3. Divide the time intervals in to equal stages.
4. Generate states by applying random actions u for each stage of the differential equations, and epoch the procedure, where u is a scalar value bounded between $u_{min} \leq u \leq u_{max.}$ Avoid very small data sets to avoid bad learning and this is done by increasing the number of epochs while storing the generated data in a matrix.

5. Predict the next state from the differential equationsoneach combination of states using some of the possible actions or every possible actions, and store it in a column in a 3D matrix and add an extra row above the next states that stores the rewards R which are equal to zero except at the final time step it's equal to the performance index as shown in figure 3-1.

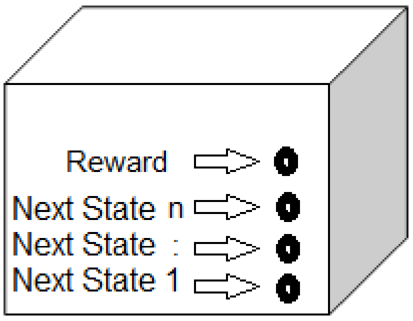


Figure 3-1 The predicted next state 3D matrix showing the location of the Next states and the reward

3.4. Robust Q learning algorithm

- We input to the algorithm the data generated and obtain from it a mapping function between the system states and actions.
- The algorithm consists of four loops one is the extreme outer loop iterates on the number of trials (N=0,....,NumN), the second is the outer loop iterates on the states till it covers all the states (S=1,....,NumS), the third is the middle loop iterates on the test actions till it cover all the test actions (A=1,....,NumA), the fourth is implemented as a matrix operation to simplify and speed up the computations, it iterates the next states on all the possible test actions.
- At the beginning set the optimum actions OA equal to zero, and the initial Q-valuesequal to - infinity.
- The algorithm is shown in figure 3 as a flowchart diagram.
- **Procedure:**
  1. The inner loop performs the following step, We apply the modified Q-function equation:

$$Q(x_t, u_t) = \mathrm{r}(\mathrm{x_t}, \mathrm{u_t}) + \max_{\mathrm{u_{t+1}}}(\mathrm{Q}(\mathrm{x_{t+1}}, \mathrm{u_{t+1}})$$

Where an approximation function could be used such as a support vector machine (SVM) or a neural network (NN). To predict the Q-value from the next states and test actions, after the extreme outer loop finishes one trial. Where the mapping between them is done in step 4 Where we apply each Next states in the NS3D matrix on all possible test actions TA. It could be done in two possible ways:

A. Matrix Operation:

Where the current next states is repeated by the size of the TA and then entered as an input matrix to the chosen approximation function. and the maximum of the output is taken and added to the current reward.

B. Loop iteration:

Where the current next states is entered to the approximation function along with the first test action TA and store the output in a matrix, then repetition is done until all the test actions are covered. The maximum of the output is then added to the current reward.

2. The middle loop performs the following steps:

   A. Obtain from the next state 3D matrix a vector of a next states NS and the scalar of the reward R make the Q-value equal to R, where the vector and the reward is from the values that correspond to S & A from the NS3D matrix.

   B. If the total iteration N>0 we apply the inner loop.

   C. If the result ofthe previous equation is higher than the initial Q-values then the optimal Q-value will equal to the Q-value and OA=TA.

3. When the middle loop finishes (A=NumA),repeat the middle loop until the outer loop is completed (S=NumS).

4. Map between the states and the optimum action vector with the rewards vector. using the previously used approximation function.

5. Repeat the previous steps until (N=NumN).

6. Map all the states with the desired optimum policy values using a neural network or support vector machine (Mdl). Test the approximation function using various initial conditions.

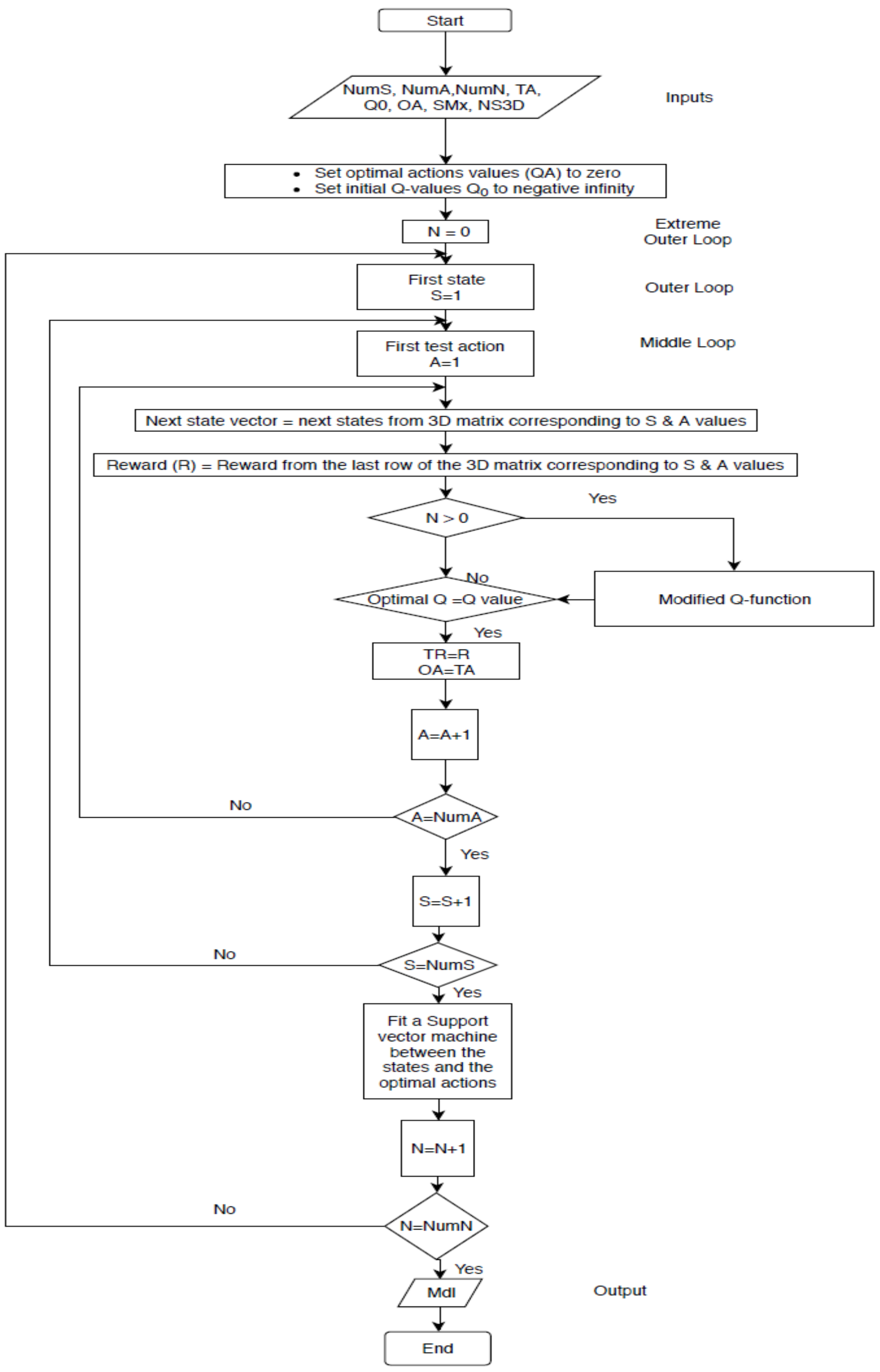

Start
NumS, NumA,NumN, TA, Q0, OA, SMx, NS3D
Inputs
Set optimal actions values (QA) to zero
Set initial Q-values Q_0 to negative infinity
N = 0
Extreme Outer Loop
First state S=1
Outer Loop
First test action A=1
Middle Loop
Next state vector = next states from 3D matrix corresponding to S & A values
Reward (R) = Reward from the last row of the 3D matrix corresponding to S & A values
N > 0
Yes
No
Modified Q-function
Optimal Q =Q value
Yes
TR=R
OA=TA
A=A+1
No
A=NumA
Yes
S=S+1
No
S=NumS
Yes
Fit a Support vector machine between the states and the optimal actions
N=N+1
No
N=NumN
Yes
Mdl
Output
End

Figure 3-2 Flow diagram of the proposed Modified Q-learning algorithm

4. RESULTS & DISCUSSIONS

4.1. Simple batch reactor

Consider the batch reactor studied by Logsdon and Biegler [71] but with random initial conditions. The reaction scheme is $A \rightarrow P \rightarrow C$ and the problem is to find the best temperature policy to maximize the yield the component B after 1 hour of the reaction. The equations describing the reactor are found in 1-17 and 1-18:

Where:

[A] is A mole fraction,

[P] is P mole fraction,

t is the time,

T is the temperature in (K).

With initial conditions $[A]_0=1,[P]_0=0$

With constraints on the temperature ($298< T < 398$)

The performance index to be maximized is the final concentration of P. $I([P]_f)$

Where the final time =1.0 hr

we apply the algorithm discussed earlier. The results are considered very good for a near optimal solution method the final concentration of component B was 0.6091.

The data are generated from solving the differential equations 150 times and dividing the whole time interval in to ten equal stages and the states generated (i.e. mole fractions of A & B& time step) where stored in the matrix SMx its size was (400,3) the test actions values where chosen between 298 and 398 equally. We used the TA vector and the SMx matrix along with equations (2) & (3) to generate the NS3D matrix And using support vector machines with kernel function as polynomial of the second order. Table 4-1 shows the parameters used

Table 4-1 Parameters used in solving example 1

Short name	Value
NumS	400
NumA	11
NumN	40
TA	298,...,398

The original Q learning method gives the policy shown in figure 4-1, with a final performance index varying with each trial where it's bounded between 0.58 and 0.605 for all the trials. A neural network was used with one hidden layer the results vary greatly according to the number of the neural network neurons so they have to be chosen carefully.

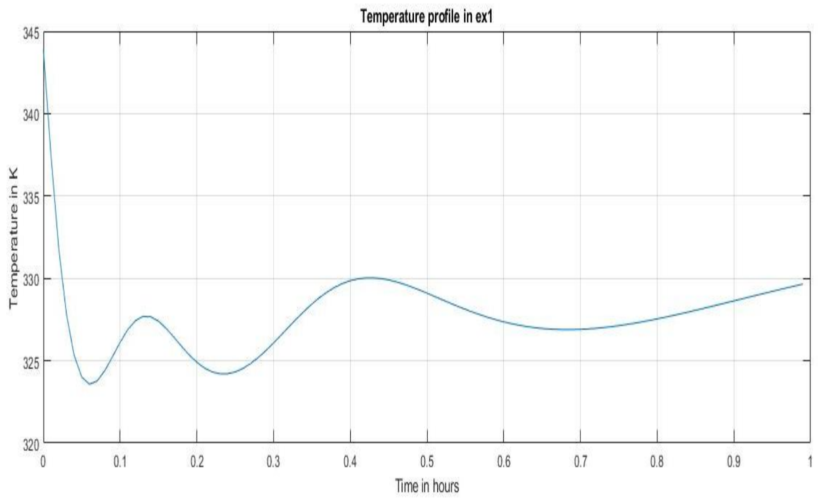


Figure 4-1 Temperature versus time for the original Q learning method for first case study

Comparing these results with another solution method that uses Matlab built in function fmincon using the algorithm IPO (Interior point optimization) we get Final mole fraction of B = 0.61014 and the solution using iterative dynamic programming by Luus [6] obtained the final concentration equal to 0.6109.A comparison of the trends of the temperature with time is given in figure 4-2.In figure 4-3 the performance index with the number of trials is shown for all methods.

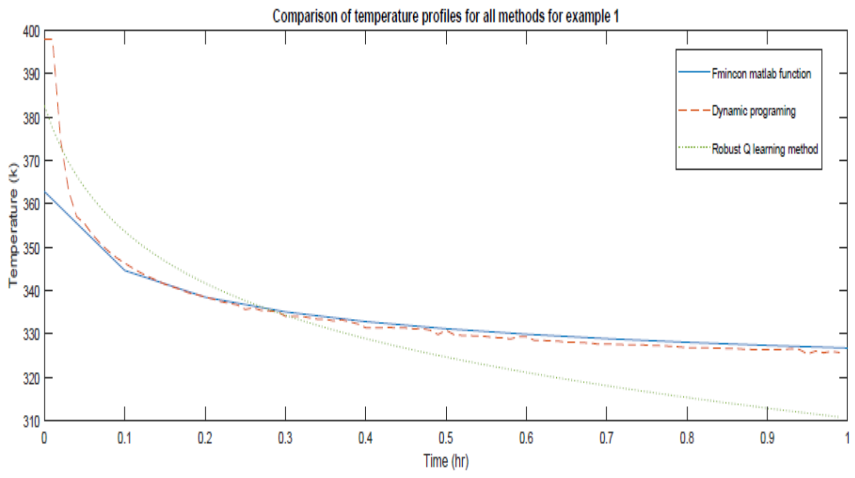


Figure 4-2 Temperature versus time for comparison for all the methods for first case study

we now will assume some scenarios to proof the robustness of the method lets assume that the batch reactor temperature is controlled by a cooling jacket and a heating coil the ideal scenario is what the previous results showed us. what if the heating coil blocked and the temperature fell down or the cooling jacket failed to cool down the reactor for any reason and the temperature rose. with our proposed method the intelligent intervention will assure the optimum path after the failure has been solved. which will be different than the ideal situation optimum path.

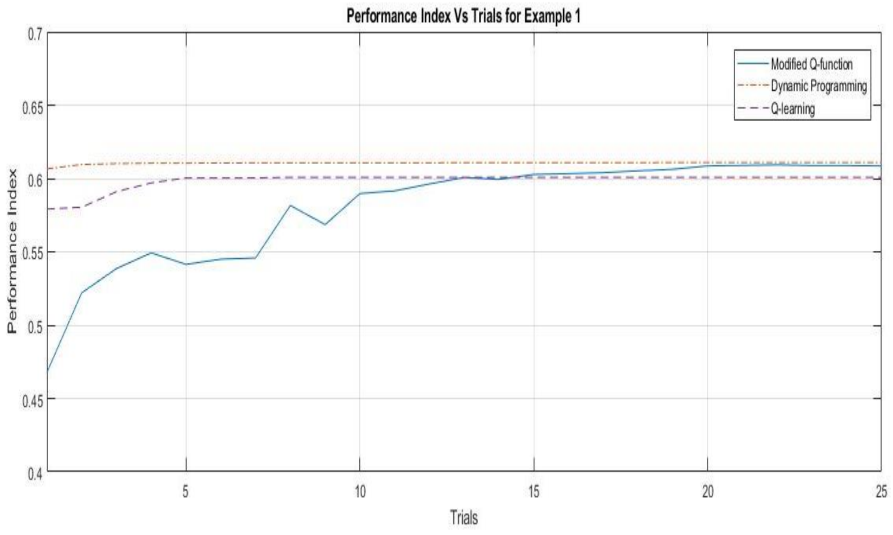


Figure 4-3 Performance index versus trials comparison for all the methods for first case study

In figure 4-4 the temperature is assumed to reach the minimum between 0.2 and 0.6 of the total time, and then the optimum path after will be lower it will achieve a final mole fraction of 0.59 versus 0.58 without intervention. In figure 4-5 the concentration profile is shown for both methods. In figure 4-6 the temperature is assumed to reach the maximum between 0.2 and 0.3 of the total time, and then the optimum path after will be lower it will achieve a final mole fraction of 0.58 versus 0.57 without intervention. In figure 4-7 the concentration profile is shown for both methods.

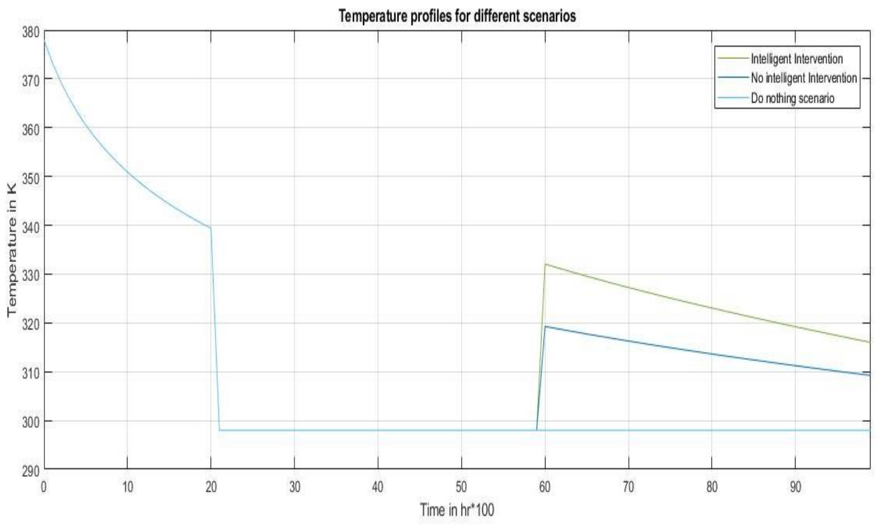


Figure 4-4 Temperature control profile for heating failure for first case study

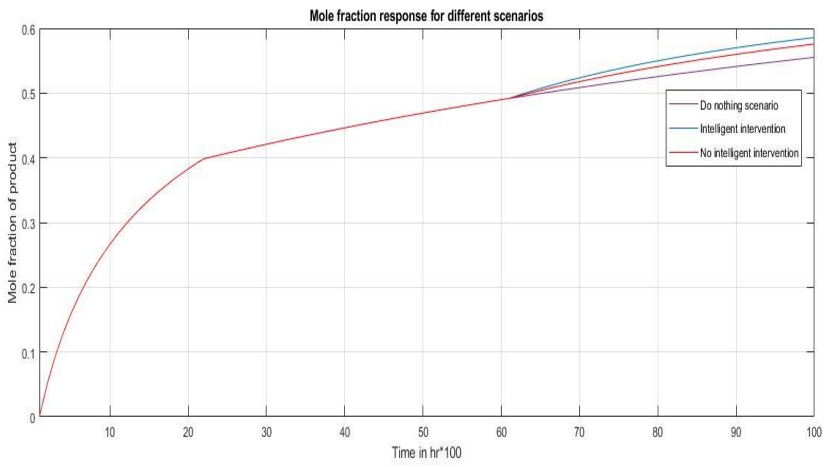


Figure 4-5 Concentration profile for heating failure for first case study

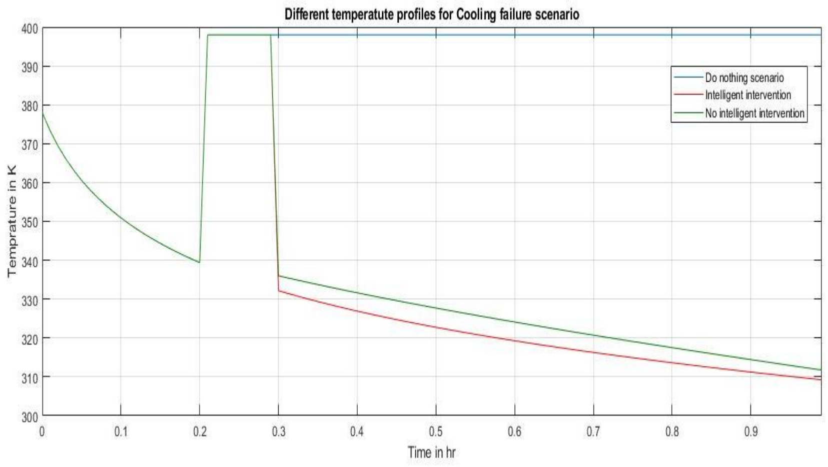


Figure 4-6 Temperature control profile for cooling failure for first case study

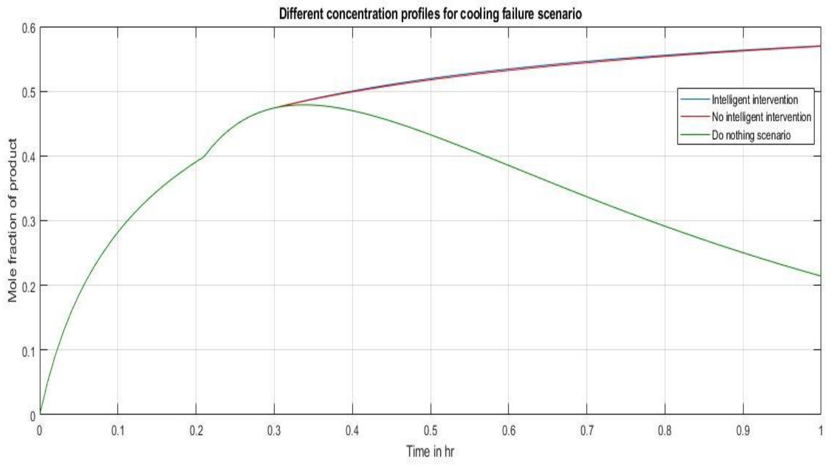


Figure 4-7 Temperature Concentration profile for cooling failure for first case study

As for the minimum time to reach the maximum mole fraction, the time was 5.75 hours and the temperature and the mole fraction profile is shown in figure 4-8 & figure 4-9.

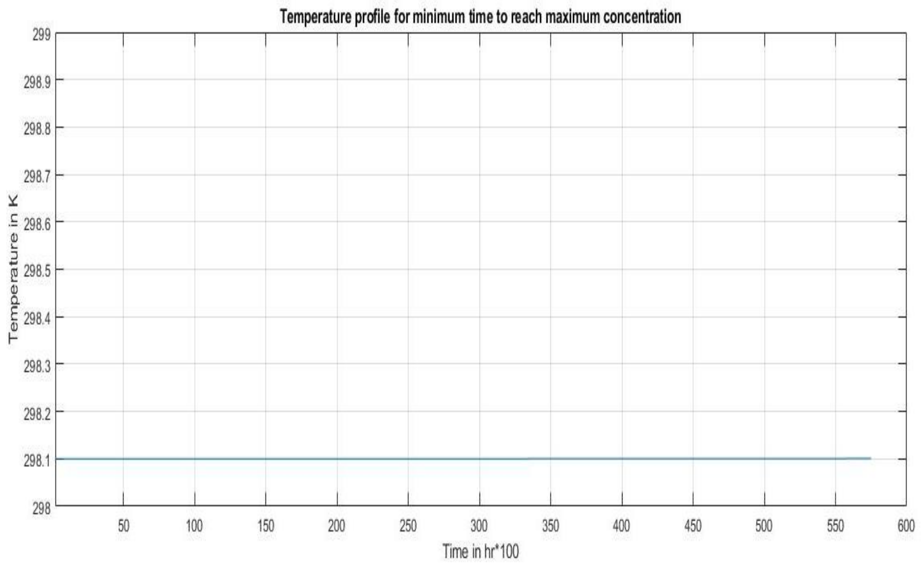


Figure 4-8 Temperature profile for maximum mole fraction in minimum time for first case study

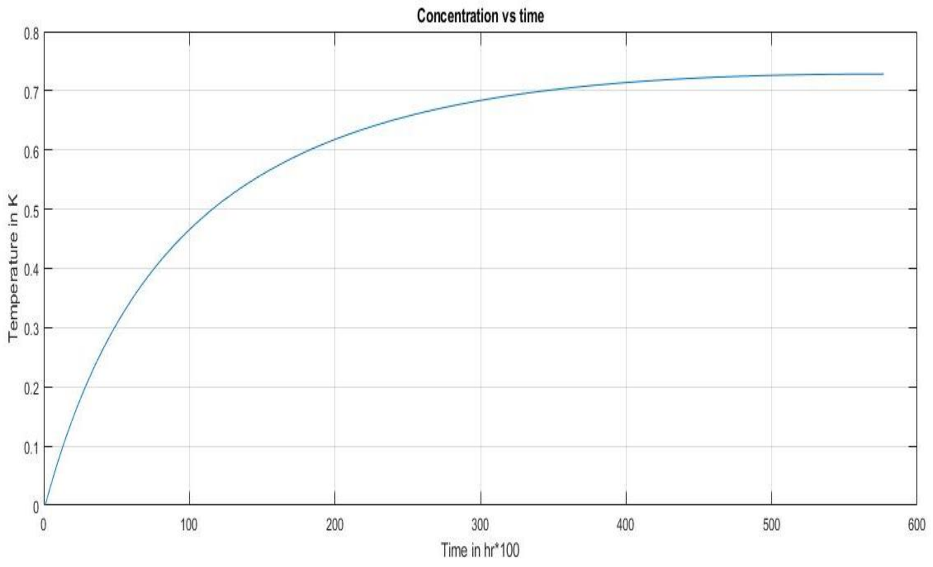


Figure 4-9 Mole fraction profile for maximum mole fraction in minimum time for first case study

The optimum temperature for the final product at various end times is shown in figure 4-10, and its corresponding mole fraction is shown in figure 4-11.

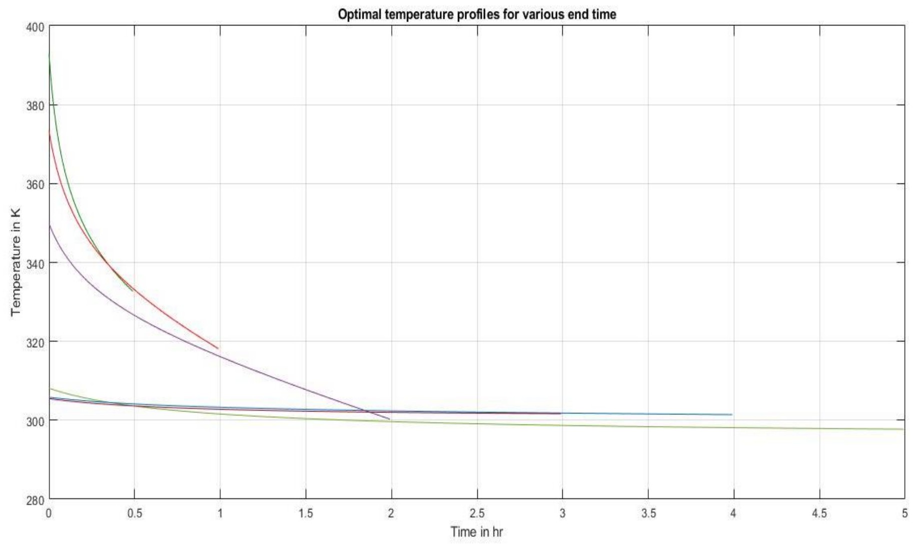


Figure 4-10 Optimum temperature profiles for different end times for first case study

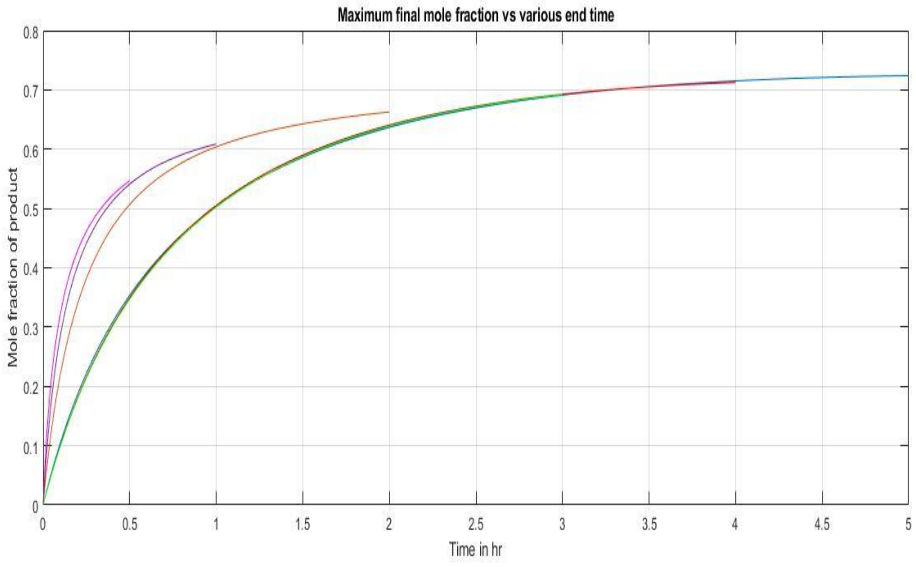


Figure 4-11 Optimum mole fraction profiles for various end times for first case study

The rate of formation of the reactant species is shown in figure 4-12 and as it shown the rate of formation absolute value decreases as the reaction progress.

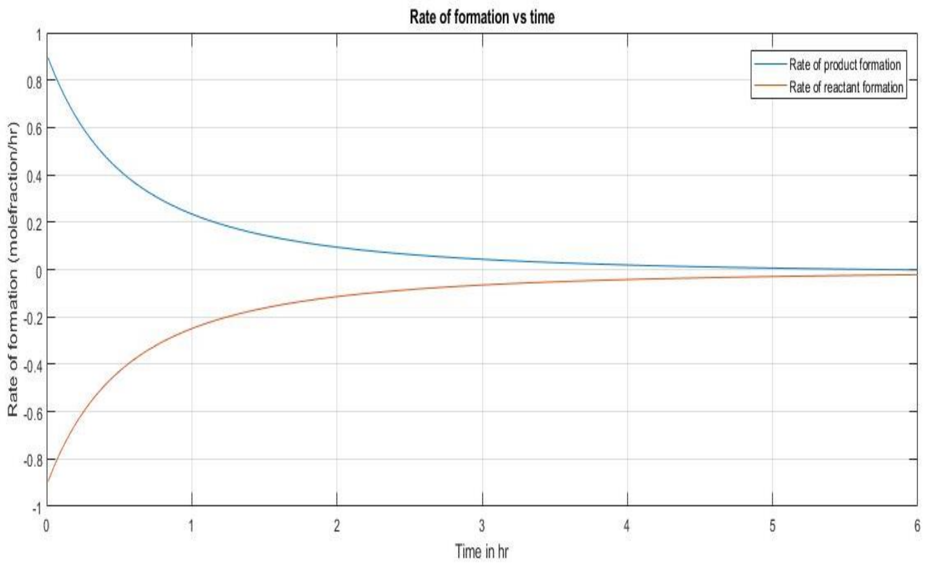


Figure 4-12 Rate of formation versus time for first case study

Table 4-2 Results for various methods for example 1 in the case of nothing goes wrong

Method	Maximum final mole fraction of product in 1 hr.
IPO (Interior point optimization)	0.61014
Dynamic programming	0.6109
Q-Learning	0.585-0.61
Robust Q-learning	0.609

Table 4-3 Final optimum mole fraction at various end time for example 1 using Robust Q-learning method

Final time	**0.5 hr**	**1 hr**	**2 hr**	**3 hr**	**4 hr**	**5 hr**
Final optimum mole fraction (Robust Q-learning)	0.546	0.609	0.642	0.69	0.714	0.725

Table 4-4 Minimum time in hr to reach maximum mole fraction for example 1

Minimum time in hr to reach maximum mole fraction	**5.77 hr**
Maximum mole fraction reached	0.7284

The Matlab codes for solving the previous case study is presented in the appendix.

4.2. A Fed batch reactor:

Consider a chemical reaction system:

A+B → C

B+B → D

Conducted in an isothermal semi-batch reactor. While simple, this example characterizes an important class of industrial problems. Since it is also a common benchmark for optimal control approaches, we will assume the same numerical values as given by Terwich [72], and the problem is to find the best flow rate policy to maximize the yield the component C after 120mins of the reaction. The equations describing the reactor are:

$$\frac{d[A]}{dt} = -k1 * [A][B] - \frac{[A]}{V}u \quad \textbf{(4-1)}$$

$$\frac{d[B]}{dt} = -k1 * [A][B] - 2 * k2 * [B][B] + (b_{feed} - \frac{[B]}{V}) * u \quad \text{4-2)}$$

$$\frac{d[C]}{dt} = k1 * [A][B] - \frac{[C]}{V}u \quad \textbf{(4-3)}$$

$$\frac{d[D]}{dt} = 2 * k1 * [A][B] - \frac{[D]}{V} \quad \textbf{(4-4)}$$

$$\frac{dV}{dt} = u \quad \text{4-5)}$$

Where:

[A] is A concentration in Kmoles per m^3,

[B] is B concentration in Kmoles per m^3,

[C] is C concentration in Kmoles per m^3,

[D] is D concentration in Kmoles per m^3,

V is reactor volume in liters,

t is the time,

b_{feed} is the feed flow rate concentration in Kmoles per m^3,

k1 is the reaction rate constant for the first reaction (Kmol/min.m^3).

k2 is the reaction rate constant for the second reaction(Kmol/min.m^3),

u: is the feed rate of B to the reactor in liters per seconds

With initial conditions:

[A] = 0.2

[B] = 0

[C] = 0

[D] = 0

V = 0.5

With constraints on:

$[B]_f < 0.035$ moles per liter

$[D]_f < 0.018$ moles per liter

V< 1 liter

u is bounded between 0 & 0.1

The performance index to be maximized is the final concentration of C. I(**[B]**$_f$)

where the final time =120 mins.

We apply the algorithm discussed earlier. The results are considered very good for a near optimal solution method the final concentration of component C was 0.617.In figure 4-14 the performance index with the number of trials is shown.

The data are generated from solving the differential equations 75 times using u either 0 or 0.01 and dividing the whole time interval in to ten equal stages and the states generated (i.e. mole fractions of A & B & C &D & V & time step) where stored in the matrix SMx its size was (750,6) the test actions values where chosen between 0 and 0.1. We used the TA vector and the SMx matrix along with equations (4-3) till (4-7) to generate the NS3D matrix And using support vector machines with kernel function as polynomial of the second order. Table 4-5 shows the parameters used.

The solution was better to the one provided by Ruppen [73] where the final performance index was 0.0622 the solution using matlab built in function fmincon was 0.0635 the feed rate vs time is shown in figure 4-13 for All the modified Q-learning, dynamic programming, Q-learning and the IPO solution.

Table 4-5 Parameters used in solving example 2

Short name	**Value**
NumS	750
NumA	11
NumN	10
TA	0,…,0.1

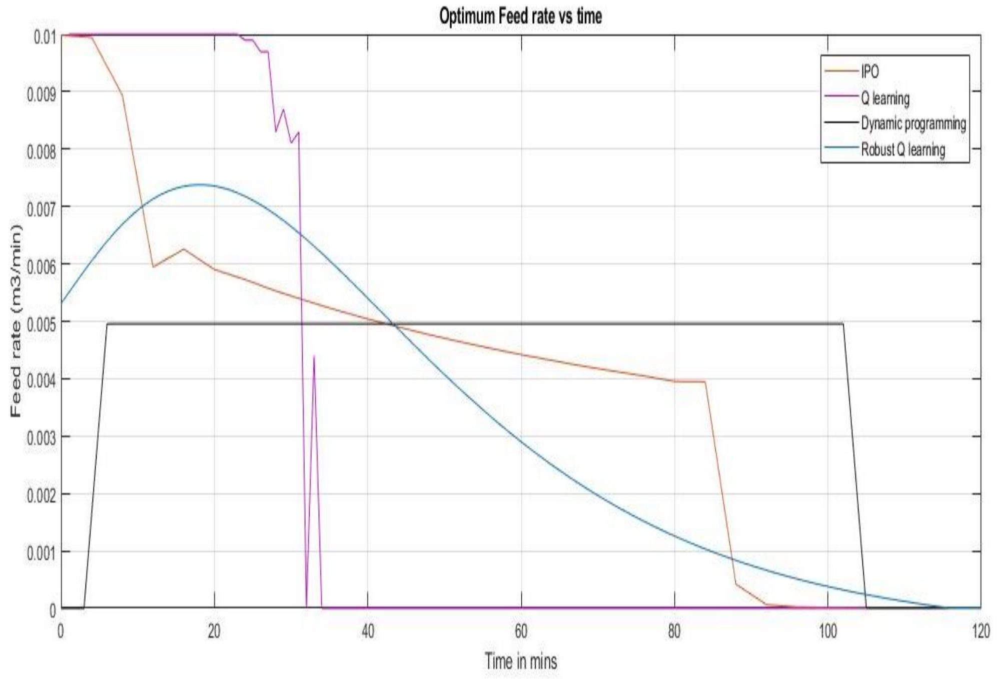


Figure 4-13 Feed rate vs time comparison for all the methods for example 2 in the case of nothing goes wrong

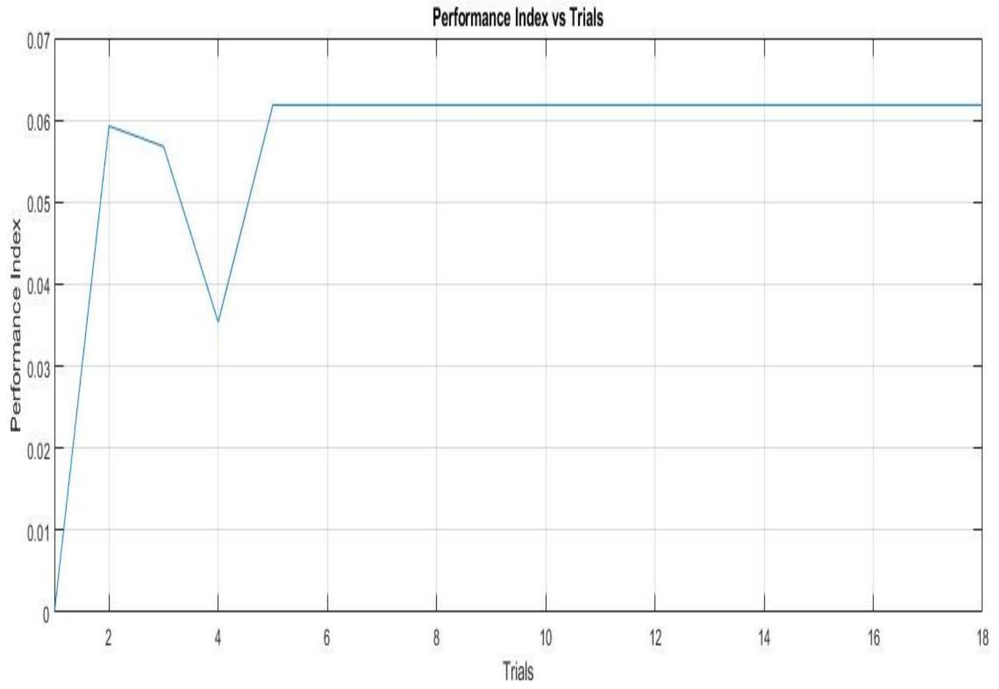


Figure 4-14 Performance index versus trials for example 2

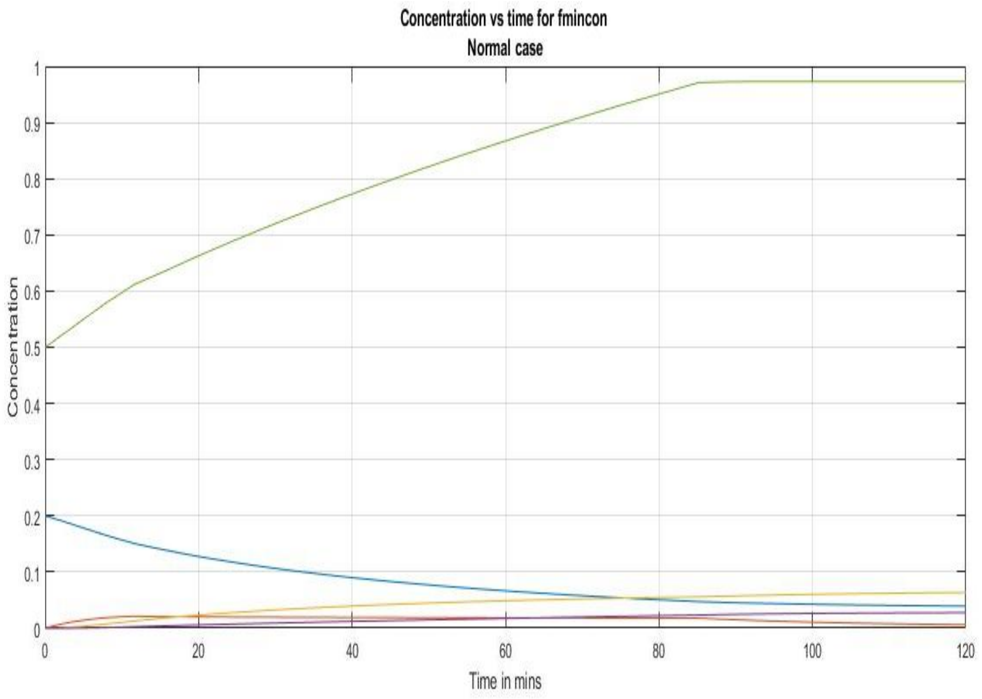


Figure 4-15 Concentration profiles for all components for example 2 in case of IPO (FMINCON)

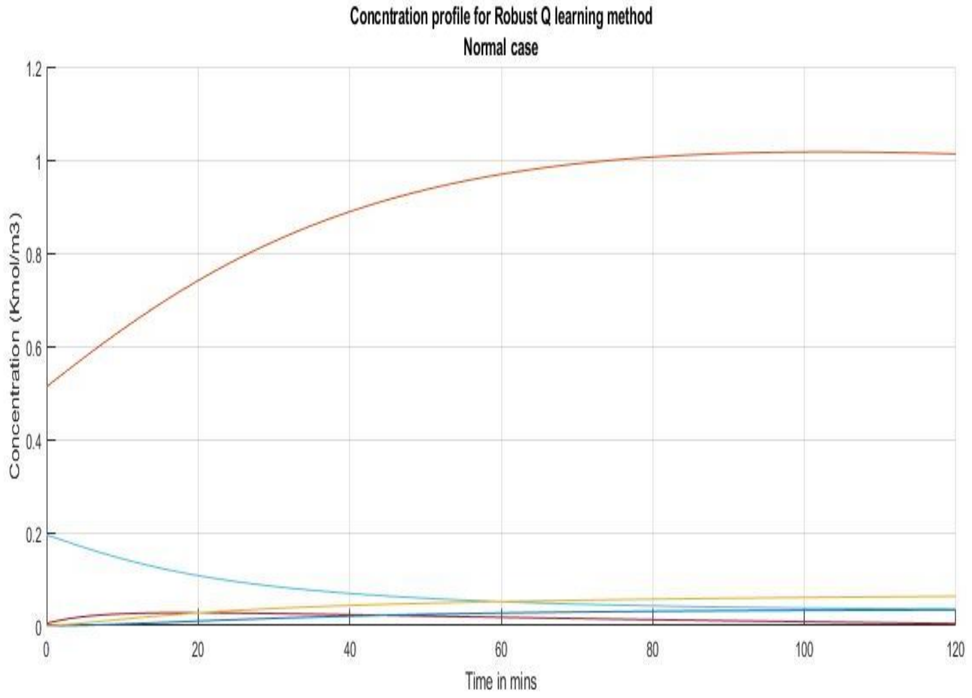


Figure 4-16 Concentration profile for Robust Q-learning method for all components for example 2

we now will assume a scenarios to proof the robustness of the method let's assume that the batch reactor feed flow rate is controlled by a pump, the ideal scenario is what the previous results showed us. what if the pump blocked or failed to pump and the flow rate fell down to zero. with our proposed method the intelligent intervention will assure the optimum path after the failure has been solved. which will be different than the ideal situation optimum path. The feed flow is assumed to stop from the 20th till the 60th minutes, the feed profile is shown in figure 4-17 and the concentration response is in figure 4-18.

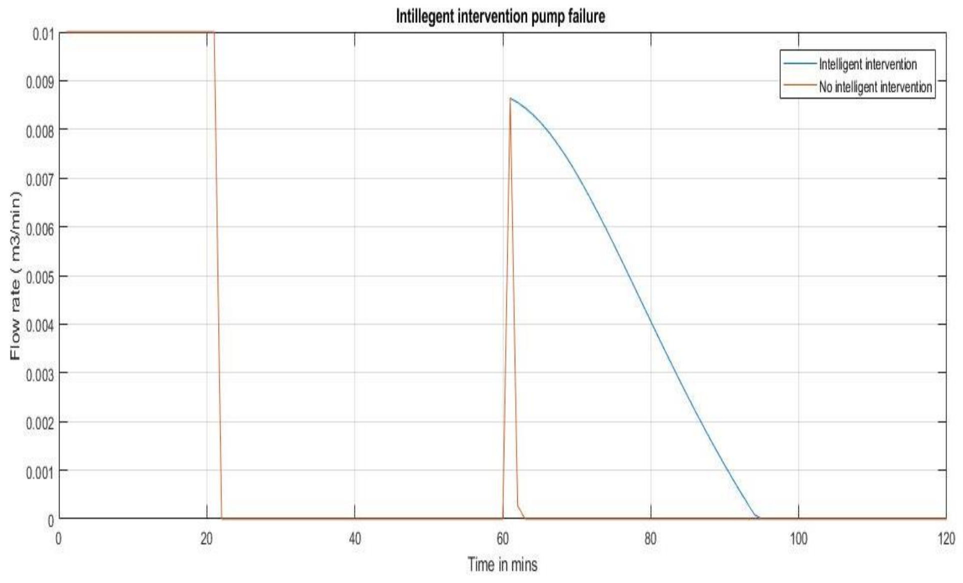


Figure 4-17 Feed flow rate failure scenario for example 2

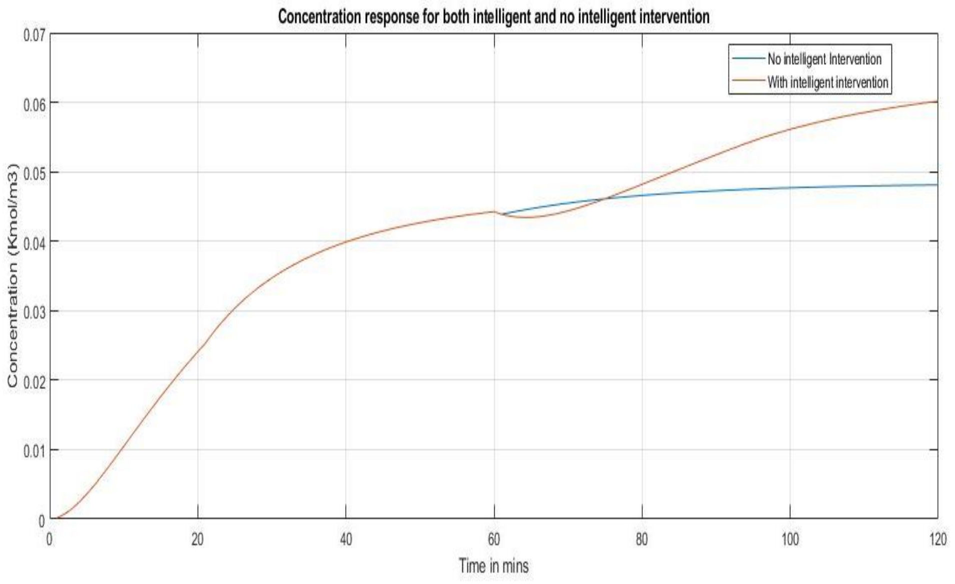


Figure 4-18 Concentration of product response for pump failure scenario for example 2

As for the minimum time to reach the maximum concentration, it was shown in figure 4-19 & figure 4-20 to show the feed flow rate profile and concentration profile respectively. the minimum time was 675 minutes.

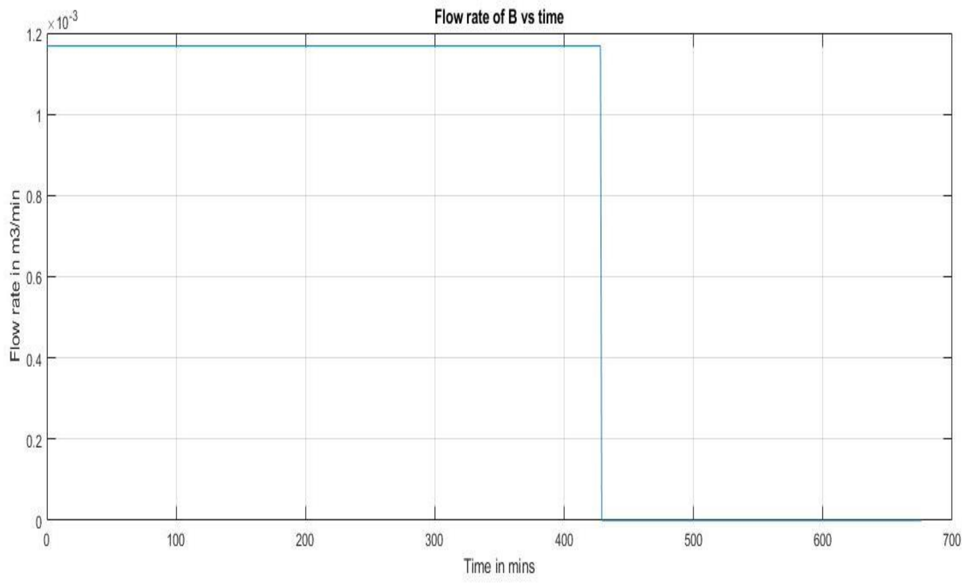


Figure 4-19 Flow rate profile to reach maximum concentration in minimum time for example 2

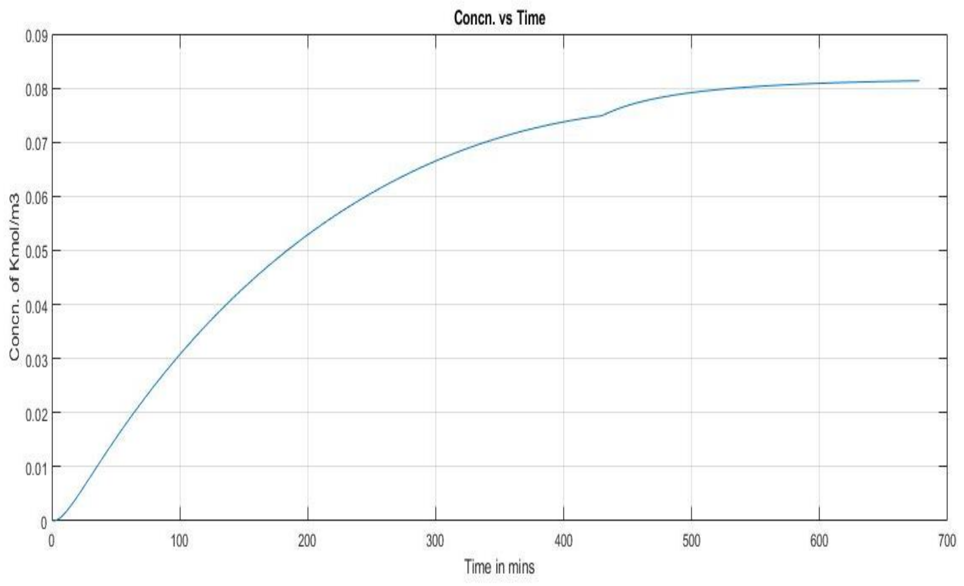


Figure 4-20 Concentration profile of product to reach maximum concentration in minimum time for example 2

The rate of formation of the reactant species is shown in figure 4-21 and as it shown the rate of formation absolute value decreases as the reaction progress.

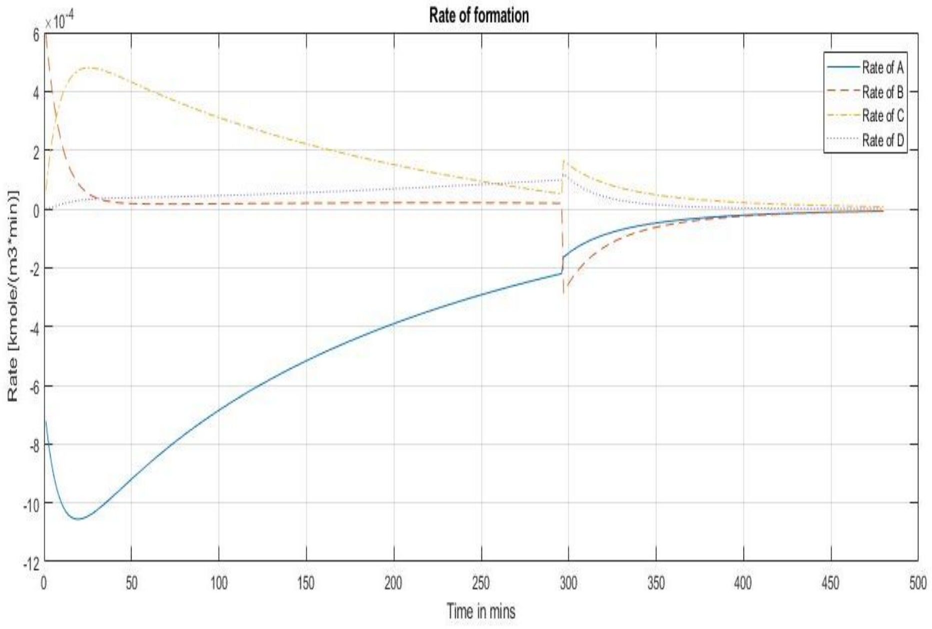


Figure 4-21 Rate of formation for all components for example 2

The optimum flow rate for various end time is shown in figure 4-22, and in figure 4-23 the concentration response for product for various end time.

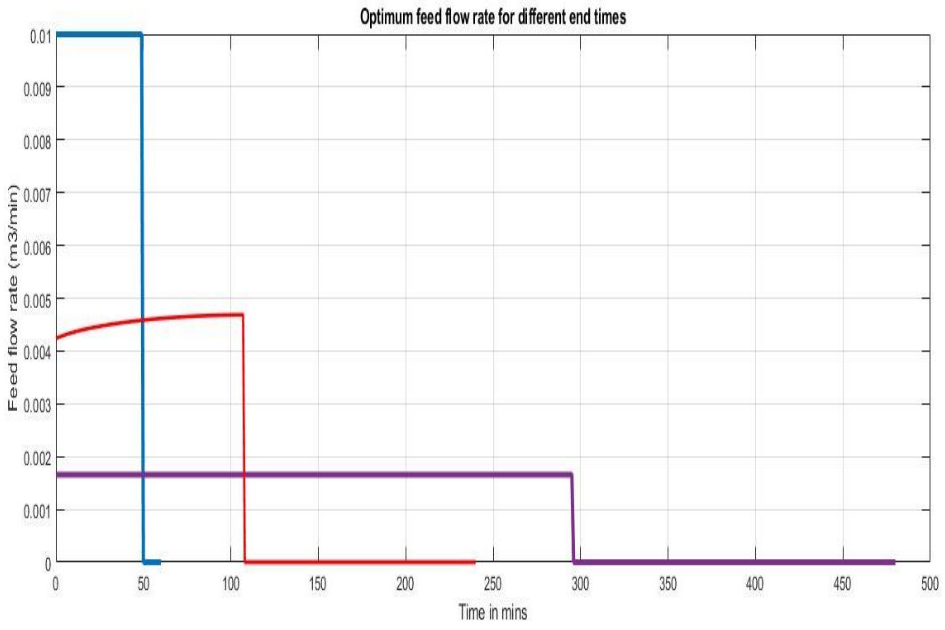


Figure 4-22 Optimum feed flow rate for different end times for example 2

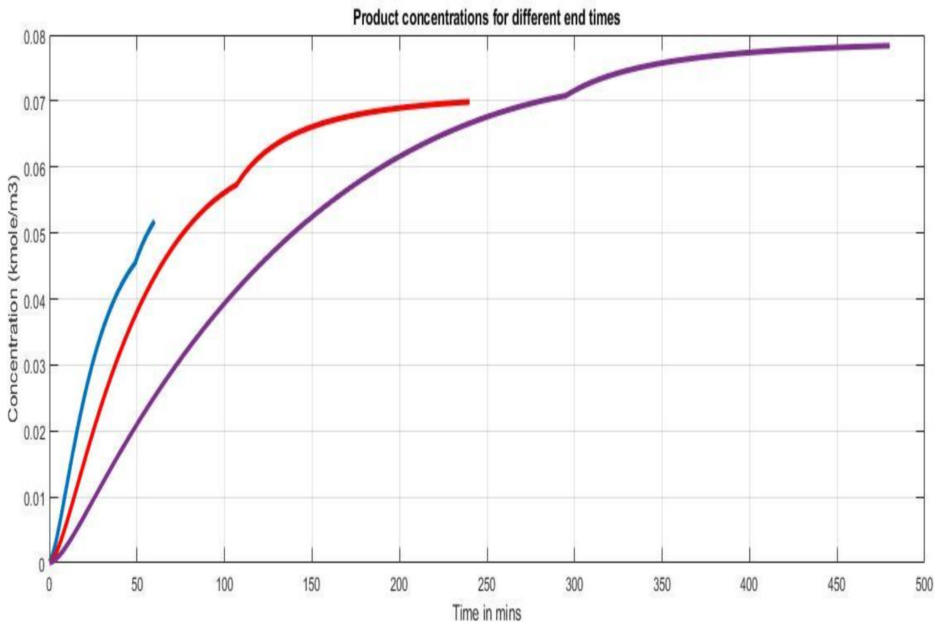


Figure 4-23 Product concentrations for different end times for example 2

Table 4-6 Pump failure scenario: No pumping between the 20th and the 60th minute

Pump failure scenario: No pumping between the 20th and the 60th minute	
Intelligent intervention final concentration of product	0.06 (Kmol/m3)
No intelligent intervention final concentration of product	0.048 (Kmol/m3)

Table 4-7 Minimum time for the maximum final concentration of product

Minimum time for the maximum final concentration of product	**Final maximum concentration of final product (kmol/m3)**
675 mins	0.081

Table 4-8 Final optimum mole fraction (Robust Q-learning)

Final time	**1 hr**	**2 hr**	**4 hr**	**8 hr**
Final optimum mole fraction (Robust Q-learning)	0.0517	0.0625	0.07	0.0784

The Matlab codes for solving this case study is presented in the appendix.

4.3. A semi-batch reactor {minimum time problem}:

- Consider a chemical reaction system found in [2]: $A+B \rightarrow C$
- Objective: Minimize the time needed to produce a given amount of C.
- Manipulated variable: Feed rate of B.
- Constraints: Input bounds; constraint on the maximum temperature reached under cooling failure; constraint on the maximum volume.
- Comments: In the case of a cooling failure, the system becomes adiabatic. The best strategy is to immediately stop the feed. Yet, due to the presence of un reacted components in the reactor, the reaction goes on. Thus, chemical heat will be released, which causes an increase in temperature. The maximum attainable temperature under cooling failure is given by:

$$T_d(t) = T(t) + min(C_A(t), C_B(t)) \frac{(-\Delta H)}{\rho C_p} \qquad \textbf{(4-6)}$$

- **The equations describing the reactor and model parameters:**

$$\dot{C}_A = -KC_AC_B - \frac{u}{V}C_A, C_A(0) = C_{A0} \qquad \textbf{(4-7)}$$

$$\dot{C}_B = -KC_AC_B - \frac{u}{V}(C_{Bin} - C_B),\ C_B(0) = C_{B0} \qquad \textbf{(4-8)}$$

$$\dot{V} = u\,, V(0) = V_0 \qquad \textbf{(4-9)}$$

$$C_c = \frac{C_{A0}V_0 + C_{C0}V_0 - C_AV}{V} \qquad \textbf{(4-10)}$$

Table 4-9 Parameters used in case study 3

k	0.0482	l/mol h
T	70	°C
ΔH	−60 000	J/mol
ρ	900	g/l
c_p	4.2	J/gK
$c_{B\,\mathrm{in}}$	2	mol/l
$u_{\min}$	0	l/h
$u_{\max}$	0.1	l/h
$T_{\max}$	80	°C
$V_{\max}$	1	l
$n_{C\,\mathrm{des}}$	0.6	mol
$c_{A\,o}$	2	mol/l
$c_{B\,o}$	0.63	mol/l
V_o	0.7	l

- Since isothermal conditions are chosen, the condition $T_{cf}(t) < T_{max}$ implies $C_B(t) < C_{Bmax}$,

- With C_B equal to $\frac{\rho C_p(T_{max}-T)}{-\Delta H}$ **(4-11)**

- Furthermore, the initial conditions correspond to having as much B as possible, i.e. $C_{Bo}=C_{Bmax}=0.63$ mol/l.

- The solution with the robust Q-learning was identical to the one provided by Bonvin [2], the minimum time to reach product concentration of 0.6 mol/l was 19.8 hr which shows great potential of the robust Q-learning method.

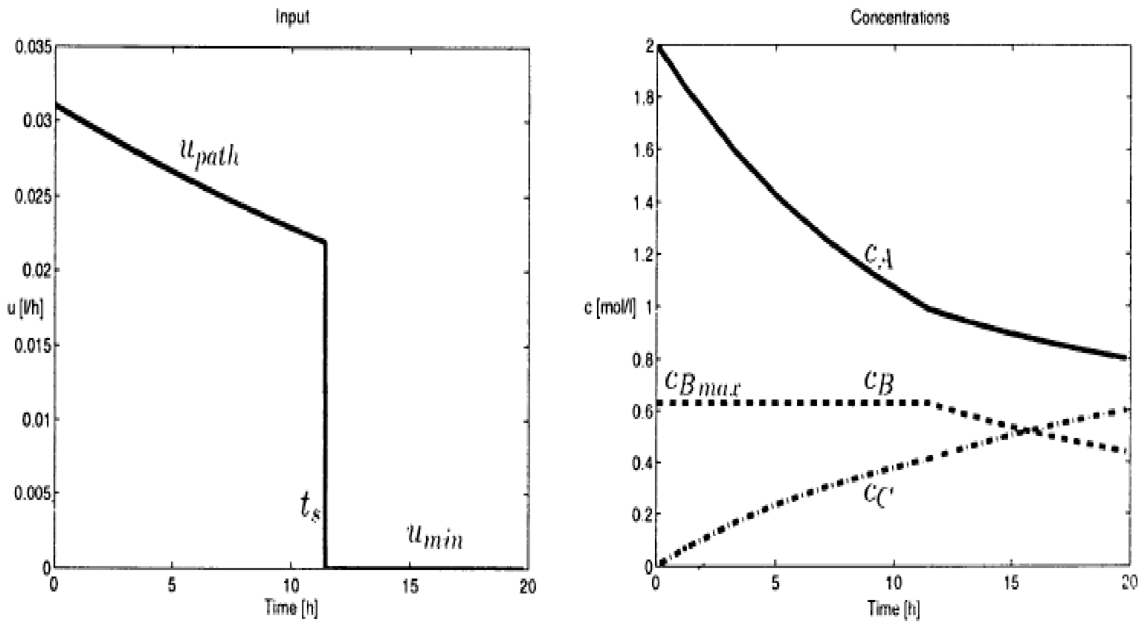


Figure 4-24 Optimum feed profile and concentrations for example 3 using analytical solution

(Source:[2] **page:13)**

- While there were attempts to solve this case study with the other methods Shown before they all failed to obtain results. The reference analytical solution is shown in figure 4-24. The robust Q- learning solution for the case study in the case nothing goes wrong is shown in figure 4-25, figure 4-26 and figure 4-27.

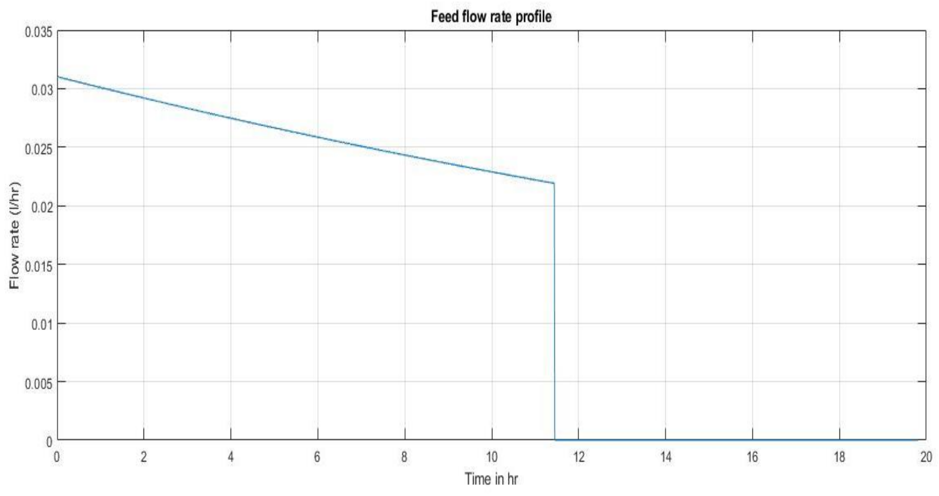


Figure 4-25 Optimum feed profile for example 3 using robust Q-learning algorithm

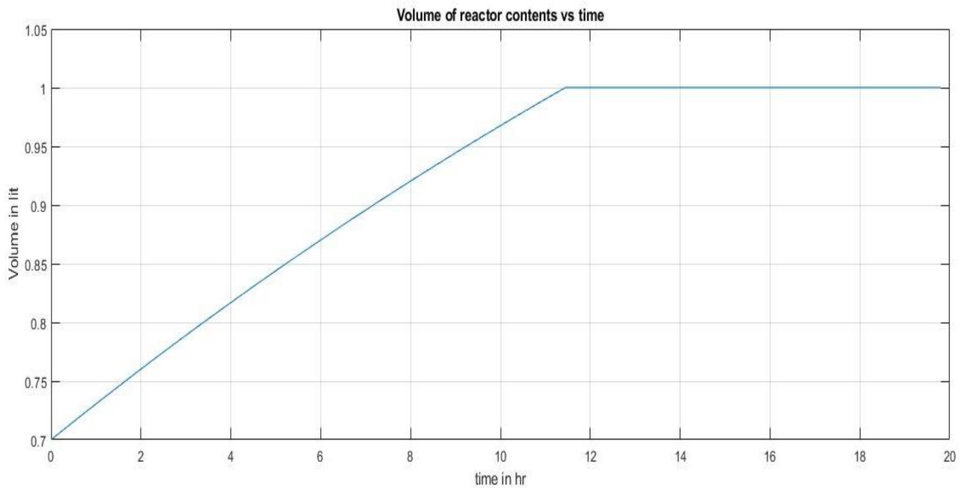


Figure 4-26 Volume response for example 3 using robust Q-learning algorithm

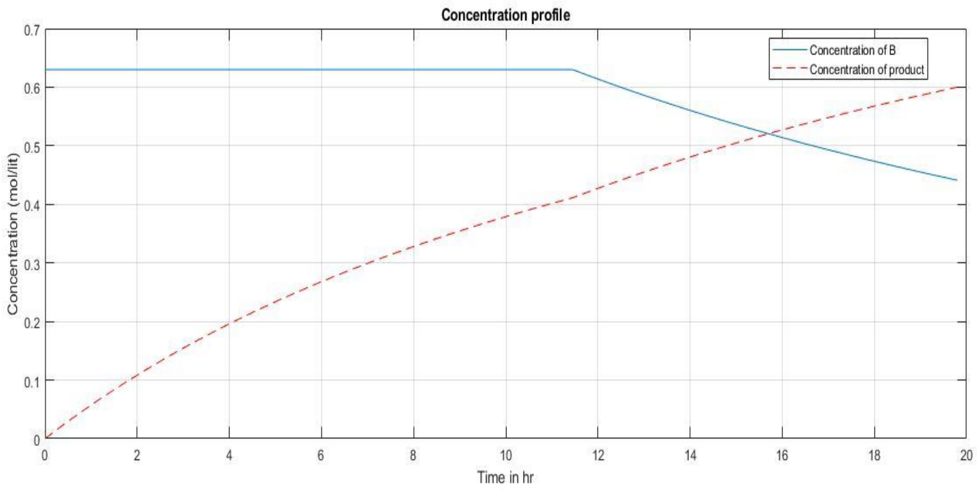


Figure 4-27 Optimum concentration profile for example 3 using robust Q-learning algorithm

we now will assume again a scenarios to proof the robustness of the method let's assume that the batch reactor feed flow rate is controlled by a pump, the ideal scenario is what the previous results showed us. what if the pump blocked or failed to pump and the flow rate fell down to zero. with our proposed method the intelligent intervention will assure the optimum path after the failure has been solved. which will be different than the ideal situation optimum path. The feed flow is assumed to stop from the 20th 5th till the 10th hours, the feed profile is shown in figure 4-28 and the concentration response is in figure 4-29.

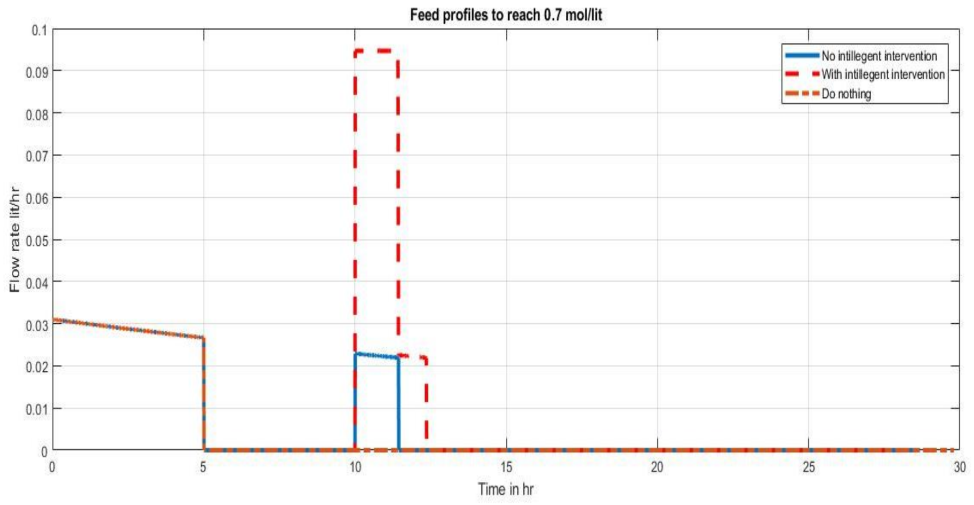


Figure 4-28 Pump failure scenario for example 3 using robust Q-learning algorithm

The optimum feed profile for different end concentration and the final product concentration profile are shown in figure 4-30 & figure 4-31 respectively.

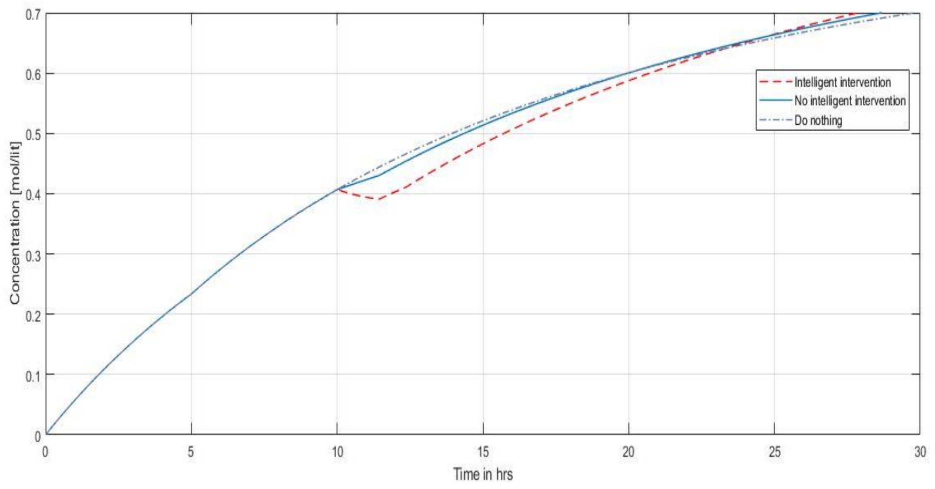


Figure 4-29 Pump failure scenario concentration response for example 3 using robust Q-learning algorithm

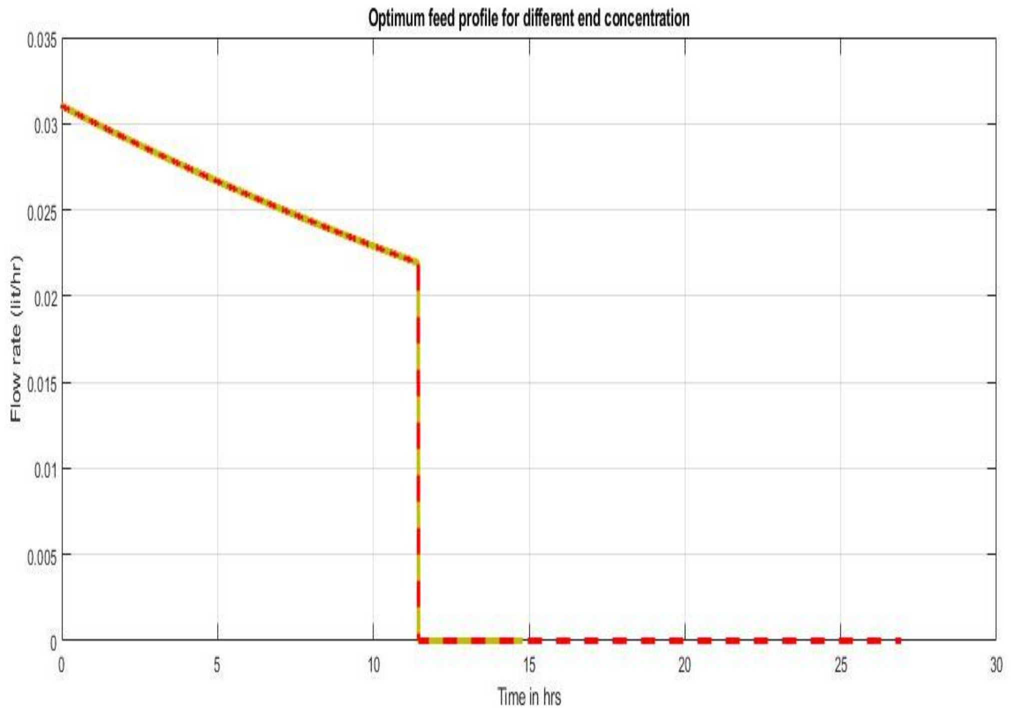


Figure 4-30 Optimum feed profile for different end concentration for example 3 using robust Q-learning algorithm

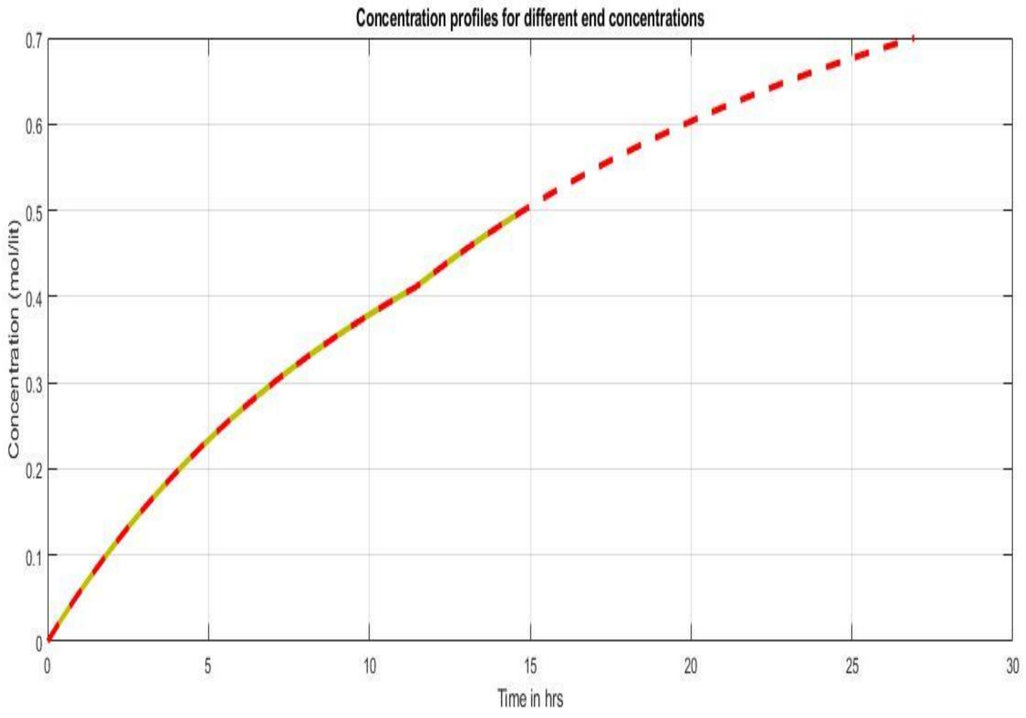


Figure 4-31 Optimum concentration profiles for different end concentration for example 3 using robust Q-learning algorithm

The rate of formation of the reactant species is shown in figure 4-32 and as it shown the rate of formation absolute value decreases as the reaction progress and it approaches zero value.

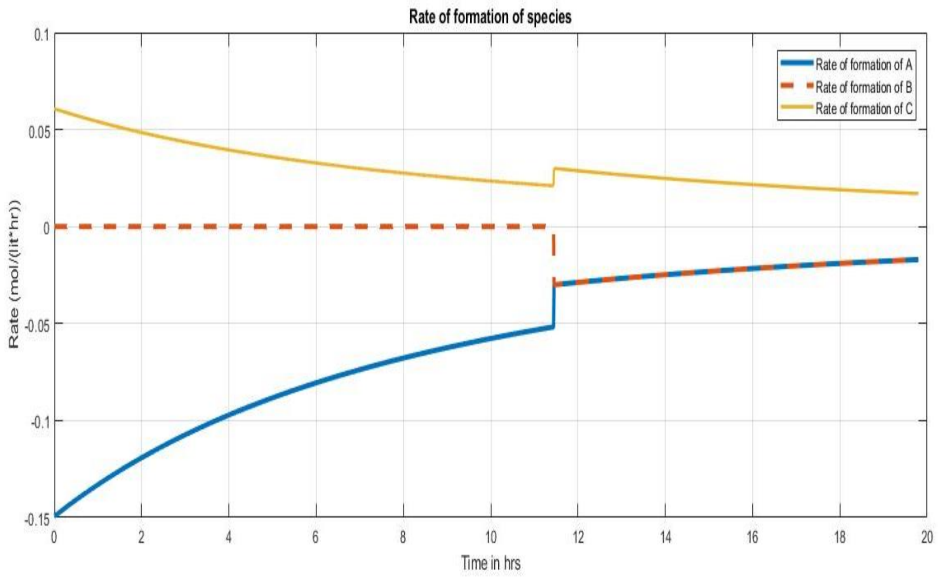


Figure 4-32 Rate of formation for example 3 using robust Q-learning algorithm

Table 4-10 Final optimum time for various end concentrations

Final Concentration	**0.5 mole/lit**	**0.6 mole/lit**	**0.7 mole/lit**
Final optimum time (Robust Q-learning)	**14.8 hr**	**19.8 hr**	**26.95 hr**

Table 4-11 Pump failure scenario results

Pump failure scenario: No pumping between the 5th and the 10th hr	
Intelligent intervention [final time of to reach 0.7 mol/lit]	**27.86 hr**
No intelligent intervention [final time to reach 0.7 mol/lit]	**28.7 hr**
Do nothing [final time to reach 0.7 mol/lit]	**29.81 hr**

5. CONCLUSION

The modified method called robust Q learning showed good results for the dynamic optimization of a batch reactor. It have where This method maps between the states and policy instead of the policy and time which gives a huge advantage over other methods, cause it increases the robustness and reliability of the optimum profile as it can adjust the profile for any unplanned changes during the process.

The method was tested on three different case studies and showed great potential where, the method corrected the paths of the optimal trajectory when exposed to difficulties in the process. The corrected path showed a better final optimum value for all cases.

The method needs further experiments on more complicated processes to show its great advantage as the introduction of this method to chemical engineering problems will benefit both the manufacturer and the operation engineers in case of crisis if met in their plants.

REFERENCES

[1] D. Bonvin, “Optimal operation of batch reactors—a personal view,” *J. Process Control*, vol. 8, no. 5–6, pp. 355–368, 1998.

[2] B. Srinivasan, D. Bonvin, E. Visser, and S. Palanki, “Dynamic optimization of batch processes,” *Comput. Chem. Eng.*, vol. 27, no. 1, pp. 27–44, 2003.

[3] D. Bonvin, “Control and Optimization of Batch Processes,” *Encycl. Syst. Control. Springer-Verlag London*, 2014.

[4] J. Zhang, “Batch-to-batch optimal control of a batch polymerisation process based on stacked neural network models,” *Chem. Eng. Sci.*, vol. 63, no. 5, pp. 1273–1281, 2008.

[5] Y. Tian, J. Zhang, and J. Morris, “Optimal control of a fed-batch bioreactor based upon an augmented recurrent neural network model,” *Neurocomputing*, vol. 48, no. 1–4, pp. 919–936, 2002.

[6] F. Caccavale, M. Iamarino, F. Pierri, and V. Tufano, *Control and Monitoring of Chemical Batch Reactors*. 2011.

[7] A. E. Bryson, “Applied Optimal Control: Optimization, Estimation and Control.” p. 496, 1975.

[8] D. E. Kirk, “Optimal Control Theory: An Introduction,” *IEEE Transactions on Automatic Control*, vol. 17, no. 3. p. 452, 2004.

[9] T. F. Edgar, D. Himmelblau, and L. Lasdon, *Optimization of chemical processes*. 2001.

[10] S. Støren and T. Hertzberg, “The sequential linear quadratic programming algorithm for solving dynamic optimization problems - a review.,” *Comput. Chem. Eng.*, vol. 19, no. SUPPL. 1, pp. 495–500, 1995.

[11] D. S. Naidu, *Optimal control Systems*. 2010.

[12] S. R. Upreti, *Optimal control for chemical Engineers*, vol. 91. 2017.

[13] D. E. Seborg, T. F. Edgar, and D. A. Mellichamp, “Process dynamics and control.” p. 713, 2004.

[14] C. Tricaud and Y. Chen, “Time-optimal control of systems with fractional dynamics,” *Int. J. Differ. Equations*, vol. 2010, 2010.

[15] S. H. Zak and W. A. Crossley, “Discrete-Time Synergetic Optimal Control of Nonlinear Systems,” *J. Guid. Control. Dyn.*, vol. 31, no. 6, pp. 1561–1574, 2008.

[16] M. H. Ghasemi, N. Kashiri, and M. Dardel, “Time-optimal trajectory planning of robot manipulators in point-to-point motion using an indirect method,” *Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci.*, vol. 226, no. 2, pp. 473–484, 2012.

[17] M. P. Nandakumar, “Determination of Compromise Solutions for Linear Steady State Regulator with Vector Valued Performance Index,” vol. 3, no. 11, pp. 1–5, 2013.

[18] R. E. Bellman, “Dynamic Programming,” *Princet. Univ. Press. Princeton, N J*, 1957.

[19] F. Irani, “On Dynamic Programming Technique Applied to a Parallel Hybrid Electric Vehicle,” *Control*, 2009.

[20] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA., 1995.

[21] H. Li, D. Liu, Q. Wei, D. Wang & X. Yang, “Adaptive dynamic programming with applications in optimal control,” *Springer Int. Publ.*, vol. Jan 2017.

[22] R. Luus, “Optimal control of batch reactors by iterative dynamic programming,” *J. Process Control*, vol. 4, no. 4, pp. 218–226, 1994.

[23] M. Kubat, “Neural networks: a comprehensive foundation by Simon Haykin, Macmillan, 1994, ISBN 0-02-352781-7.,” *The Knowledge Engineering Review*, vol. 13, no. 4. pp. 409–412, 1999.

[24] F. Rosenblatt, “Preceptrons and the theory of brain mechanisms,” 1961.

[25] H. Lodish, A . Berk & SL. Zipursky, *Molecular Cell Biology. 4th edition.* 2000.

[26] S. Haykin, *Neural Networks and Learning Machines: A Comprehensive Foundation*. 2008.

[27] S. Esteban, “Nonlinear flight control using adaptive critic based neural networks,” 2002.

[28] J. Zhang, “Introduction to Neural Networks,” Newcastle University.

[29] M. Hagan, H. Demuth and M. Beale, *Neural Network Design*. PWS Publishing Company, 1999.

[30] J. F.Smuts, *Process Control for Practitioners*. OptiControls, 2011.

[31] P. Werbos, “Neuro control and Supervised Learning: An Overview and Evaluation,” in *Handbook of Intelligent Control*, Van Nostr and Reinhold, 1992.

[32] S. N. Balakrishnan and G. Saini, “Adaptive Critic Based Neurocontroller for Auto landing of Aircraft with Varying Glide slopes,” *Proc. IEEE Int. Conf. Neural Networks*, vol. 4, pp. 2288–2291, 1997.

[33] S. N.. Balakrishnan, and V. Biega, “Adaptive Critic Based Neural Networks for Aircraft Optimal Control,” *J. Guid. Control Dyn.*, vol. 19, no. 4, pp. 893–898.

[34] C. . Anderson, “Learning to Control an Inverted Pendulum using NeuralNetworks,” *IEEE Control Syst. Mag.*, vol. 9, pp. 33–37.

[35] DARPA, *Neural Network Study*. Lexinton, MA: MIT Lincoln Laboraroty, 1988.

[36] V. Cortes, "Support vector Networks," *Mach. Learn.*, vol. 20, pp. 273–297.

[37] A. Ben-Hur, D. Horn, H. Siegelmann and V. Vapnik, "Support vector clustering," *J. Mach. Learn. Res.*, vol. 2, pp. 125–137, 2001.

[38] Dnl Institute, "Building predective model using svm and r," 2015. [Online]. Available: http://dni-institute.in/blogs/building-predictive-model-using-svm-and-r/.

[39] L. T. Christopher J.C. Burges (Bell Laboratories, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.*, vol. 2, pp. 121–167, 1998.

[40] N. Bambrick, "Support vector machines for dummies," 2016. [Online]. Available: http://blog.aylien.com/support-vector-machines-for-dummies-a-simple/.

[41] J. S.-T. Nello Cristianini, *An introduction to support vector machines and kernell methods*. Cambridge University Press, 2000.

[42] L. C. Baird, "Residual Algorithms: Reinforcement Learning with Function Approximation," in *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[43] J. Si, A. Barto, W. Powell, D. Wunsch, G. G. Lendaris, and J. C. Neidhoefer, "HANDBOOK of LEARNING and APPROXIMATE DYNAMIC PROGRAMMING Chapter 4: Guidance in the Use of Adaptive Critics for Control GUIDANCE IN THE USE OF ADAPTIVE CRITICS FOR CONTROL," pp. 97–124, 2004.

[44] A. Gosavi, "Reinforcement Learning: A Tutorial Survey and Recent Advances," *INFORMS J. Comput.*, vol. 21, no. 2, pp. 178–192, 2008.

[45] J. Randløv and P. Alstrøm, "Learning to Drive a Bicycle using Reinforcement Learning and Shaping," *Proc. Int. Conf. Mach. Learn.*, pp. 463–471, 1998.

[46] M. E. Harmon, L. C. Baird, and A. Klopf, "Advantage Updating Applied to a Differential Game.," in *Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference, MIT Press, Cambridge, Massachusetts.*, 1994.

[47] J. Gimsa *et al.*, "Spermidine-Induced Attraction of Like-Charged Surfaces Is Correlated with the pH-Dependent Spermidine Charge: Force Spectroscopy Characterization," *Langmuir*, vol. 34, no. 8, pp. 2725–2733, 2018.

[48] E. W. Weisstein, "Backgammon," *Mathworld*. [Online]. Available: http://mathworld.wolfram.com/Backgammon.html.

[49] C. W. Anderson, "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control Syst. Mag.*, vol. 9, no. 3, pp. 31–37, 1989.

[50] J. A. Boyan and A. W. Moore, “Generalization in Reinforcement Learning: Safely Approximating the Value Function,” *Adv. Neural Inf. Process. Syst. 7*, pp. 369–376, 1995.

[51] A. Dutech *et al.*, “Reinforcement Learning Benchmarks and Bake-offs II,” *Nips*, pp. 1–50, 2005.

[52] M. E. Harmon, L. C. Baird, and A. Klopf, “Reinforcement learning applied to a differential game. Adaptive Behavior.,” *MIT Press*, vol. 4, no. 1, pp. 3–28, 1995.

[53] M. E. Harmon and S. S. Harmon, “Reinforcement Learning: A Tutorial Scope of Tutorial.”

[54] C. Watkins, “Learning from delayed rewards,” Cambridge University, 1989.

[55] C. Watkins, P. Dayan, “Q-Learning,” *Mach. Learn.*, vol. 8, pp. 179–292, 1992.

[56] R. S. Sutton and A. G. Barto, *Reinforcement Learning, an Introduction*. MIT Press, 1998.

[57] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-Based Batch Mode Reinforcement Learning,” *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, 2005.

[58] N. Shimkin, “Learning in complex systems,” *Isreal Intitute of Technology*, 2011. .

[59] R. E. Bellman, “A Markovian Decision Process,” *J. Math. Mech.*, 1957.

[60] E. Altman, *Constrained Markov decision processes*. 1999.

[61] J. Zhang. C. Cardona, M. Baron, “Re-optimisation control of a fed-batch fermentation process using bootstrap aggregated extreme learning machine,” in *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO2017)*, 2017.

[62] R. Fisher, J. Zhang, “Reliable Multi-objective On-Line Re-optimisation Control of a Fed-Batch Fermentation Process Using Bootstrap Aggregated Neural Networks,” in *Computer Science and Intelligent Controls (ISCSIC)*, 2017.

[63] J. Zhang. Z.Xiong, Y. Xu, J. Dong, “Neural network based iterative learning control for product qualities in batch processes,” *Int. J. Model. Identif. Control*, vol. 11, no. 1–2, pp. 107–114, 2010.

[64] L. A. Houben C, “Automatic discovery and optimization of chemical processes,” *Curr. Opin. Chem. Eng.*, vol. 1, no. 9, pp. 1–7.

[65] N. P. Subramanian, F. Ghouse, “Fault diagnosis of batch reactor using machine learning methods.,” *Model. Simul. Eng.*, vol. 1.

[66] M. S. Iyer and D. C. Wunsch, “Dynamic reoptimization of a fed-batch fermentor using adaptive critic designs,” *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1433–1444, 2001.

[67] N. Jin, D. Wang, D. Liu, Q. Wei and D. Zhao, “Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming,” *Automatica*, vol. 48, no. 8, 2012.

[68] K. Arulkumaran, M. P. Deisenroth, M. Brundage, “Deep reinforcement learning: A brief survey,” *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38.

[69] N. Sistani, B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, “Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics,” *Automatica*, vol. 50, no. 4, pp. 1167–1175, 2014.

[70] D.Wang, B. Luo, D. Liu, T. Huang, “Model-free optimal tracking control via critic-only Q-learning,” *IEEE Trans. neural networks Learn. Syst.*, vol. 27, no. 10, pp. 2134–2144.

[71] L. T. Logsdon, J.S. and Biegler, *Comput. Chem. Engng.*, vol. 17, p. 367, 1993.

[72] P. Terwiesch, D. Ravemark, B. Schenker, and D. W. T. Rippin, “Semi-batch process optimization under uncertainty: Theory and experiments,” *Comput. Chem. Eng.*, vol. 22, no. 1–2, pp. 201–213, 1998.

[73] D. B. D.Ruppen, D.W.T Rippin, “An online optimization strategy for fast batch and semi batch reaction systems,” *IFAC Adv. Control Chem. Process.*, 1991.

Appendix

1. Matlab code for batch reactor dynamic programming:

```
clear all
clc
p=100;
N=99;
M=99;
Tau=1/p;
r=100;
%% generating all possible actions
for o=0:N-1
un0(o+1,:)=[(298+(r/N)*o)*ones(1,p)];
end
for o1=0:M-1;
um0(o1+1,:)=[(398-(r/M)*o1)*ones(1,p)];
end

for trial=1:40
trial
clearx1nx2nj1j2urrRCcc1r1zc
   %% generating all possible states
   x1n=ones(N,1);
   x2n=zeros(N,1);
if trial==1
uN=un0;
end
for n=1:N;
for k= 1:p;
u(k)=uN(n,k);
         x1n(n,k+1)=x1n(n,k)+Tau* (-4000*x1n(n,k)^2*exp(-2500/u(k)));
         x2n(n,k+1)=x2n(n,k)+Tau*(4000*x1n(n,k)^2*exp(-2500/u(k))-
620000*x2n(n,k)*exp(-5000/u(k)));
end
end
   %% finding the last step optimal action
if trial ==1
um=um0;
end
for n=1:N;
for m=1:M;
u(k)=um(m,p);
         x1n(n,k+1)=x1n(n,k)+Tau* (-4000*x1n(n,k)^2*exp(-2500/u(k)));
         x2n(n,k+1)=x2n(n,k)+Tau*(4000*x1n(n,k)^2*exp(-2500/u(k))-
620000*x2n(n,k)*exp(-5000/u(k)));
dx2p(m)=x2n(n,k+1);
```

```
end
        [rr(n),c(n)]=max(dx2p);
end
    [Ipxnmax(p)]=max(rr);
u(p)=um(c(xnmax(p)),p);
    %% finding all the optimal actions before the last step
for L=0:p-2;
        k= p-L-1;
for n=1:N;
for m=1:M;
u(k)=um(m,k);
                x1m(m,k+1)=x1n(n,k)+Tau*(-4000*x1n(n,k)^2*exp(-2500/u(k)));
                x2m(m,k+1)=x2n(n,k)+Tau*(4000*x1n(n,k)^2*exp(-2500/u(k))-
620000*x2n(n,k)*exp(-5000/u(k)));
X(m)=[abs(x2m(m,k+1)-x2n(xnmax(k+1),k+1))];
end
            [r1 c1(n)]=min(X);
u(k)=um(c1(n),k);
for k1=p-L:p
                x1m(c1(n),k1+1)=x1m(c1(n),k1)+Tau*(-4000*x1m(c1(n),k1)^2*exp(-
2500/u(k1)));
                x2m(c1(n),k1+1)=x2m(c1(n),k1)+Tau*(4000*x1m(c1(n),k1)^2*exp(-
2500/u(k1))- 620000*x2m(c1(n),k1)*exp(-5000/u(k1)));
end
It(n)=x2m(c1(n),end);
end
        [Jmaxupos]=max(It);
xnmax(k)=upos;
u(k)=um(c1(upos),k);
end
    %% refining the actions to a more optimal actions
    r=0.9*r;
for k=1:p;
forHm=2:2:M-1;

um(M,k)=u(k);
um(Hm,k)=(u(k)+r*Hm/(M-1));
um(Hm/2,k)=u(k)-r*Hm/(M-1);
if u(k)+r*Hm/(M-1)> 398;
um(Hm,k)=398;
end
if u(k)-r*Hm/(M-1) < 298;
um(Hm/2,k)=298;
end
end
end
for k=1:p;
for H=2:2:N-1;
```

```
uN(N,k)=u(k);
uN(H,k)=(u(k)+r*H/(N-1));
uN(H/2,k)=u(k)-r*H/(N-1);
if u(k)+r*H/(N-1)> 398;
uN(H,k)=398;
end
if u(k)-r*H/(N-1) < 298;
uN(H/2,k)=298;
end
end
end
end
%% testing the results
for k= 1:p;
x1(1)=1;
x2(1)=0;
x1(k+1)=x1(k)+Tau*(-4000*x1(k)^2*exp(-2500/u(k)));
x2(k+1)=x2(k)+Tau*(4000*x1(k)^2*exp(-2500/u(k))- 620000*x2(k)*exp(-
5000/u(k)));
end
holdon
plot(0:0.01:0.99,u)
final(trial)=x2(end)
```

2. Matlab code for left or right problem:

```
% The problem of left or right its target is to find the best direction of
% movement either left or right to gain maximum rewards by reaching the left or the
right end
% starting from different initial points and a penalty is taken on each step
clc; clearvars;
%% generation of fourtuples date set
episodes= 500;                    % number of episodes
u=[2,-2];                         % possible actions
j=0;                              % initial total steps
for N=0:episodes;
   x=randi([0 10]);               % random generation of initial state from 0 to 10
i=0;                              % initial step for each episode
while x(i+1)<=10 && x(i+1)>=0   % loop for the whole episode
      a=randi([1 2]);             % random generation of action 2 or -2
i=i+1;                            % episode steps
      j=j+1;                      % total steps
x(i+1)=x(i)+u(a);                 % The action-state model describing the system
if x(i+1)>=10;                    % conditions if the final state reached the right end
x(i+1)=10;
r(j)=(0.75^(i-1))*100;  % reward value obtained when reaching right end after a
number of episodes steps
action(j)=u(a);
state(j)=x(i);
Next_state(j)=x(i+1);
break
elseif x(i+1)<=0 ;                % conditions if the final state reached the right end
x(i+1)=0;
r(j)=(0.75^(i-1))*50;    % reward value obtained when reaching right end after a
number of episodes steps
action(j)=u(a);
state(j)=x(i);
Next_state(j)=x(i+1);
break
else
r(j)=0;                  % reward value obtained inside the range of ]0,10[
action(j)=u(a);
state(j)=x(i);
Next_state(j)=x(i+1);
end
end
end
end
  %% Reward value iteration function
Fourtuples_size=j;
value_iterations=10 ;
Possible_actions=u;
```

```
reward=r ;
Fourtuples=[state; action; reward; Next_state];
net=Reward(Fourtuples, Fourtuples_size, value_iterations, Possible_actions);
      %% Simulation of the final results
      Input1=[0:0.1:10;2*ones(1,101)];      % first possible action on all states
      Input2=[0:0.1:10;-2*ones(1,101)];     % second possible action on all states
      Qn2=net(Input1);
      Qnn2=net(Input2);
% plots of the Q-function for the first & second possible action.
% the values of Qnn2 higher than Qn1 will determine the optimum action value
stairs(0:0.1:10,Qn2);
holdon
stairs(0:0.1:10,Qnn2);
holdoff
%% This function is for the value iteration algorithm
function net= Reward(fourtuples,fourtuples_size, Number_value_iterations,
possible_actions)
oV1=fourtuples(3,:);               % initial value function
net=fitnet([35,25],'trainscg');      % Feed foreward neural network using SCG algorithm
with 2 hidden layers
for N = 0:Number_value_iterations    % number of value function iterations
display(N)
net=train(net,[fourtuples(1,:);fourtuples(2,:)],oV1); % Training of the NN
testactions=[net([fourtuples(4,:);possible_actions(1)*ones(1,fourtuples_size)])...;net([f
ourtuples(4,:);possible_actions(2)*ones(1,fourtuples_size)])];     %testing all the
possible actions
   oV0=max(testactions);
   oV1=fourtuples(3,:)+ 0.75*oV0;
end
```

3. Simple batch reactor Matlab codes:

```
function y=dadeo(t,x,a )
y(1)=-4000*x(1)^2*exp(-2500/a);
y(2)=4000*x(1)^2*exp(-2500/a)- 620000*x(2)*exp(-5000/a);
y(3)=1;
y=[y(1);y(2);y(3)];
```

```
function nx=sim_next(x,a)
if x(3) >= 1
    error('the next state will go beyond the end-time')
end

[Tn Yn]=ode45(@(t,y)dadeo(t,y,a),[0 0.1],x);
nx(1)=Yn(end,1);
nx(2)=Yn(end,2);
nx(3)=Yn(end,3);

nx(4) = 0; % store reward
if nx(3) == 1 % end of time
    nx(4) = nx(2); % reward = nx(2)
end
```

```
clc; clear;rng default;
j=0;                %Counter
steps=10;
for episodes=1:100    %Number of episodes to generate the
data
    x0(1)=rand;
    x0(2)=rand;
    x0(3)=0;
for i=0:steps-1
        a=randi([298 398]);
        [T Y]=ode45(@(t,y)dadeo(t,y,a),[i/steps
(i+1)/steps],x0);
if i>=0
          j=j+1;
        action(j)=a;
        state1(j)=x0(1);
        state2(j)=x0(2);
        state3(j)=x0(3);
end
      x0=Y(end,:);
end
end
sMx=[state1;state2;state3]';
sMx = unique(sMx,'rows');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m =1000; % number of samples
k = 11 ; % number of actions
N_max = 49; % number of iterations for the reinforcment learning
% sMx = rand(m,3); % To be generated vis several episodes (work in progress)
aV = [298:10:398];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parallelism
myCluster = parcluster('local');
myCluster.NumWorkers = 8;
saveProfile(myCluster);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
parfor i = 1 : m
for j = 1:k
        NSCube(i,j,:) = sim_next(sMx(i,:),aV(j)); % stores next_state_vect & the
associated reward
end
end
opt_aV = zeros(m,1);
qV = -inf*ones(size(opt_aV));
for N = 0:N_max
    tic;
for i = 1 : m
for j = 1:k
            nsV = reshape(NSCube(i,j,1:3),1,3);
            nq = NSCube(i,j,4); % the reward associated with next state
if N>0
                nq = nq + max(predict(Mdl,[repmat(nsV,k,1),aV.'])); % the reward +
the best q_value for the next state -- given that we search for the optimal next
action
end
if  nq > qV(i)
                qV(i) = nq;
                opt_aV(i) = aV(j);
end
end
end
    Mdl =
fitrsvm([sMx,opt_aV],qV,'Standardize',true,'KernelFunction','Polynomial','PolynomialO
rder',2);
    toc;N
end
```

```
%%
x0f=[1 0 0];
n
etB= fitrsvm([sMx(:,1),sMx(:,2),sMx(:,3)],opt_aV);
for jj=0:99
act(jj+1)=predict(netB,[x0f(1),x0f(2),x0f(3)]);

Ideal=[377.960983134561,373.289696394048,369.384135827649,366.040113675455,363.124331
077106,360.545122497176,358.236856462802,356.150999149416,354.250697607175,352.507342
874915,350.898305262324,349.405393722408,348.013778878922,346.711222229720,345.487513
008385,344.334049222190,343.243520883927,342.209667027149,341.227086880020,340.291091
389529,339.397585217160,338.542972030728,337.724077808277,336.938088213279,336.182497
067809,335.455063656135,334.753777112166,334.076826533026,333.422575754182,332.789541
944758,332.176377352991,331.581853664529,331.004848539836,330.444333978384,329.899366
221777,329.369076959310,328.852665640645,328.349392733475,327.858573790986,327.379574
215876,326.911804625652,326.454716738733,326.007799713141,325.570576879692,325.142602
820093,324.723460747449,324.312760152617,323.910134684915,323.515240239899,323.127753
230578,322.747369021487,322.373800507674,322.006776822921,321.646042163435,321.291354
714943,320.942485672541,320.599218343901,320.261347327552,319.928677758850,319.601024
617124,319.278212088166,318.960072976886,318.646448165513,318.337186113193,318.032142
393265,317.731179264906,317.434165276141,317.140974895516,316.851488170018,316.565590
407033,316.283171878374,316.004127544570,315.728356797783,315.455763221896,315.186254
368395,314.919741546851,314.656139628866,314.395366864475,314.137344710070,313.881997
667001,313.629253130070,313.379041245207,313.131294775673,312.885948976193,312.642941
474461,312.402212159515,312.163703076502,311.927358327422,311.693123977432,311.460947
966353,311.230780025044,311.002571596317,310.776275760109,310.551847162643,310.329241
949329,310.108417701162,309.889333374412,309.671949243407,309.456226846215,309.242128
933056];
% if jj>50
%  act(jj+1)=Ideal(jj+1);
% end
if jj>20 && jj<50
    act(jj+1)=298;
end

    [T Yf]=ode45(@(t,y)dadeo(t,y,act(jj+1)),[jj/100 (jj+1)/100],x0f);
    x0f=Yf(end,:);
    zz(jj+1)=Yf(end,2);
end
finalproduct(N+1)=Yf(end,2)
plot(0:1:jj,act);
```

4. Fed batch reactor Matlab code:

```
function yp=terw1(t,y,f)
k1=0.5;
k2=0.5;
bf=0.2;
yp=[-k1*y(1)*y(2)-y(1)*f/y(5);
   -k1*y(1)*y(2)-2*k2*y(2)*y(2)+(bf-y(2))*f/y(5);
   k1*y(1)*y(2)-y(3)*f/y(5);
   2*k2*y(2)*y(2)-y(4)*f/y(5);
    f;1];
```

```
function nx=sim_next(x,a)



[Tn Yn]=ode45(@(t,y)terw1(t,y,a),[0 12],x);
nx(1)=Yn(end,1);
nx(2)=Yn(end,2);
nx(3)=Yn(end,3);
nx(4)=Yn(end,4);
nx(5)=Yn(end,5);
nx(6)=Yn(end,6);

nx(7) = 0; % store reward
if nx(6) >= 119 % end of time
    nx(7) = 10*nx(3); % reward = nx(2)
```

```
clc; clear;rng default;
j=0;                %Counter
steps=10;
for episodes=1:150    %Number of episodes to generate the data
    x0(1)=0.2;
    x0(2)=0;
    x0(3)=0;
    x0(4)=0;
    x0(5)=0.5;
    x0(6)=0;
for i=0:steps-1
        a=0.001*randi([0 10]);
if x0(5)>=0.99
            a=0;
end
        [T Y]=ode45(@(t,y)terw1(t,y,a),[i*12 (i+1)*12],x0);
if i>=0
          j=j+1;
        action(j)=a;
        state1(j)=x0(1);
        state2(j)=x0(2);
        state3(j)=x0(3);
        state4(j)=x0(4);
        state5(j)=x0(5);
        state6(j)=x0(6);
end
      x0=Y(end,:);
end
end
sMx=[state1;state2;state3;state4;state5;state6]';
sMx = unique(sMx,'rows');
```

```
m = 1140; % number of samples
k = 11 ; % number of actions
N_max = 4; % number of iterations for the reinforcment learning
% sMx = rand(m,3); % To be generated vis several episodes (work in progress)
aV = [0:0.001:0.01];
% Parallelism
myCluster = parcluster('local');
myCluster.NumWorkers = 8;
saveProfile(myCluster);
parfor i = 1 : m
for j = 1:k
        NSCube(i,j,:) = sim_next(sMx(i,:),aV(j)); % stores next_state_vect & the
associated reward
end
end
opt_aV = zeros(m,1);
qV = -inf*ones(size(opt_aV));
for N = 0:N_max
    tic;
for i = 1 : m
for j = 1:k
            nsV = reshape(NSCube(i,j,1:3),1,3);
            nq = NSCube(i,j,7); % the reward associated with next state
if N>0
                nq = nq + max(predict(Mdl,[repmat(nsV,k,1),aV.'])); % the reward + the
best q_value for the next state -- given that we search for the optimal next action
end
if  nq > qV(i)
                qV(i) = nq;
                opt_aV(i) = aV(j);
end
end
end
    Mdl =
fitrsvm([sMx(:,1:3),opt_aV],qV,'Standardize',true,'KernelFunction','Polynomial','Polyno
mialOrder',2);
    toc;N
end
```

```
%%
x0f=[0.2 0 0 0 0.5 0];
netB= fitrsvm([sMx],opt_aV);
for jj=0:120

Ideal=[0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.
0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.010000000
0000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0
.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.01000000
00000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,
0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000
000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000
,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.010000
0000000000,0.0100000000000000,0.0100000000000000,0.0100000000000000,0.009992241976641
82,0.00944271908496751,0.00889176798454548,0.00834276296290602,0.00779862456124113,0.
00726185177176751,0.00673454721413157,0.00621845884051629,0.00571500661112089,0.00522
532193849122,0.00475027632637114,0.00429051140371860,0.00384647068949028,0.0034184208
2138476,0.00300647839412170,0.00261063014135664,0.00223075138569050,0.001866625172391
76,0.00151795398527986,0.00118437907780765,0.000865484505306199,0.000560812489988280,
0.000269871837412325,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
    act(jj+1)=predict(netB,[x0f]);
if jj>20 && jj<60
         act(jj+1)=0.01;
end
%      if jj>60
%          act(jj+1)=Ideal(jj+1);
%      end
if act(jj+1)<0
        act(jj+1)=0;
end
if act(jj+1)>0.01
        act(jj+1)=0.01;
end

    [T Yf]=ode45(@(t,y)terw1(t,y,act(jj+1)),[jj:1:(jj+1)],x0f);
    x0f=Yf(end,:);
    zzp(jj+1)=Yf(end,3);
end
finalproduct(N+1)=Yf(end,3)
zza(N+1)=Yf(end,1);
zzb(N+1)=Yf(end,2);
zzd(N+1)=Yf(end,4);
zzv(N+1)=Yf(end,5);
plot(0:1:jj,act);
```

5. Case 3: Fed batch reactor Matlab code:

```
act = [];
xMx =[];
tt=0;
time_step = 0.01;
x0f=[2 0.63 0 0.7 0];
opt_aV_trial = opt_aV_Mx(:,2);
MdlB= fitrsvm(sMx,opt_aV_trial);
i=0;
while x0f(3)<=0.7
    i =i+1;
    xMx(i,:) = x0f;
    act(i)=predict(MdlB,([x0f(1),x0f(2),x0f(3),x0f(4),x0f(5)]));
if x0f(4)>=1
        act(i)= 0;
end
%     if tt<24
%      act(i)=ideal(i);
%     end
%     if tt>=24
%      act(i)=0;
%     end
%
if tt>5
        act(i)=0;
end


    new_tt = tt +time_step;
    xnf(2) = 1;
    delta = 0.0001;
      act(i) = act(i) +delta;
while xnf(2) > 0.63
        act(i) = act(i) -delta;
        [T Yf]=ode45(@(t,y)bonvin(t,y,act(i)),[tt,new_tt],x0f);
        xnf = Yf(end,:);
end
    tt = new_tt
    x0f=xnf;
end
finalproduct=Yf(end,5)
figure(1);plot(xMx(:,5),act.');
figure(2);plot(xMx(:,5),xMx(:,2));
figure(3);plot(xMx(:,5),xMx(:,3));
figure(4);plot(xMx(:,5),xMx(:,4));
```

```
function sMx = GetSMx()
time_step = 0.25;
i=0;
for epoch =1:50    %Number of episodes to generate the data
    epoch
    x0=[2 0.63 0 0.7 0];
while x0(3)<=0.6 && x0(4)<= 1
        i=i+1;
        sMx(i,:) = x0;
while true
            a=0.033*rand;
            [TVect XVect]= ...

ode45(@(t,x)bonvin(t,x,a),[x0(5),x0(5)+time_step],x0);
            x0=XVect(end,:);
if x0(2) <=0.63
break;
end% end if
end% end small-while
end% end big-while
end% end for epoch
```

```
function tf = ComputeRewardAfterSwitch(x0)

a =0;
time_step = 0.25;
while x0(3) < 0.6
%     if x0(5) > 200
%         error('ggg');
%     end
    [TVect
XVect]=ode45(@(t,x)bonvin(t,x,a),[x0(5),x0(5)+time_step],x0);
     x0=XVect(end,:);
end
tf = x0(5);
```

```
function nx=sim_next(x0,ac)

time_step = 0.25;
[TVect
XVect]=ode45(@(t,x)bonvin(t,x,ac),[x0(5),x0(5)+time_step]
,x0);
nx=XVect(end,:);

if nx(2) > 0.63 % Cb
    nx(6) = Inf;
elseif nx(3) >= 0.6
    nx(6) = nx(5);
elseif nx(4) < 1 % volumes less than 1
    nx(6) = 1000; % feasible point but did not reach the
switching time
else
    nx(6) = ComputeRewardAfterSwitch(nx);
end
```

```
function y=bonvin(t,x,a)
k1=0.0482;
bf=2;
y(1)=-k1*x(1)*x(2)-a*x(1)/x(4);
y(2)=-k1*x(1)*x(2)+a*(bf-x(2))/x(4);
y(3)=k1*x(1)*x(2)-a*x(3)/x(4);
y(4)=a;
y(5)=1;
y=[y(1);y(2);y(3);y(4);y(5)];
```

```
close all;clc; clear;rng default;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sMx = GetsMx();
sMx = unique(sMx,'rows');
m =size(sMx,1); % number of samples
N_max = 10; % number of iterations for the reinforcment learning
aV = 0:0.01:0.1;
k = size(aV,2); % number of actions
NSCube=zeros(m,k,6);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parallelism
myCluster = parcluster('local');
myCluster.NumWorkers = 8;
saveProfile(myCluster);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1 : m
parfor j = 1:k
        NSCube(i,j,:) = sim_next(sMx(i,:),aV(j)); % stores next_state_vect & the
associated reward
end
end
for N = 0:N_max
    tic;
    opt_aV = zeros(m,1);
    qV = inf*ones(size(opt_aV)); %the q_value associted with the current state
and current optmal action
for i = 1 : m
for j = 1:k
            nsV = reshape(NSCube(i,j,1:5),1,5);
            nq = NSCube(i,j,6); % the reward associated with next state
if N>0
                nq = nq + min(predict(Mdl,[repmat(nsV,k,1),aV.'])); % the reward
+ the best q_value for the next state -- given that we search for the optimal
next action
end
if  nq < qV(i)
                qV(i) = nq;
                opt_aV(i) = aV(j);
end
end
end
    Mdl = fitrsvm([sMx,opt_aV],qV);
    toc;N
%     plot(opt_aV);pause(10);
    opt_aV_Mx(:,N+1)=opt_aV;
end
```

نبذة مختصرة

يجب تطبيق التحسين الديناميكي للأنظمة الكيميائية غير الخطية - مثل المفاعلات الباتش - المباشر ، ويجب أن تكون السيطرة المناسبة التي يتم اتخاذها وفقًا للحالة الحالية للنظام بدلاً من اللحظة الحالية.

تستخدم أحدث حالة من الأساليب الفنية عنصر التحكم استنادًا إلى الوقت الحالي فقط. هذا غير مناسب لمعظم الحالات ، حيث أنه غير قوي للتغييرات المحتملة في النظام. تقترح هذه الأطروحة طريقة-Q للتعلم قوية متينة لإجراء تحسين قوي مباشر لمفاعل الباتش.

في هذه الأطروحة ، يتم تطبيق طريقة-Q للتعلم على مفاعل باتش بسيط ، من أجل إظهار فعالية الطريقة المقترحة حديثًا ، تتم مقارنة نتائجها بطريقة التعلم Q القوية المحسنة حيث تم الحصول على تحسن كبير.

تتمثل الميزة الرئيسية لطريقة-Q للتعلم أو الطريقة المقترحة في أنها يمكن أن تستوعب التغييرات غير المخطط لها أثناء العملية عن طريق تغيير التحكم في العملية.

نحاول زيادة المنتج النهائي الذي تم الحصول عليه أو تحقيق مواصفات معينة من المنتجات (أي تقليل المنتجات الجانبية).

ويمكن القيام بذلك باستخدام عدة طرق ، وهي ميزة أسلوبنا المقترح القادر على التغلب على التغيرات المفاجئة أثناء التفاعل.

المسار الأمثل لعمليات الشحنات بواسطة خورزميات الذكاء الاصطناعي

إعداد

عبد الرحمن سعيد ابراهيم عبدالقادر

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستير
ماجستير العلوم في الهندسة الكيميائية

كلية الهندسة
جامعة القاهرة
2019

المسار الأمثل لعمليات الشحنات بواسطة خورزميات الذكاء الاصطناعي

إعداد

عبد الرحمن سعيد ابراهيم عبدالقادر

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستير
ماجستير العلوم في الهندسة الكيميائية

تحت اشراف

الأستاذ الدكتور/ أحمد سليمان
أستاذ بقسم الهندسة الكيميائية
كلية الهندسة، جامعة القاهرة

الأستاذ الدكتور/ سيف الدين فطين
أستاذ بقسم الهندسة البيئية
كلية الهندسة، جامعة زويل

كلية الهندسة
جامعة القاهرة
2019