BIOCHEMICAL MODELING OF LOGIC GATES IN PROKARYOTIC TRANSCRIPTIONAL REGULATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

İREM AKSOY CANKUR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
BIOINFORMATICS

AUGUST 2022

BIOCHEMICAL MODELING OF LOGIC GATES IN PROKARYOTIC TRANSCRIPTIONAL REGULATION

submitted by **İREM AKSOY CANKUR** in partial fulfillment of the requirements for the degree of **Master of Science in Health Informatics Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin Dean, **Graduate School of Informatics**	________________
Assoc. Prof. Dr. Yeşim Aydın Son Head of Department, **Health Informatics**	________________
Assist. Prof. Dr. Aybar Can Acar Supervisor, **Health Informatics, METU**	________________

Examining Committee Members:

Assoc. Prof. Dr. Yeşim Aydın Son Health Informatics, METU	________________
Assist. Prof. Dr. Aybar Can Acar Health Informatics, METU	________________
Assoc. Prof. Dr. Tunca Doğan Computer Engineering, Hacettepe University	________________

Date: 31.08.2022

ABSTRACT

BIOCHEMICAL MODELING OF LOGIC GATES IN PROKARYOTIC TRANSCRIPTIONAL REGULATION

Cankur, İrem Aksoy
M.S., Department of Health Informatics
Supervisor: Assist. Prof. Dr. Aybar Can Acar

August 2022, 79 pages

Logic gates are seen in cells in their daily routines and yet, constructing a synthetic one is still a challenge. In this thesis, AND, OR and XOR logic gates in a bacteria cell are modelled using Python as a simulation tool. Furthermore, function fitting is sought to decrease the complexity of gate functions and increase efficiency. Finally, how protein production would be if an average E.coli bacteria had governed these gates is simulated. The major outcome of this thesis is that it is possible to create logic gate simulations of a bacteria cell that gives exact characteristics of a real life case. Also, it is found out that the characteristics of the gates are prone to changes based on transcription factor concentrations, dissociation constants and cooperativity term parameter selections. However, it must be noted that this thesis is completely performed on a computer based environment. In vitro studies are excluded from the scope of the thesis.

Keywords: Logic gates, bacteria, Python

ÖZ

MANTIK KAPILARININ PROKARYOTİK TRANSKRİPSİYONEL DÜZENLEMEDE BİYOKİMYASAL MODELLENMESİ

Cankur, İrem Aksoy

Yüksek Lisans, Sağlık Bilişimi Bölümü

Tez Yöneticisi: Dr. Öğr. Üyesi. Aybar Can Acar

Ağustos 2022, 79 sayfa

Mantık kapıları hücrelerin günlük rutinlerinde görülür ama yine de yapay bir mantık kapısı üretmek hala oldukça zordur. Bu tezde, bir bakterideki AND, OR ve XOR mantık kapıları, simülasyon aracı olarak Python kullanılarak modellenmiştir. Ayrıca, kapı fonksiyonlarının karmaşıklığını azaltmak ve verimliliğini arttırmak için fonksiyon uydurma aranmıştır. Son olarak, ortalama bir E.coli bakterisi bu kapılara sahip olsaydı, protein üretiminin nasıl olacağı simüle edilmiştir. Bu tezin en önemli bulgusu, bir bakteri hücresinde gerçek hayattaki mantık kapılarının aynı özelliklerini gösteren bir mantık kapısı simülasyonu yaratmanın mümkün olduğudur. Dahası, transkripsiyon faktörü konsantrasyonları, ayrışma sabitleri ve işbirliği terimi parametre seçimlerine bağlı olarak kapıların özellik değiştirmeye eğilimli olduğu tespit edilmiştir. Lakin bu tezin tamamen bilgisayar ortamında gerçekleştirildiği belirtilmelidir. Laboratuvar ortamı çalışmaları, tezin kapsamı dışındadır.

Anahtar Kelimeler: Mantık kapıları, bakteri, Python

To my family and friends

ACKNOWLEDGMENTS

I would like to start with expressing my sincere gratitude to my supervisor Assist. Prof. Dr. Aybar Can Acar for both accepting me as his student and also for his advice, guidance, and reccomendations throughout the study.

Also, I would like to thank the rest of my thesis committee: Assoc. Prof. Dr. Yeşim Aydın Son and Assoc. Prof. Dr. Tunca Doğan for their considerate comments and contributions.

Furthermore, I would like to thank my friends and colleagues for all the motivation and support throughout the study.

And foremost, I would like to express my hearty gratitude and respect to my parents Rıdvan and Fatma Aksoy, to my brother Burak Aksoy and to my partner Anıl Cankur for providing me with the support and continuous encouragement throughout my study and through the process of this research and writing. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DNA	Deoxyribonucleic Acid
RNA	Ribonucleic Acid
mRNA	Messenger Ribonucleic Acid
rRNA	Ribosomal Ribonucleic Acid
tRNA	Transfer Ribonucleic Acid
RNAp	Ribonucleic Acid Polymerase
TF	Transcription Factor
K	Effective Dissociation Constant

CHAPTER 1

INTRODUCTION

Logical gates are used to animate the ideas of human intelligence in electrical devices. The same pattern should be applicable in biological environment as well. There are endless areas where biological gates could be used from detection of diseases to cure for environmental disasters.

Taking bacteria as an example, researches show that there are various areas that programmed bacteria could be used. It can be used to produce graphene which is a strong, flexible and conductive nanomaterial composed of a single layer of carbon atoms. It also can be used to break down the oil spills and prevent environmental damage. Moreover, bacteria can be used to detect arsenic contamination in water which is a worldwide health risk to millions of people since it is causing cancer and death. Furthermore, it can be used for detecting tumours and can be used for cutting down the plastic waste [3].

Logic gates in cells is not an operation that is unheard or never done before. Cells create biological gates in their daily operations all the time. For example, white blood cells detects the presence of infection and fights with it. Cell membranes let some of the particles into the cells, while they refuse to let some of the particles in. However, creating biological gates synthetically is still a challenge.

What makes an eye cell different from a blood cell is due to gene regulation in a cell. Minor changes in the gene regulation may be the underlying reasons of diseases, may yield to the population differences or the morphological variations of species. Therefore, modeling and controlling gene regulation is of great importance [4]. At the same time, the successful modeling of gene regulation may help to discover the function of an uncharacterised piece of DNA [5]. Thus it is safe to say that gene regulation is the key element in synthetic biology.

It is possible to turn a gene on or off by manipulating the environmental signals or changing the binding factors and altering the cooperativity. With the help of this information, it is possible to create logic gates in cells.

1.1 Aim Of The Thesis

The primary aim of this thesis is to formulate and simulate AND, OR and XOR gate in E.coli using Python as a simulation tool. The secondary aim is to search for a simpler formula to replace the complex one. Finally, the third aim is to simulate protein generation of aforementioned gates.

Thermodynamic equations are used to formulate the dynamic changes in cells. This attribute makes them vital for formulating the logical gates in cells. Taking this as a starting point, thermodynamic equations are used to formulate logic gates in cells in this thesis.

Once logical gates are settled, it is possible to search a function fitting. Since simpler function is more desirable in terms of efficiency if it can provide sufficient accuracy. Therefore, as a second step, function fitting is searched in this thesis.

When a gene y of a cell is "ON", it starts to produce protein Y. If it is possible to make a gene y ON by implementing AND, OR and XOR gates, it should also be possible to observe protein generation. Therefore, as a final step in this thesis, expected protein generation of these gates are modelled.

Simulating gene logic gates and protein generation reveals the possibility of creating mini-computers based on controlled bacteria activity. This activity can be replicated by using simpler functions to achieve fast and accurate results that increase the efficiency, therefore, decreasing the need of resources to create a test environment and yielding opportunity to create and test more hypothesis. The outcome of this study can enable advancement in genetic engineering in terms of creating genetic circuits and designing algorithms in a reliable programmable way in order to support production of personalized medicine, biofuels, biomaterials, etc.

CHAPTER 2

BACKGROUND

In this chapter, biological and mathematical background behind the related studies are given in detail.

2.1 Biological Background

DNA, RNA and protein are the three major biological macromolecules that are essential for all known forms of life. The flow of genetic information in a cell is from DNA through RNA to proteins. DNA makes RNA makes protein, which is a central tenet of molecular biology. DNA carries the required genetic information for the cell to grow, to take in nutrients, and to propagate. Proteins are the key workers of the cell as they take various roles ranging from structural to signaling. RNA is the photocopy of the DNA and it comes into picture when the cell needs to produce a certain protein [6].

2.1.1 What Is Amino Acid?

Amino acids are organic compounds that consist of amino and carboxylic acid, -NH2 and -COOH respectively [7]. There are 20 common amino acids found in biological chemistry [8]. Amino acids are one of the key elements for body homeostasis as they take part in various functions. They are the pioneers of the hormone synthesis and low-molecular weight nitrogenous substances. They are responsible from cell signaling. They take part in gene expression and the protein phosphorylation cascade. Also, recently there is an acceptance that some amino acids are necessary for maintenance, growth, reproduction, and immunity as they regulate key metabolic pathways [9].

2.1.2 What Is Protein?

Proteins are large biopolymeric structures composed of one or more long amino acid chains. Their sequence is determined by the genetic code. Proteins fold up to take various parts in cells which is crucial to cell life. They serve as structural support (cytoskeleton), mechanical support (muscle), biochemical catalysts (enzymes). They take part in cell signaling (hormones), building blocks, and they are initiators of cellular death.

Proteins can be further defined by their four structural levels of primary, secondary, tertiary, and quaternary [8]. Four structures of Human PCNA protein is given in Figure 1.

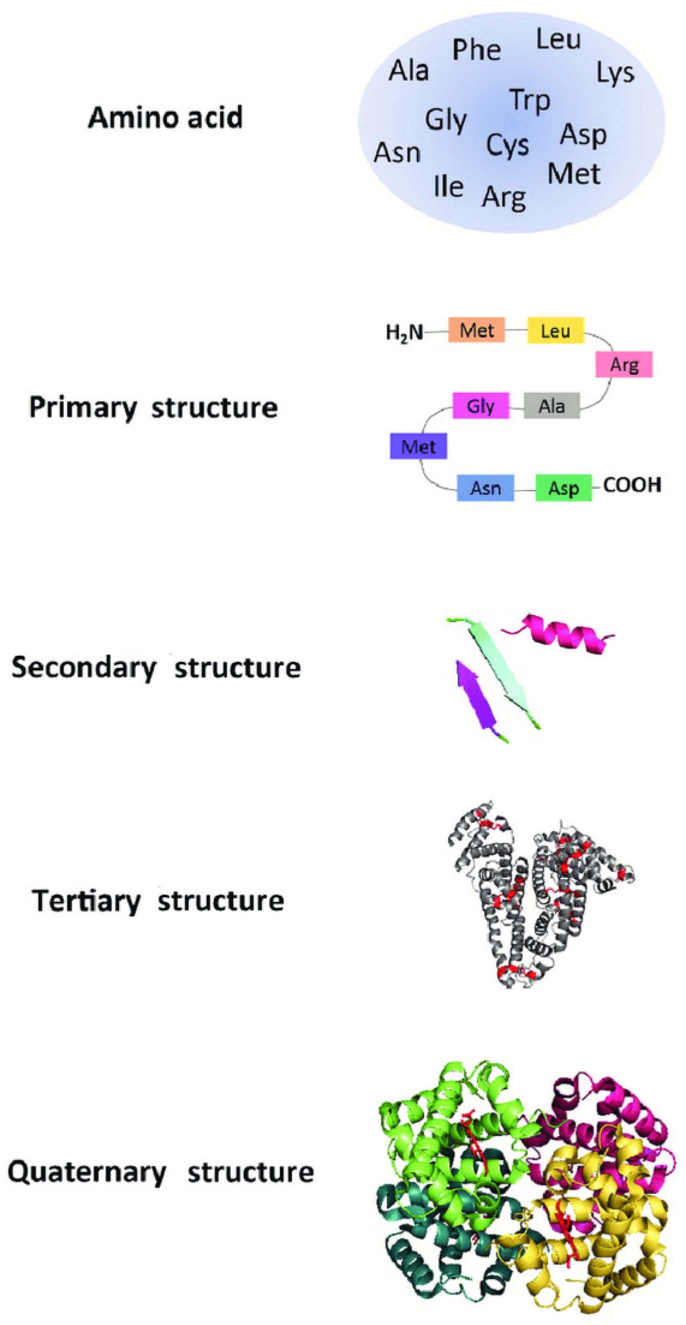

Figure 1: Protein Structure (adapted from [1])

The primary structure is the protein backbone. It is basically the linear order of amino acids that make the protein. Amino acids connect via peptide bonds. Based on these bonds, carbon atoms position is determined as alpha, beta or gamma position [8].

The secondary structure is the 3D form of local parts of protein. Segments of protein comes together via hydrogen bonding and construct shapes such as alpha helix, beta-pleated sheet, and beta-turn. Hydrogen bonds stabilize all of these shapes [8].

The tertiary structure is the real 3D model of a protein. It is constructed when the synthesized polypeptide chain exits the terminal end of the ribosomal subunit complex and enters the aqueous environment. The aim is achieving the thermodynamic stability which is further driven by a variety of chemical interactions. The tertiary structure of a protein is dynamic [8].

The quaternary structure of protein, which is the final level, is the 3D form of protein when multiple polypeptides in it folds independently from each other. These forms are called subunits which are interchangeable with promoter [8].

The quaternary structure is the reason behind two important phenomenons called allostery and coopeativity. Allostery is the activity and structure of a protein changing from the binding of a molecule to a side besides the active site, and cooperativity is the increases or the decreases in the likelihood of a binding of a molecule at one side of the protein due to the binding of another molecule at another side of the same protein [10]. Hemoglobin is the most known example of this situation. Hemoglobin consists of two pairs of different proteins. These subunits work dependently. When one subunit binds, conformation of the protein changes. This change is called an allosteric effect. When one subunit binds, it is easier for successive oxygen molecules to bind at the remaining three sites. The increase in the likelihood of binding is called cooperativity [11].

2.1.3 What Is Rna?

RNA is ribonucleic acid. There are different types of RNAs in cells. Messenger RNA (mRNA) carries the genetic sequence of a gene, ribosomal RNA (rRNA) is a part of ribosome and takes part in protein synthesis, transfer RNA (tRNA) physically carries amino acids to the translation site. Besides these there are some small RNAs involved in regulating gene expression [12]. For the majority of cells DNA carries the genetic information; however, in some viruses RNA carries the genetic information.

In order to produce a certain protein, cell activates the portion of DNA that codes for that protein, gene, and produces multiple copies of that gene in the form of messenger RNA. Not only RNA carries the required information for a protein to be generated, it also regulates the amount of protein to be made as the multiple copies of mRNA are used to translate the genetic code into protein at the ribosomes [6].

Up until recently, it is realized that RNA has broader roles besides working as a DNA photocopy (mRNA), as a coupler between the genetic code and the protein building blocks (tRNA), and as a structural component of ribosomes (rRNA). It is discovered that RNA is working as enzymes (called ribozymes) to speed chemical reactions. Furthermore, RNA takes part in regulating cellular processes–from cell division, differentiation and growth to cell aging and death. Some of the important human diseases, including heart disease, some cancers, stroke and many others are linked to the defects in certain RNAs or the regulation of RNAs [6].

2.1.4 What Is Dna?

DNA is deoxyribonucleic acid. It is the blueprint of life. It is what makes each species unique. DNA has a double helix form which consists of two-stranded chemical structure. Each strand is actually a chain of nucleotides linked with the phosphate and sugar groups alternating. These nucleotides are made of three parts: a phosphate group, a sugar group and one of four types of nitrogen bases [13].

DNA is called the blueprint of life because it carries the genetic information an organism needs to survive, develop and breed. In order to perform these functions, DNA sequences are deciphered into a form of messenger RNA. mRNA carries the genetic information to ribosomes to produce proteins which do most of the work in our bodies [13].

A DNA sequence responsible from protein generation is called gene. Only about 1 percent of the DNA is made of genes. Remaining 99 percent of DNA takes part in regulating protein generation in how much, when and how is made [13].

2.1.5 What Is Gene?

Gene is a fragment of DNA that contains the necessary code for generating a specific protein or RNA molecule. It is passed from a generation to a generation. It is the unit of heredity. A gene can produce one or multiple proteins. Sometimes it does not even produce proteins. In that times, it generates RNA molecules to have have other functional roles in cells [14].

In humans, the size of the gene varies immensely. It ranges from almost 1,000 bases to 1 million bases which makes only about 1 percent of DNA [13].

The Human Genome Project, an international research effort to map and understand all of the genes of human bodies, revealed that there are around 20500 genes in human body [15].

Each individual has two copies of each gene, one coming from mother and the other coming from father. More than 99 percent of the genes are identical and less than 1 percent is slightly different. This slight difference is what makes each individual unique [16].

2.1.6 What Is Gene Regulation?

In the human genome, there is around 20,500 genes; however, not all genes are active in every cell. Actually, the number of active genes differs from cell to cell and it is the key element that determines the role of a cell in a body. The turning on and off a gene is called gene regulation. Different gene regulation yields to different gene expression and results in a cell to become an eye cell, or a blood cell or a muscle cell [17]. Cells can drive conclusions from the environmental signals such as the availability of food or the awareness of dangers due to gene regulation. Gene regulation is vital for cells, the failure of it can lead to severe developmental abnormalities or diseases [18].

2.1.7 How Is Gene Regulation Done

Regulation of gene is done by regulatory molecules called activators, repressors and inducers.

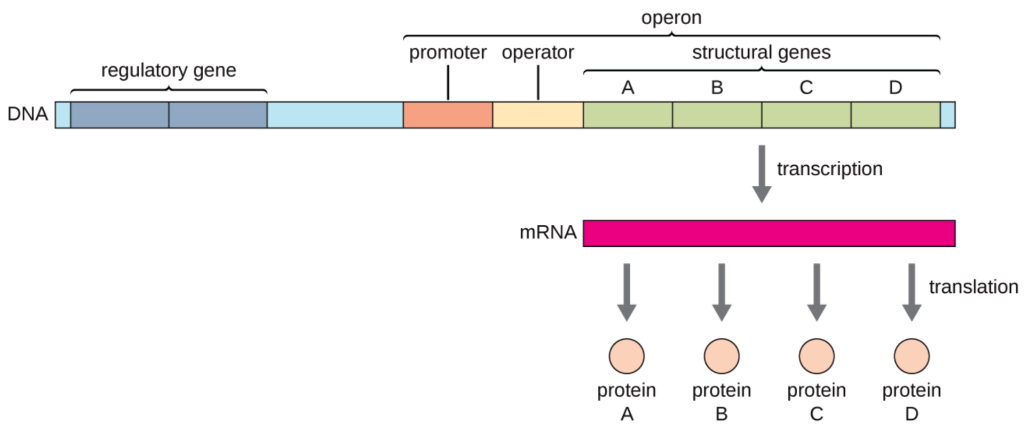

Figure 2: Gene Regulation in Prokaryotes [2]

As seen in Figure 2, in prokaryotes, structural genes are usually located near each other in a block called operon and a single promoter controls the transcription of them together. Each operon has a regulatory region that regulates its own transcription. Regulatory region consists of promoter and the region surrounding the promoter where proteins encoded by regulatory genes called transcription factors can bind [2]. The activity of transcription factors determines the regulation of a gene [18].

Transcription factors can change the binding of RNA polymerase to promoter by increasing or decreasing it. An activator is a transcription factor that increases the transcription of gene by facilitating the RNA polymerase binding to promoter. A repressor is a transcription factor decreases the transcription of gene by blocking the RNA polymerase binding to promoter. Whereas an inducer interacts with transcription factor and either activates or represses gene transcription [2].

2.2 Mathematical Background

2.2.1 Boole's Logic

In 1847, George Boole, mathematician and philosopher, published the first exposition of Boolean Algebra called "The Mathematical Analysis of Logic". He published the longer version "An Investigation of the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities" seven years later in 1854. In his work he explains the logic in an algebraic form for the first time in the history [19]. George Boole was the first person to intertwine mathematics and logic [20].

Besides logic there are two main fields that Boolean algebra is being used. The first one is the theory of probability since Boolean algebra uses intersection and union of sets to treat the combination of sets. It also deals with the individual elements of the sets which makes it even more convenient to use for the theory of probability. The other area that Boolean algebra is being used is the electronics. Claude E. Shannon introduced Boolean algebra to electronics by showing that Boolean algebra could be used to represent basic properties of series and parallel combinations of bistable electrical devices. After that, Boolean algebra has become important and crucial part of designing digital circuits such as computers, switching circuits and automatic control devices [21].

Boolean algebra uses binary numbers which are 1 and 0. Binary number 1 represents “true” and binary number 0 represent “false”. What makes boolean algebra different from the elementary algebra is that boolean algebra operates on logical operations [22]. There are five basic logical operations: conjunction, disjunction, negation, implication and bi-implication. These operations, their symbols, how to use them, and their meanings are given in Table 1.

Table 1: Basic Logic Operations

Operation	Symbol	Usage	Translation
Conjunction	$\wedge$	$A \wedge B$	A and B
Disjunction	$\vee$	$A \vee B$	A or B
Negation	$\neg$	$\neg A$	not A
Implication	$\rightarrow$	$A \rightarrow B$	if A then B
Bi-implication	$\leftrightarrow$	$A \leftrightarrow B$	A if and only if B

2.2.2 Truth values and truth table

In logic, declarative sentence is valued on only one property: true (t) or false (f). A declarative sentence is either true or false but not both. The truth value of a compound sentence can be build from the founding declarative sentences and their truth values using logical operations with the help of truth tables [20].

Truth table shows the truth value of a compound sentence for all possible combinations of founding sentences. It has 2^n rows where n is the number of founding sentences.

In Boolean algebra true and false values are replaced with binary numbers. Binary numbers 0 and 1 are corresponding to false and true, respectively.

A truth table for a conjunction operation and its representation in Boolean algebra is given in Table 2.

Table 2: Conjunction Operation

Logical Represention			Boolean Represention		
A	B	$A \vee B$	A	B	$A \vee B$
T	T	T	1	1	1
T	F	T	1	0	1
F	T	T	0	1	1
F	F	F	0	0	0

A truth table for a disjunction operation and its representation in Boolean algebra is given in Table 3.

A truth table for a negation operation and its representation in Boolean algebra is given in Table 4.

Table 3: Disjunction Operation

Logical Represention			Boolean Represention		
A	B	$A \wedge B$	A	B	$A \wedge B$
T	T	T	1	1	1
T	F	F	1	0	0
F	T	F	0	1	0
F	F	F	0	0	0

Table 4: Negation Operation

Logical Represention		Boolean Represention	
A	$\neg A$	A	$\neg A$
T	F	1	0

A truth table for a implication operation and its representation in Boolean algebra is given in Table 5.

Table 5: Implication Operation

Logical Represention			Boolean Represention		
A	B	$A \rightarrow B$	A	B	$A \rightarrow B$
T	T	T	1	1	1
T	F	F	1	0	0
F	T	T	0	1	1
F	F	T	0	0	1

A truth table for a bi-implication operation; otherwise known as XNOR gate, and its representation in Boolean algebra is given in Table 6.

Table 6: Bi-Implication Operation

Logical Represention			Boolean Represention		
A	B	$A \leftrightarrow B$	A	B	$A \leftrightarrow B$
T	T	T	1	1	1
T	F	F	1	0	0
F	T	F	0	1	0
F	F	T	0	0	1

2.2.3 Basic Logic Gates

Logic gate is a device that implements Boolean algebra to produce a single output from either a single input or double inputs. There are 7 basic logic gates: AND gate, OR gate, NOT gate, NAND gate, NOR gate, XOR gate and XNOR gate. Among these gates only NOT gate performs on a single input, the rest of them performs on double inputs [23].

AND gate corresponds to conjunction, OR gate corresponds to disjunction, NOT gate corresponds to negation and XNOR gate corresponds to bi-implication operation in logical operations. Combined truth table for 6 basic logic gates is given in Table 7 and the truth table for NOT gate is given in Table 8.

Table 7: Basic Logic Gates

A	B	A AND B	A OR B	A NAND B	A NOR B	A XOR B	A XNOR B
1	1	1	1	0	0	0	1
1	0	0	1	1	0	1	0
0	1	0	1	1	0	1	0
0	0	0	0	1	1	0	1

Table 8: NOT Gate

A	NOT A
0	1
1	0

CHAPTER 3

LITERATURE REVIEW

Minor changes in gene regulation may be the underlying reason of diseases, may yield to the population differences or the morphological variations of species. Therefore, modeling and controlling gene regulation is of great importance [4]. The successful modeling of regulation may help to discover the function of an uncharacterized piece of DNA [5].

Modeling and simulating gene regulation in bacteria is searched by many researchers. Anderson [24] has generated a synthetic AND gate in a E.coli bacteria that integrates information from two promoters as inputs via an interaction between a mRNA and tRNA. The study mentions that synthetic AND gate activates a promoter output only when both input promoters are transcriptionally active. Wang [25] also constructed a modular orthogonal AND gate in E. coli bacteria. AND gate in the study consists of two co-activating genes hrpR and hrpS controlled by different promoter inputs, and a σ54-dependent hrpL promoter that directs the output. Furthermore, it is revealed that a NOT gate module could be added to an AND gate module to create a NAND gate. Also, NTNU[26] has created a XOR logic gate in E.Coli based on retrons producing msDNAs that separates RNA and DNA on the iGEM Foundation, which is an independent, non-profit organization whose purpose is the advancements of synthetic biology[27].

Generally, there are two approaches that could be employed to model gene regulation of eukaryotic cells; statistical or analytical.

Statistical approaches are used for recognizing regulatory networks under the governing dataset since they give insight about possible regulatory relations by showing which gene groups function together. Neural, Boolean, and Bayesian networks are the most common applied graph-based statistical approaches. They are typically used on transcriptome datasets that govern the expression levels of thousands of genes. Even if the statistical approaches draw the big picture of gene regulation in organisms, they fail to give insight on the underlying mechanism of gene regulation.

Analytical approaches are used for understanding the governing mechanisms behind gene regulation. However, they require extensive knowledge of the system. The binding of transcriptional activator proteins and RNA polymerase to DNA, cooperativity, activators and inhibitors, mRNA and proteins deflation, and mRNA translation rate are taken into account in analytical models. Therefore, they are used for small number of genes. Thermodynamic, Boolean, and differential equation-based models are the three major used analytical approaches. Analytical approaches are deterministic. The results due to a change in an individual parameter could be modeled in analytical approaches. Therefore, they yield to testable hypothesis for researchers [28].

In this thesis, the thermodynamic theory for a bacteria (procaryote cells) that Shea and Ackers have explained is used by implementing the formulas laid out by Buchler et al. [29][30].

The system behind thermodynamic modeling is as follows: rather than computing the level of gene expression by means of the concentration of gene product (protein), thermodynamic modeling concentrates on the probability of a RNA polymerase (RNAp) binding to a specific promoter. The main assumption in this approach is that probability of a specific promoter binding to RNA polymerase is proportional to the given gene expression level. The probability of a protein's (TF or RNAp) binding to a DNA depends on the concentration of the protein, binding affinity, and the interactions with other proteins (TF or RNAp). From the set of all possible occupancy states of the promoters and their interactions with transcription factors in equilibrium conditions, the probability of RNAp-promoter binding is calculated [31][29][32][5].

According to Buchler et al. [29] the transcription in bacteria could be modeled via the pairwise thermodynamics of protein-DNA and protein-protein interactions. The probability (p) of a transcription factor-DNA and RNAp-promoter binding (in the absence of any TFs) for a site i is as follows:

$$p_i = \frac{q_i}{1 + q_i} \tag{1}$$

q_i is called the binding affinity of a site i and it's formulation is as follows;

$$q_i = \frac{[TF_i]}{K_i} \tag{2}$$

where $[TF_i]$ is the transcription factor's molar concentration and K_i is the effective dissociation factor belonged to a specific site, i. The dissociation constant, K, is tunable across a wide range of TF concentration.

Cooperativity term $\omega_{i,j}$ is used to model the interactions between a pair of proteins (TF or RNAp) bound to sites i, and j. At Table 9 the values cooperativiy term can get and their meanings are explained.

Table 9: Cooperativity Term

$\omega_{i,j} = 0$	i, j binding sites are completely overlapping. Mutual exclusion.
$\omega_{i,j} < 1$	i, j binding sites are partially overlapping.
$\omega_{i,j} = 1$	i, j sites are no longer interacting.
$\omega_{i,j} > 1$	If a contact is possible when a pairwise proteins are bound to their sites, cooperativity may be obtained.

Under the assumption that the TF-TF interaction is glue-like and there are L TF-binding sites, the Boltzmann weight for each configuration of site occupation is as follows:

$$W[\sigma_1, ..., \sigma_L] = \prod_{i=1}^{L} q_i^{\sigma_i} \prod_{i<j} \omega_{i,j}^{\sigma_i \sigma_j}, \sigma_i \in \{0, 1\}, \tag{3}$$

where a binary variable $\sigma_i \in \{0, 1\}$ represents whether a site is occupied or not. Additionally, the Boltzmann weight due to the interaction of the RNAP with the bound TFs is as follows:

$$Q = q_p \prod_{i=1}^{L} [1 - \sigma_i \delta(\omega_{0,i}, 0)].[1 + \omega \sum_{j=1}^{L} \sigma_j \delta(\omega_{p,j}, \omega)] \tag{4}$$

The equilibrium probability, P, of the binding between RNAP and promoter resulting from their interactions with bound TFs could be used to quantify the gene transcription degree. This probability is formulated as:

$$P = \frac{Z_{ON}}{Z_{ON} + Z_{OFF}}, \tag{5}$$

where Z_{ON} represents the complete set of TF binding states for the promoter that is bound by RNAP. It can be written as the partition sum of the Boltzmann weights

$$Z_{ON} = \sum_{\sigma_1 = \{0,1\}} ... \sum_{\sigma_L = \{0,1\}} Q[\sigma_1, ..., \sigma_L].W[\sigma_1, ..., \sigma_L], \tag{6}$$

and Z_{OFF} represents the complete set of TF binding states for the promoter that is unbound by RNAP. It can be written as the partition sum of the Boltzmann weights

$$Z_{OFF} = \sum_{\sigma_1 = \{0,1\}} ... \sum_{\sigma_L = \{0,1\}} W[\sigma_1, ..., \sigma_L]. \tag{7}$$

By using this formulations, logic gates are going to be implemented. Promoter occupancy of larger than 40% is taken as sufficient for a gene to be "ON" for all the logic gates implemented. Even though it is stated that 40% is an arbitrarily chosen number, it is a plausible number since switching from ON to OFF occurs in a sigmoidal manner and after 0.5, sigmoidal function is completely in ON state.

As it is explained by Uri Alon [33], a cell starts to produce a protein y, whenever a gene Y is activated. Protein concentration in the cell depends on the rate of protein production, protein degradation and protein dilution. Protein production rate is represented with β (units of protein y concentration per time). The destruction of a protein by other specialized proteins is called protein degradation rate and is represented with α_{deg}. The decrease of concentration of protein y resulting from the increase of volume due to growth is called protein dilution rate and is represented by α_{dil}. Both α_{deg} and α_{dil} are the probabilities per unit time that each protein y is removed. Together they form the total removal rate, $\alpha = \alpha_{dil} + \alpha_{deg}$.

The change in the concentration of protein y in a cell can be formulated as:

$$dY/dt = \beta - \alpha Y. \tag{8}$$

Y_{st} represents the steady-state concentration of y. It could be found by solving for $dY/dt = 0$. At Equation 9, the steady-state concentration formula is given.

$$Y_{st} = \frac{\beta}{\alpha} \tag{9}$$

When the cell starts to produce a protein Y with an initial concentration of protein $Y = 0$, the concentration of protein Y converges to Y_{st}, gradually. Protein generation formula of a cell is given at Equation 10

$$Y(t) = Y_{st}(1 - e^{-\alpha t}) \tag{10}$$

Similarly, when the production of protein Y stops, $\beta = 0$, the concentration of protein Y in cell decreases exponentially. At Equation 11, the formula for decay is given.

$$Y(t) = Y_{st}e^{-\alpha t} \tag{11}$$

The response time, τ, is an important measure for understanding how fast the protein Y decays. It is defined as the time the concentration reaches the half level from initial level to final level. When Equation 11 solved for $Y(t) = Y_{st}/2$, τ is found as:

$$\tau = \frac{log(2)}{\alpha} \tag{12}$$

As it could be seen from the Equation 12, the response time depends only on the removal rate, α. It does not depend on the production rate, β. On the other hand, the steady state protein concentration of a cell depends on both production and removal rate.

In this thesis, these formulas are going to be used to see how an average gene from E.coli would produce protein if it is programmed to work as the aforementioned logic gates.

CHAPTER 4

MATERIALS AND METHODS

In this thesis, AND, OR, and XOR logic gates are modeled using Python and SciPy [34] as modeling tools. The logic gates are modeled considering two transcription factors as input signals. TF_A and TF_B represents these transcription factors. The concentrations of these transcription factors are changed in the interval of (1–1,000 nM).

AND and OR gates are modeled using a single promoter. On the other hand, XOR gate is modeled using both a single promoter and a double promoter. For a gate with a single promoter system, q_p represents the affinity of the promoter. For a double promoter system, q_{p_1} and q_{p_2} represent the affinity of the promoters, respectively.

As mentioned in the previous chapter, for all the logic gates implemented, promoter occupancy larger than 40% is taken as sufficient condition for a gene to be considered as "ON" [29]. When a gene is "ON", it starts to produce protein. Proteins are degraded in a cell depending on two factors: protein degration, α_{deg}, and protein dilution, α_{dil}. However, for growing cells that is not the case. In growing cells, many proteins are not actively degraded. These proteins are called stable proteins. For a stable protein, protein degradation is zero, $\alpha_{deg} = 0$, and protein production is balanced by only protein dilution, α_{dil}. This means that a stable cell grows until its volume is doubled. Then it splits into 2 cells which makes the protein concentration halved. For this kind of proteins, the response time is equal to one cell generation time. Therefore, for stable proteins, response time formula given in Equation 12 turns into Equation 13 [33],

$$\tau = \frac{log(2)}{\alpha_{dil}} \tag{13}$$

In this thesis, for the sake of simplicity, a stable protein is assumed to be generated. Typical cell generation time for an E.coli bacteria changes from twenty minutes to several hours [33]. It is assumed as 30 minutes. When Equation 13 is solved for $\tau = 30$, it is found that $\alpha_{dil} = 0.023105$. Also, as it is stated by Uri Alon [33], β is a tunable parameter. So, it is taken as $\beta = 200000$.

4.1 AND GATE

In this section, formulation of AND gate is described. Modeling of AND gate is explained and protein generation is simulated using inputs whose concentrations are in the form of a square wave.

4.1.1 AND Gate Formulation

According to Buchler et al. [29] formulation and parameters for an AND gate are as follows:

$$q_p = 1/35 \tag{14}$$

$$Z_{OFF} = 1 + q_A + q_B + w\, q_A\, q_B \tag{15}$$

$$Z_{ON} = q_p\, (1 + w\, q_A + w\, q_B + 2\, w^2\, q_A\, q_B) \tag{16}$$

$$K_A = K_B = 3500 \tag{17}$$

The heat map of AND gate for $K_A = K_B = 3500$ and $w = 20$ could be seen in Figure 3.

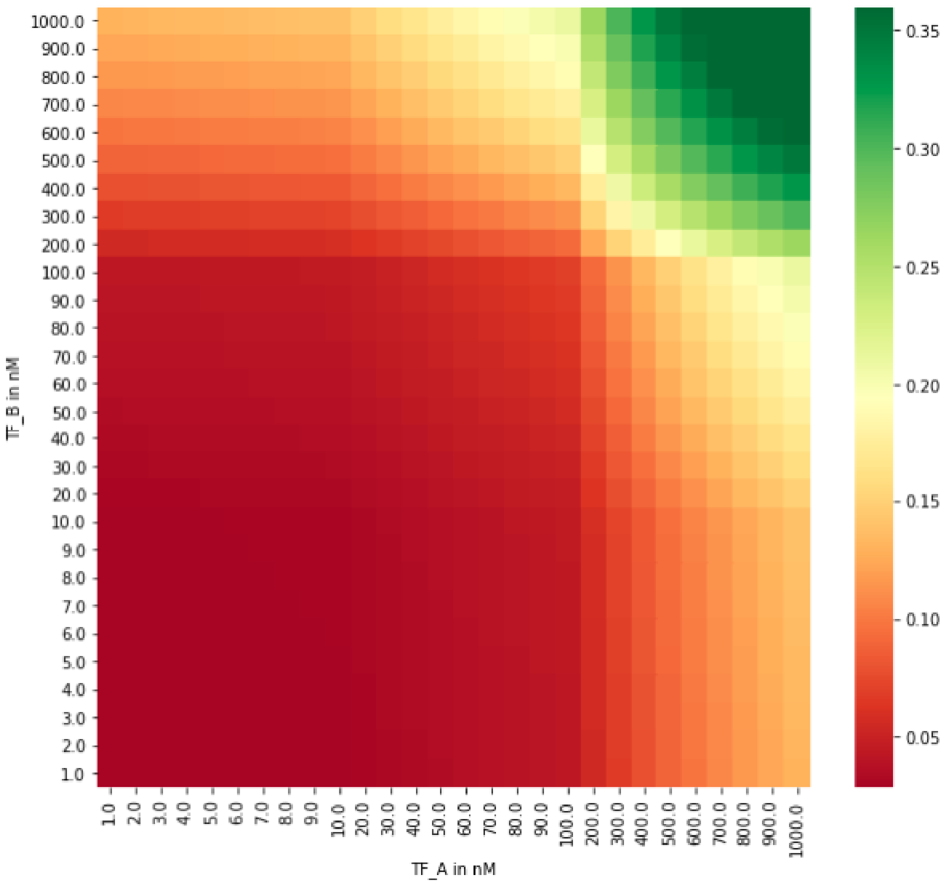

Figure 3: Heat map of AND gate for $K_A = K_B = 3500$, $q_p = 1/35$ and $w = 20$

4.1.2 AND Gate Fitting

In continuation, formulation for AND gate is searched to find if a simpler function could be used in the same molarity concentration interval of input transcription factors.

If the concentration of TF_A is a step function with an amplitude of 1000 nM and the concentration of TF_B is set to 1nM, the probability of gene expression is also a step function that is similar to

$[TF_A]$. Similarly, if the concentration of TF_B is a step function with an amplitude of 1000 nM and the concentration of TF_A is set to 1 nM, the probability of gene expression is also a step function. This similarity could be seen in Figure 4. Therefore, it could be said that equilibrium probability given in Equation5, P, can be fitted to simpler formula that takes $[TF_A]$ and $[TF_B]$ as inputs. The chosen simpler formula as a fit is:

$$\frac{a}{b\,[TF_A] + c} + \frac{d}{e\,[TF_B] + f} \tag{18}$$

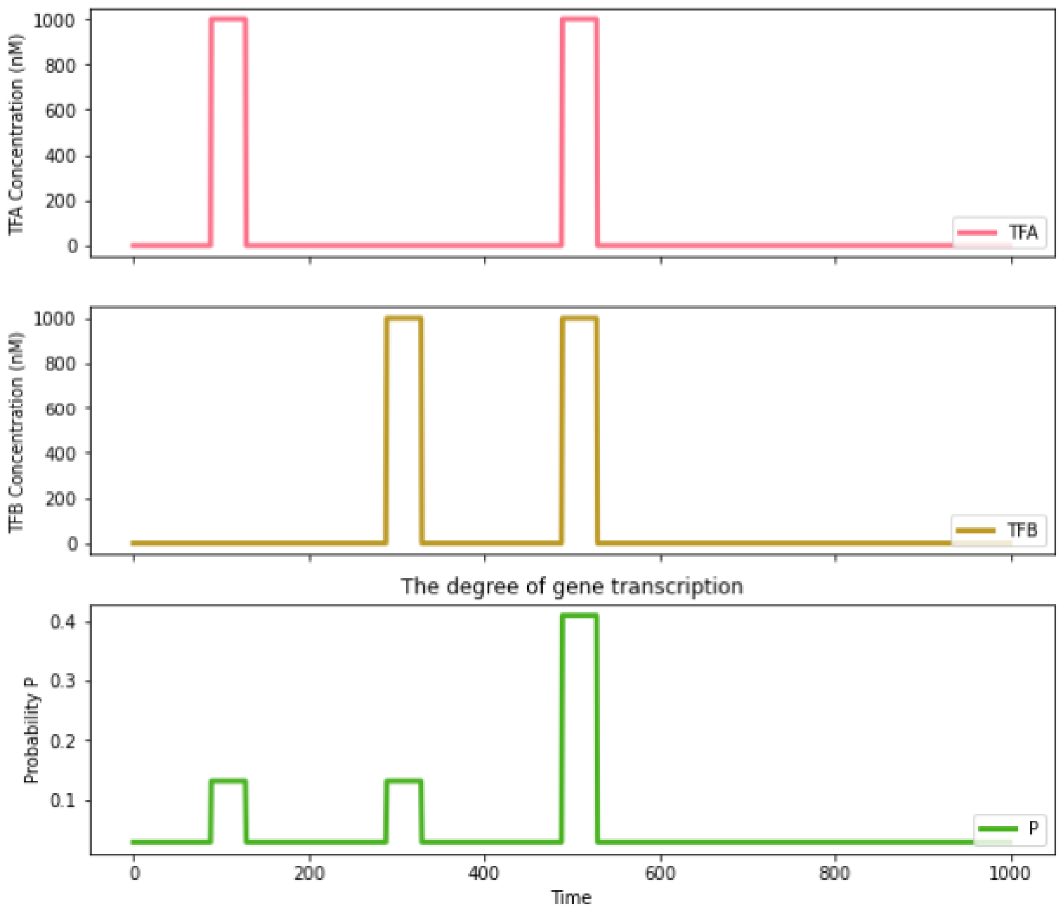

Figure 4: AND gate's probability plot for $K_A = K_B = 3500$, $q_p = 1/35$ and $w = 20$ under changing input states

The coefficients a, b, c, d, e, f are calculated by using function "curve_fit" [35] from SciPy library; because, the "curve_fit" function finds the optimal coefficients that minimizes the sum of the squared residuals of the provided data values and the chosen formula.

To find the optimal coefficients, "curve_fit" function has been run twice. The first run has been run with zero initial coefficients and the second run has been run with the coefficients calculated in the first run.

The calculated coefficients are given in Table 10.

Table 10: AND gate's fit formula coefficients

a	b	c	d	e	f
-8,274	0,020	-43,982	18,162	51,897	-163,481

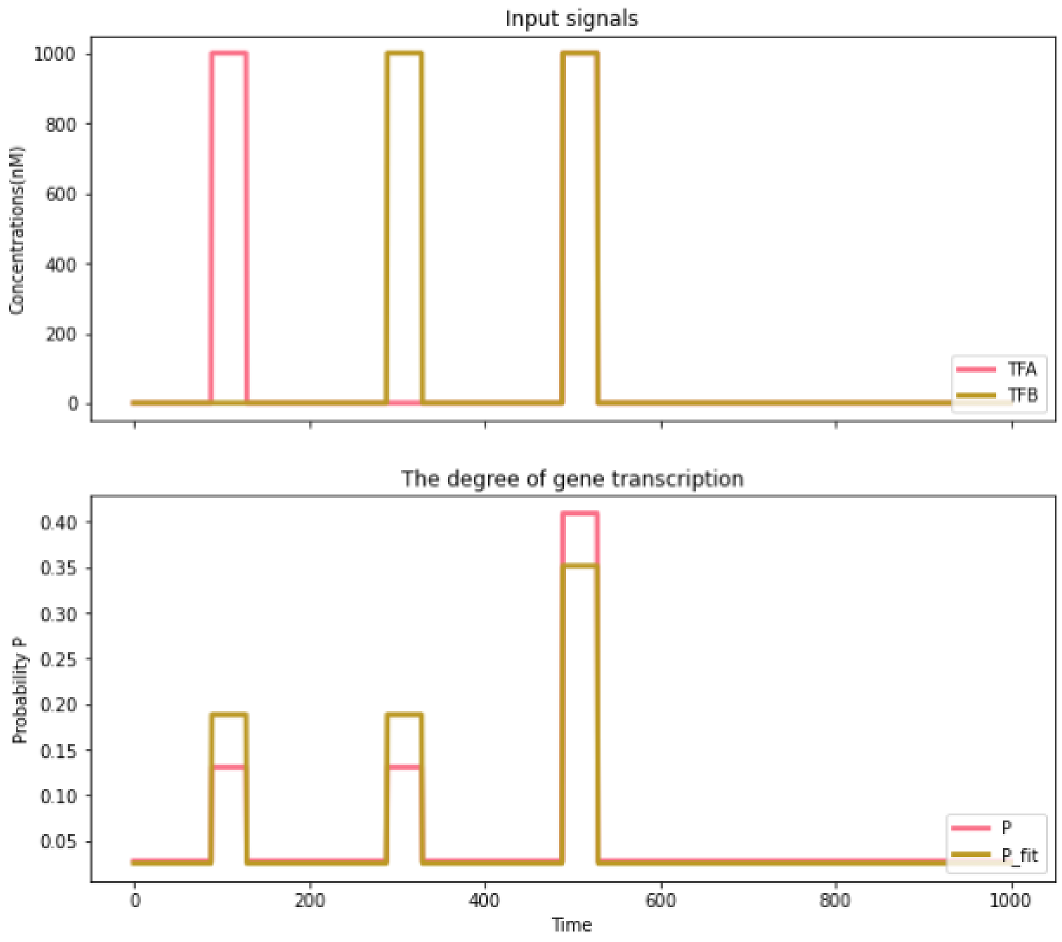

Figure 5: Graph of AND gate's original and fit function equilibrium probability results for $K_A = K_B = 3500$, $q_p = 1/35$ and $w = 20$ under changing input states

In Figure 5, the equilibrium probabilities calculated with the original formula and the fitted formula are given. In order to see how the calculated coefficients perform, R score is calculated. The first R score is calculated by comparing the real probability values and the fitted values. It is found to be 0.934. Afterwards, to test how this coefficents would succeed under different input concentrations, trial test is performed. In Figure 6, in the top plot, original concentrations used to calculate coefficients are given. In the middle plot, the concentrations used for trial case are given. In the bottom plot, the probability of the real probability function and the probability of the fitted function under trial concentrations are given. R score calculated for trial concentration is found as 0.93.

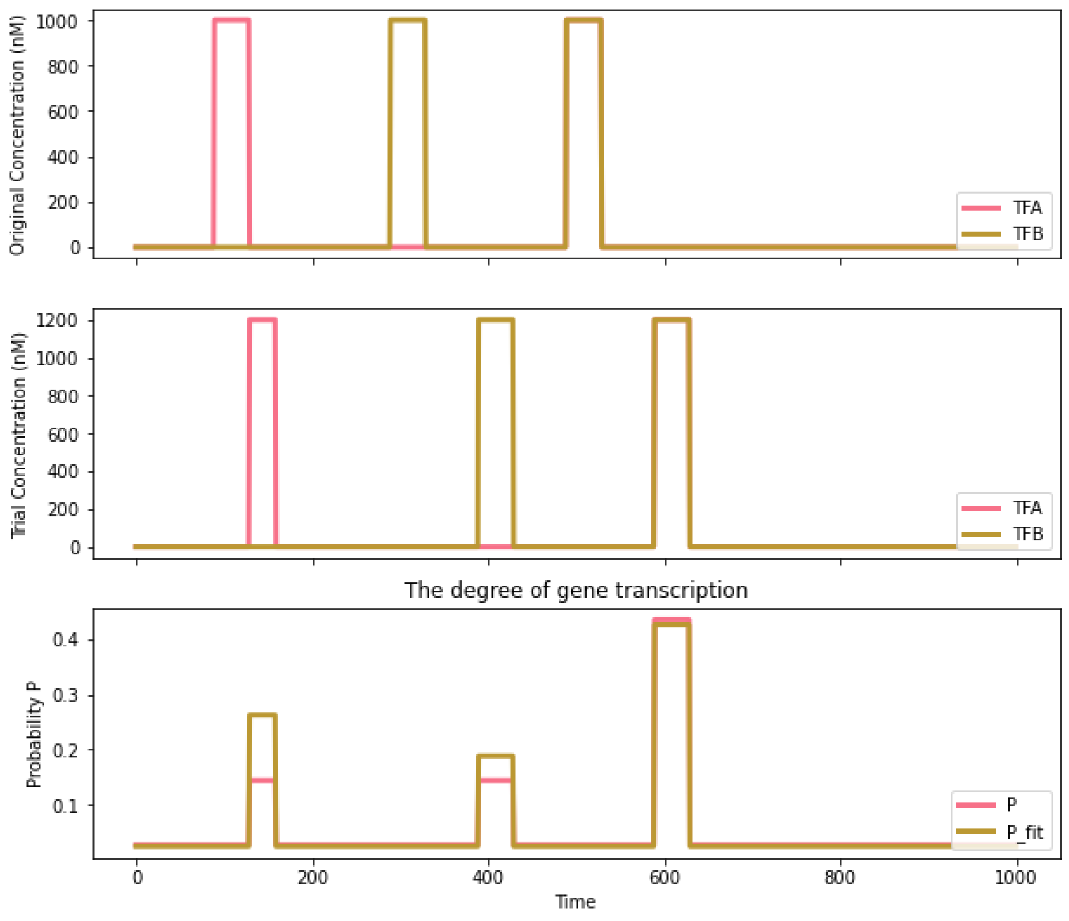

Figure 6: AND gate's original and trial input concentration graphs and P plots for trial concentrations for $K_A = K_B = 3500$, $q_p = 1/35$ and $w = 20$

4.1.3 AND Gate Protein Generation Simulation

As it could be seen in Table 7, there are four states of AND gate. In order to see what would happen to protein generation during these states, a simulation whose steps are the following is performed:

- The duration of the simulation is taken as 350 minutes.
- At the beginning, $[TF_A]$ and $[TF_B]$ are set to 1nM to observe OFF-OFF state.
- To see the characteristics of ON-OFF state, from 5 to 10 minutes, $[TF_A]$ is abruptly increased to 1000 nM then, it is decreased to 1 nM and kept constant for 95 minutes. During this time $[TF_B]$ is kept constant at 1nM.
- To see the characteristics of OFF-ON state, between the interval of 105 to 110 minutes, $[TF_B]$ is abruptly increased to 1000nM. At the 110th minute, $[TF_B]$ is decreased to 1nM and kept constant at 1nM for 95 minutes. During this time $[TF_A]$ is kept constant at 1nM.

- To see the characteristics of ON-ON state, at the 205th minute, $[TF_A]$ and $[TF_B]$ are abruptly increased to 1000nM for 5 minutes. Afterwards, their concentrations are returned back to 1nM and kept constant at 1nM for the rest of the simulation.

This input steps could be seen in the top graph of Figure 7.

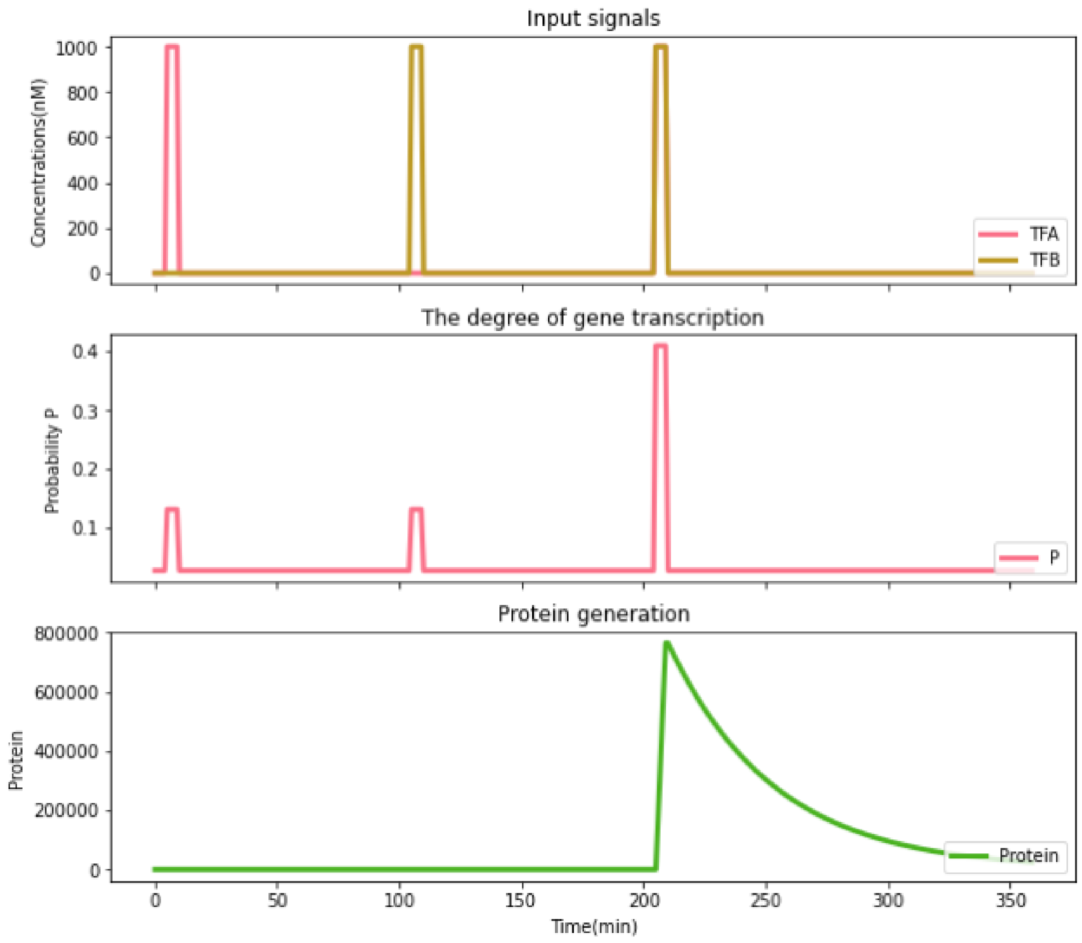

Figure 7: Input concentrations of TF_A and TF_B, P plot and protein generation of AND gate for $K_A = K_B = 3500$, $q_p = 1/35$ and $w = 20$

4.2 OR GATE

In this section, OR gate formulation is explained. Formulation fitting is discovered and protein generation simulation for OR gate is watched.

4.2.1 OR Gate Formulation

According to Buchler et al. [29] formulation and parameters for an OR gate are as follows:

$$q_p = 1/20 \tag{19}$$

$$Z_{OFF} = 1 + q_A + q_B + q_A\, q_B \tag{20}$$

$$Z_{ON} = q_p\,(1 + w\, q_A + w\, q_B + 2\, w\, q_A\, q_B) \tag{21}$$

$$K_A = K_B = 100 \tag{22}$$

The heat map generated by using the formula for OR gate with parameters of $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$ is given in Figure 8.

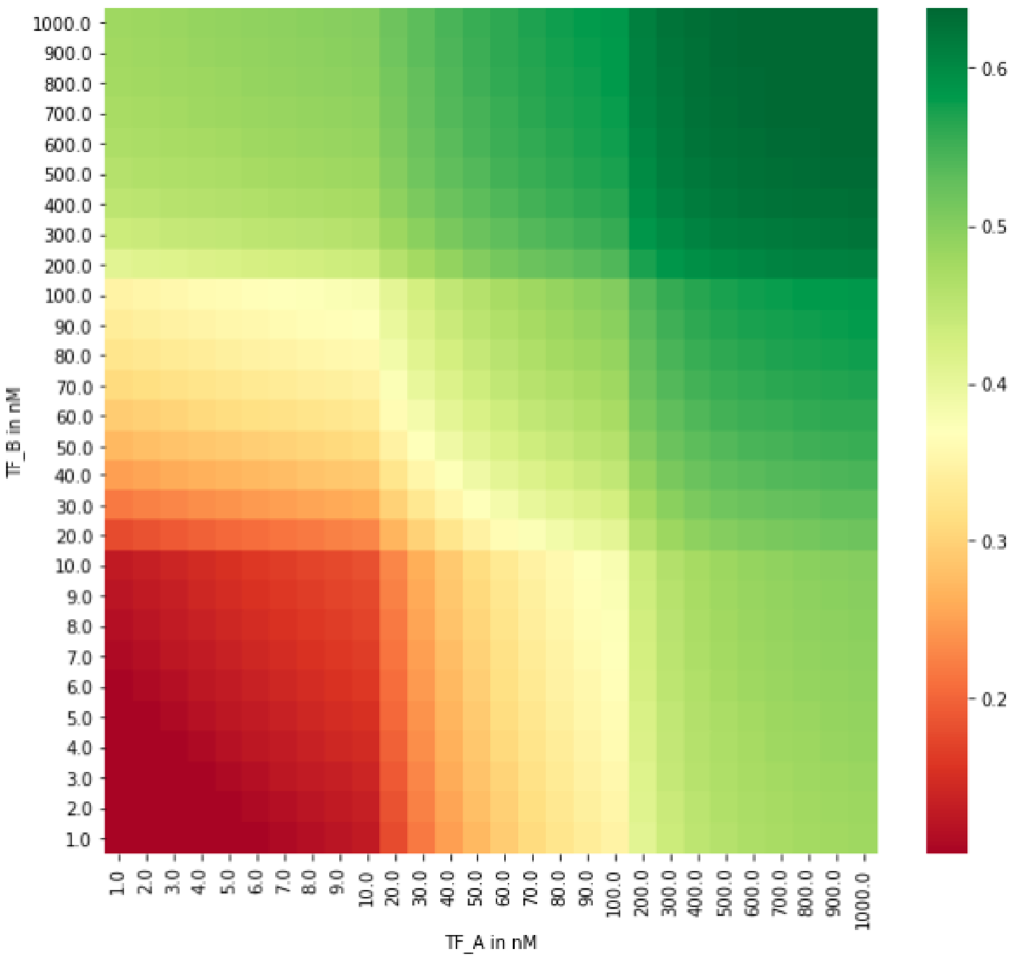

Figure 8: Heat map of OR gate for $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$

4.2.2 OR Gate Fitting

The similar search pattern that is described in Section 4.1.2 is applied to OR gate to see if the formula of OR gate could be fitted to a simpler function, as well. As it could be seen in Figure 9, OR gate has similar characteristics as AND gate.

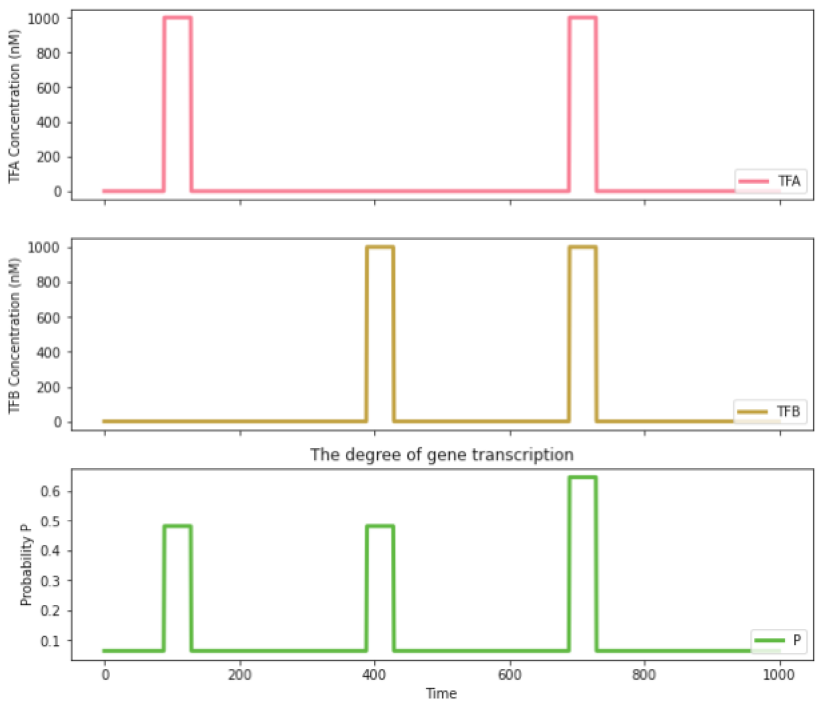

Figure 9: OR gate's probability plot for $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$ under changing input states

Therefore, it could be said that OR gate could be fitted to a simpler formula, if the concentrations of TF_A and TF_B are between [1, 1000] nM. The formula chosen for OR gate is given in Equation 23.

$$a\,[TF_A] + b\,[TF_B] + c\,[TF_A]\,[TF_B] + d \tag{23}$$

The coefficients a, b, c, d are calculated with the same procedure described for Section 4.1.2 by using "curve_fit" [35] function form SciPy library.

The calculated coefficients are given in Table 11.

Table 11: OR gate's fit formula coefficients

a	b	c	d
0,0004164	0,0004164	-0,0000003	0,0635534

In the top plot of Figure 10, the original input concentrations used to calculate the coefficients are given. In the bottom plot of Figure 10, the real probability function and the fitted function results are given. The R score obtained by comparing the real probability values and the fitted values is 1.0. If the coefficients are tested using a trial input concentrations given in Figure 11, R score is calculated as 0.98.

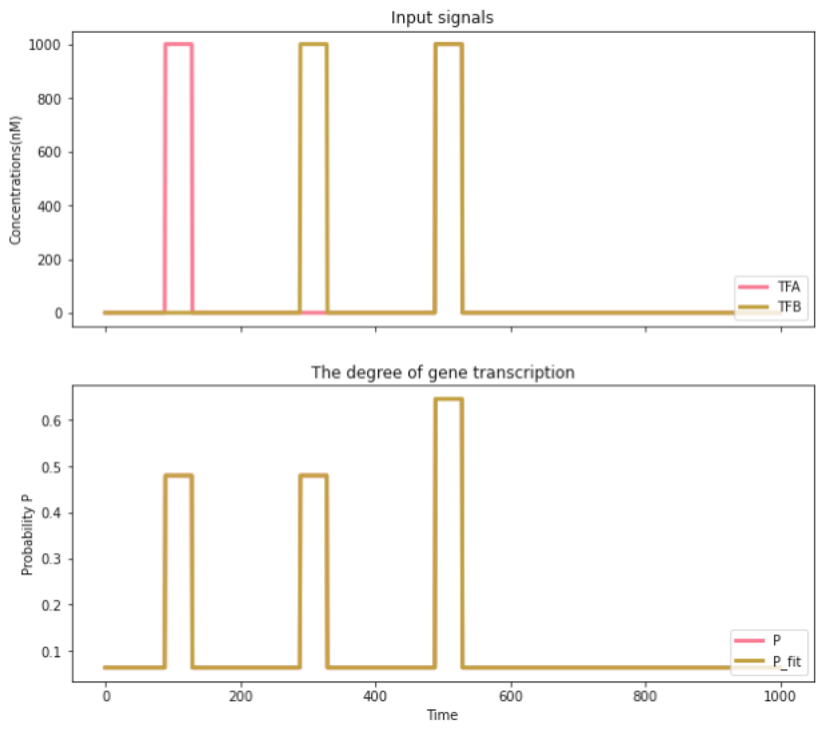

Figure 10: Graph of OR gate's original and fit function equilibrium probability results for $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$ under changing input states

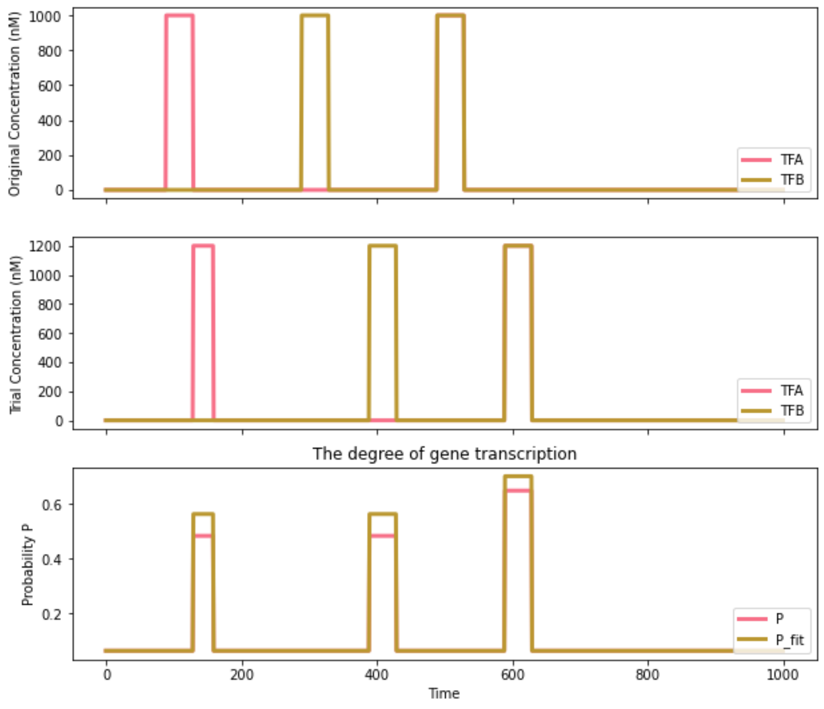

Figure 11: OR gate's original and trial input concentration graphs and P plots for trial concentrations for $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$

4.2.3 OR Gate Protein Generation Simulation

As it could be seen in Table 7, OR gate has four states, as well. Therefore, the same steps explained in Section 4.1.3 are followed for OR gate. Figure 12 shows protein generation of the programmed OR gate.

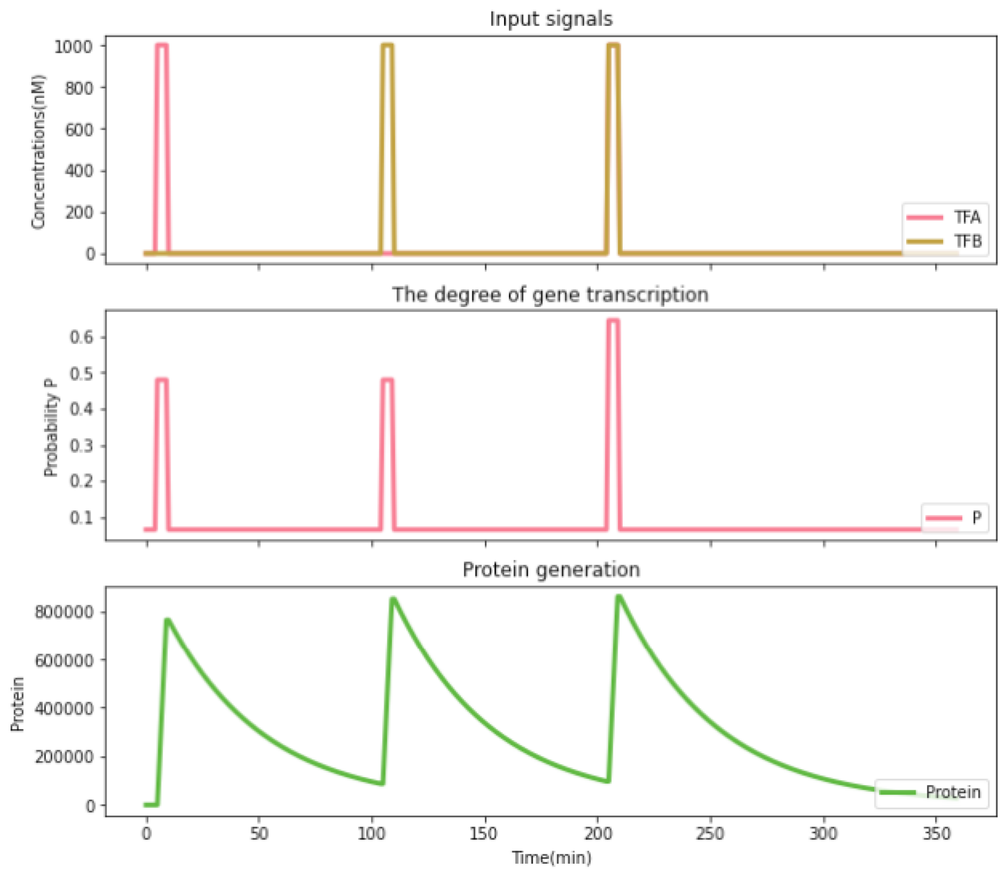

Figure 12: Input concentrations of TF_A and TF_B, P plot and protein generation of OR gate for $K_A = K_B = 100$, $q_p = 1/20$ and $w = 20$

4.3 XOR Gate

In this section formulation of XOR gate with using both single and double prometer are given. Also, fitting of these gates are investigated and their protein generation simulations are given.

4.3.1 XOR GATE SINGLE PROMOTER

4.3.1.1 XOR Gate Single Promoter Formulation

Formulation and parameters for a XOR gate with single promoter are as follows according to the Buchler et al. [29].

$$q_p = 1/10 \tag{24}$$

$$Z_{OFF} = (1 + q_{A2} + q_{B2} + wq_{A2}q_{B2}).(1 + q_{A1} + q_{B1} + q_{A1}q_{B1}) \tag{25}$$

$$Z_{ON} = q_p(1 + wq_{A1} + wq_{B1} + 2wq_{A1}q_{B1}) \tag{26}$$

$$K_{A1} = K_{B1} = 200 \tag{27}$$

$$K_{A2} = K_{B2} = 900 \tag{28}$$

The heat map generated using this formulas is given in Figure 13

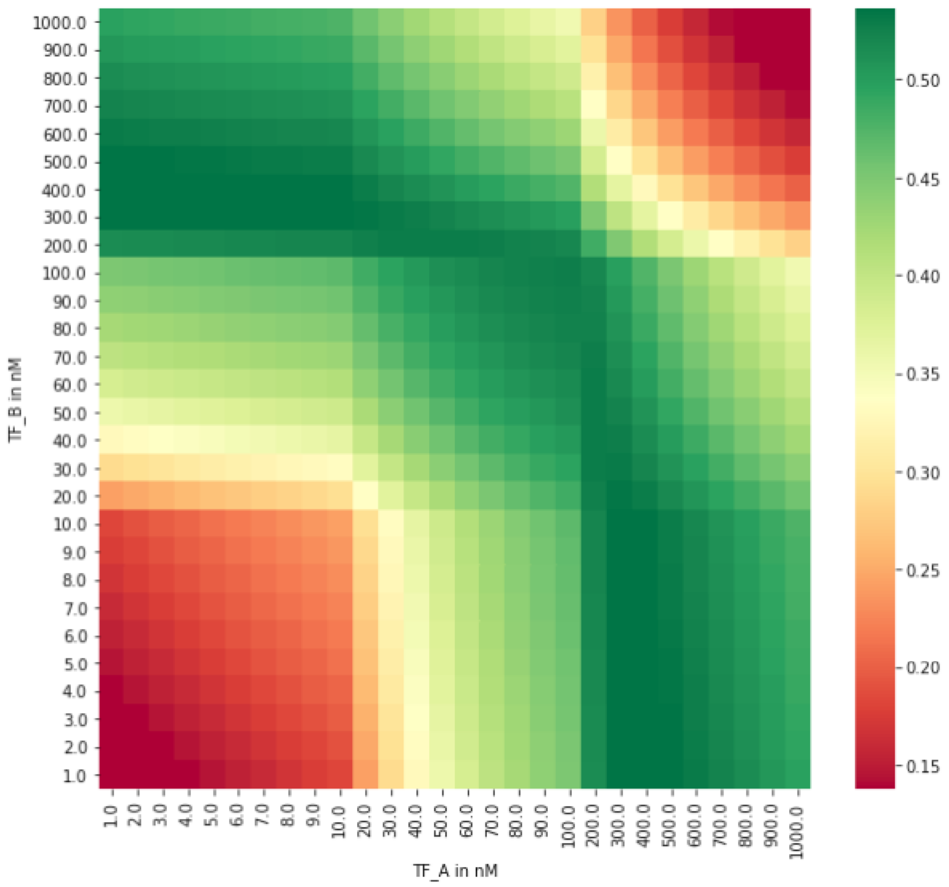

Figure 13: Heat map of XOR gate with single promoter for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 900$, $q_p = 1/10$ and $w = 25$

4.3.1.2 XOR Gate Single Promoter Fitting

The same analysis described in Section 4.1.2 is applied to XOR gate with single promoter to see if a simpler formula could be found to use as a substitute. It turns out that XOR gate single promoter could also be fitted to a simpler formula.

The probability of promoter occupancy under different input combinations could be seen in Figure 14.

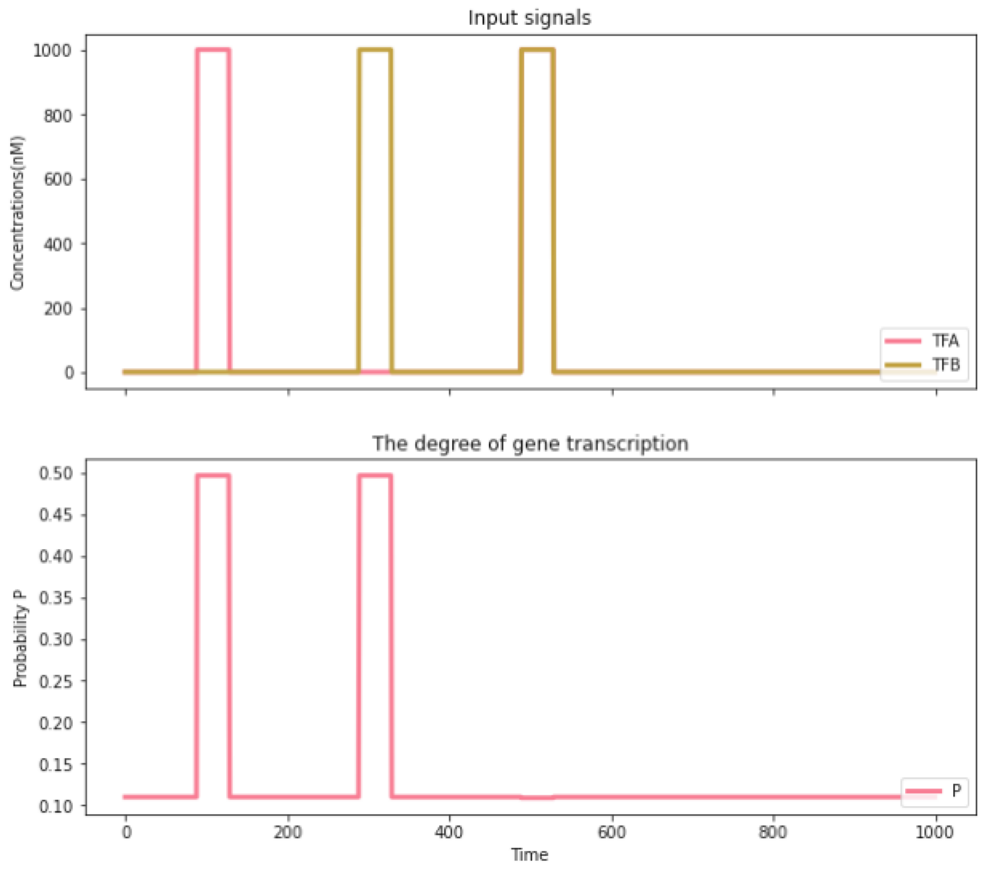

Figure 14: XOR gate with single promoter's probability plot for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 900$, $q_p = 1/10$ and $w = 25$ under changing input states

The chosen formula to be fitted is given in Equation 29.

$$a\,[TF_A] + b\,[TF_B] + c\,\frac{[TF_A]}{[TF_B]} + d \tag{29}$$

In order to calculate the coefficients,"curve_fit" [35] function form SciPy library is used just as it is described in the Section 4.1.2.

The calculated coefficients could be seen in Table 12 and R score of this coefficients is 1.00. In the top plot of Figure 15, the original input concentarations used for coefficient calculation are given. When this coefficients are tested under the trial input concentrations given in the middle plot of Figure 15, R score drops down to 0.93. In the bottom plot of Figure 15, the real probability and the fitted probability calculated for trial input concentrations are shown.

Table 12: XOR gate with single promoter's fit formula coefficients

a	b	c	d
-0.000389	0.000388	0.000775	0.109233

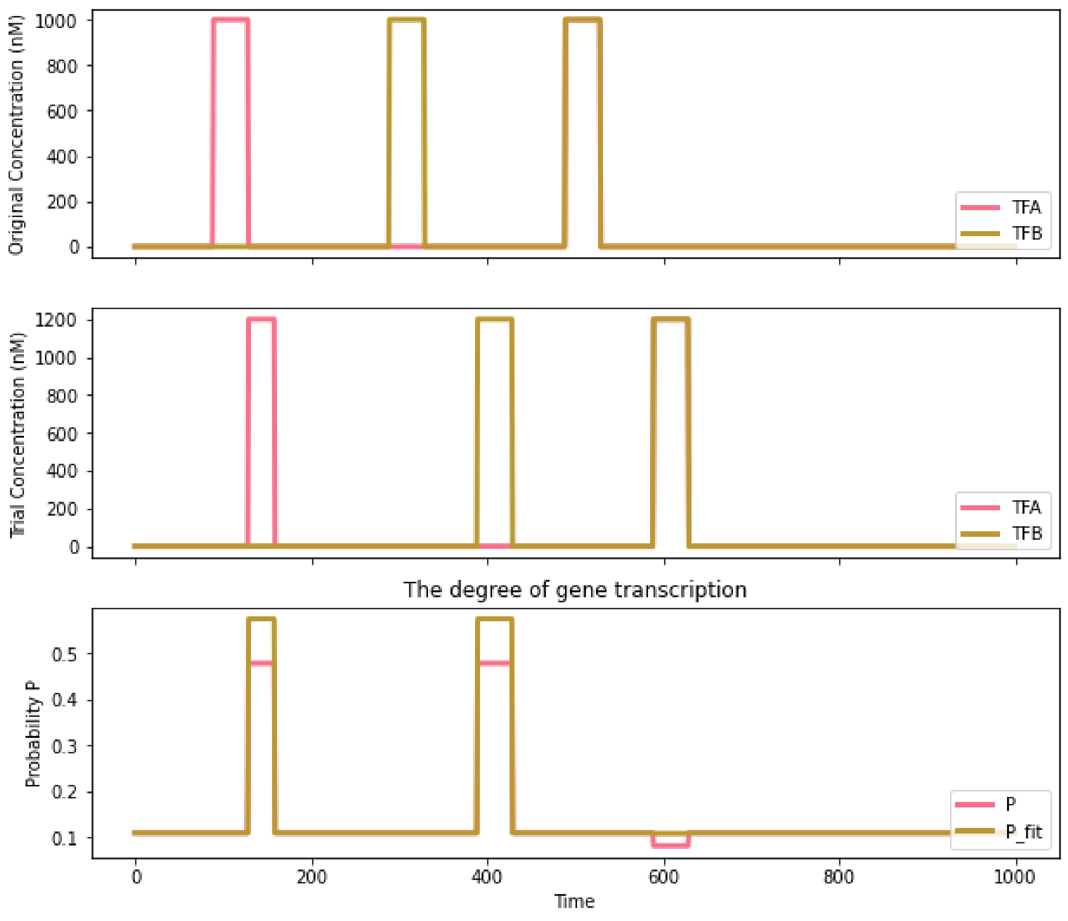

Figure 15: XOR gate with single promoter's original and trial input concentration graphs and P plots for trial concentrations for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 900$, $q_p = 1/10$ and $w = 25$

4.3.1.3 XOR Gate Single Promoter Protein Generation Simulation

In order to see how the protein generation of XOR gate with single promoter at four different states given in Table 7 is, the procedure defined in Section 4.1.3 is followed. The results of the procedure is given in Figure 16.

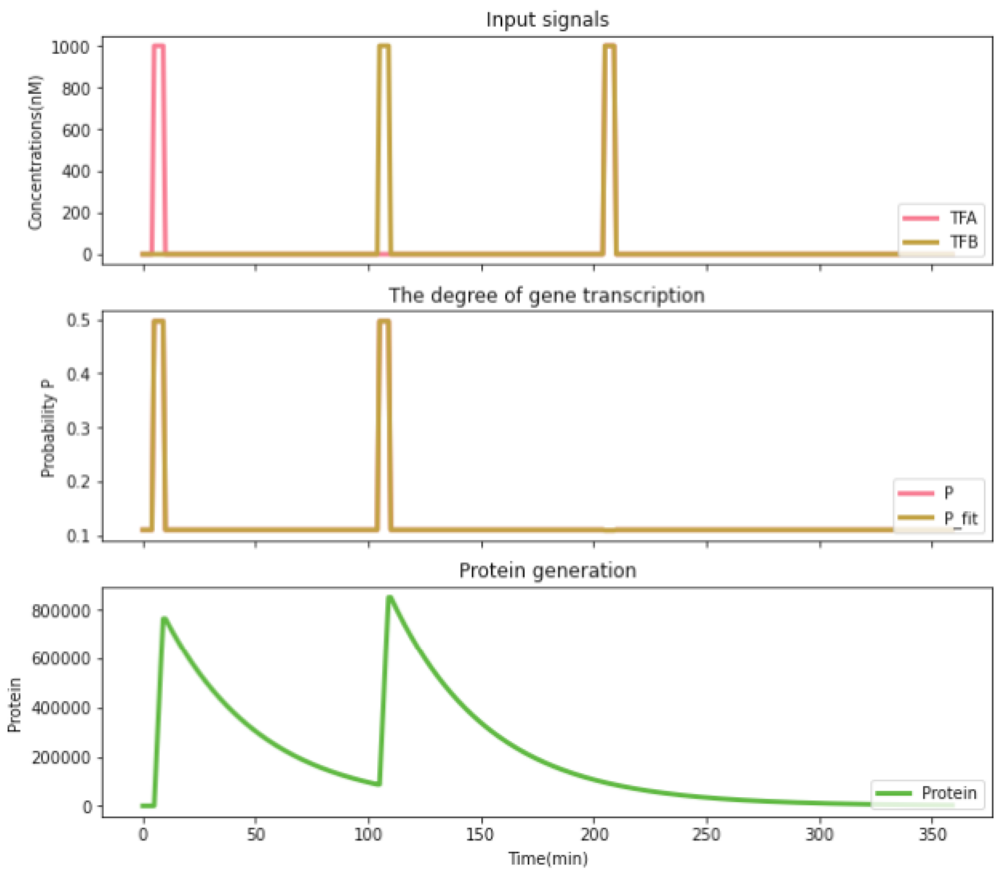

Figure 16: Input concentrations of TF_A and TF_B, P plot and protein generation of XOR gate with single promoter for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 900$, $q_p = 1/10$ and $w = 25$

4.3.2 XOR GATE DOUBLE PROMOTER

4.3.2.1 XOR Gate Double Promoter Formulation

XOR gate with double promoter formulations given by Buchler et al.[29] are listed below.

$$q_{p1} = q_{p2} = 1/20 \tag{30}$$

$$Z^1_{OFF} = (1 + q_{A1})\,(1 + q_{B1}) \tag{31}$$

$$Z^2_{OFF} = (1 + q_{A2})\,(1 + q_{B2}) \tag{32}$$

$$Z^1_{ON} = q_{p1}\,(1 + w\,q_{A1}) \tag{33}$$

$$Z^2_{ON} = q_{p2}\,(1 + w\,q_{B2}) \tag{34}$$

$$K_{A1} = K_{B2} = 500 \tag{35}$$

$$K_{A2} = K_{B1} = 100 \tag{36}$$

In Figure 17, heat map generated using these formulas is given.

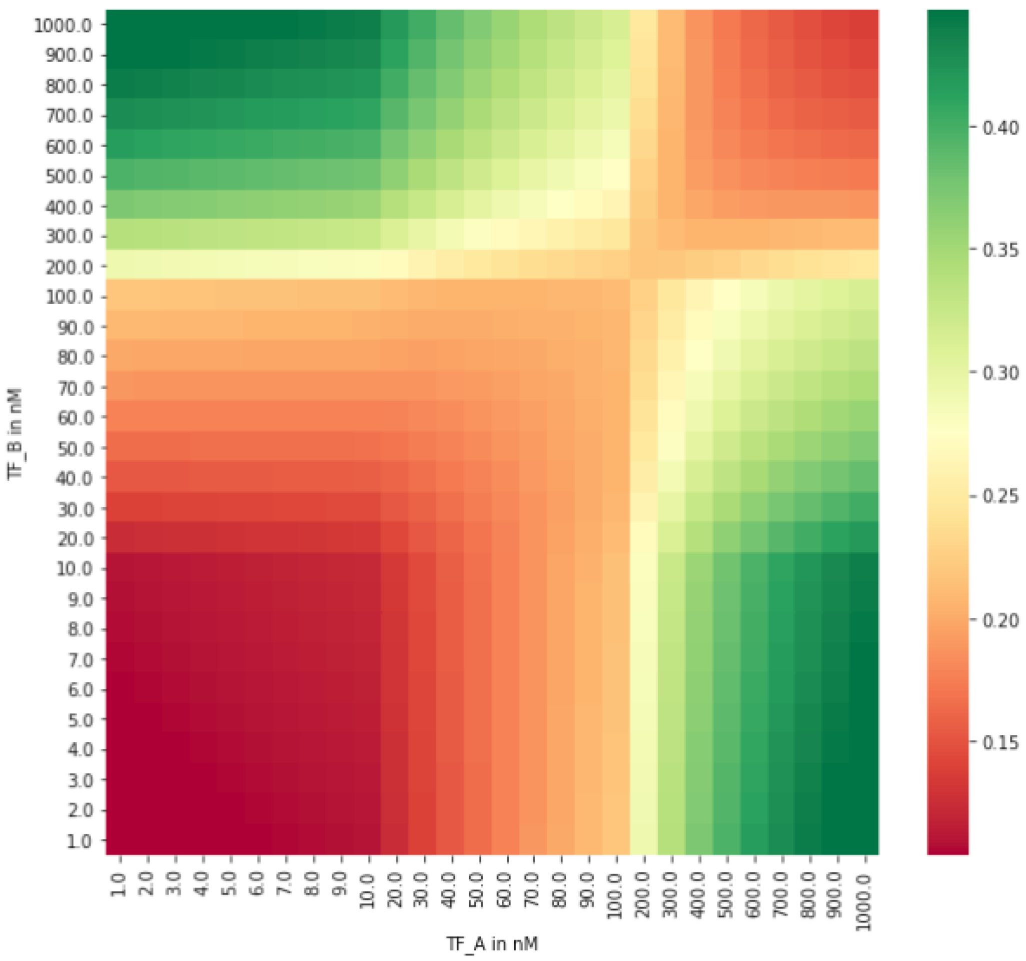

Figure 17: Heat map of XOR gate with double promoter for $K_{A1} = K_{B2} = 500$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$

4.3.2.2 XOR Gate Double Promoter Formulation Fitting

With an aim of finding a simpler formula for the XOR gate with double promoter, the steps defined in Section 4.1.2 is followed. It is found out that a simpler formula could be used in the interval of [1, 1000] nM for $[TF_A]$ and $[TF_B]$. The equilibrium probability of XOR gate with double promoter for different input states could be seen in Figure 18.

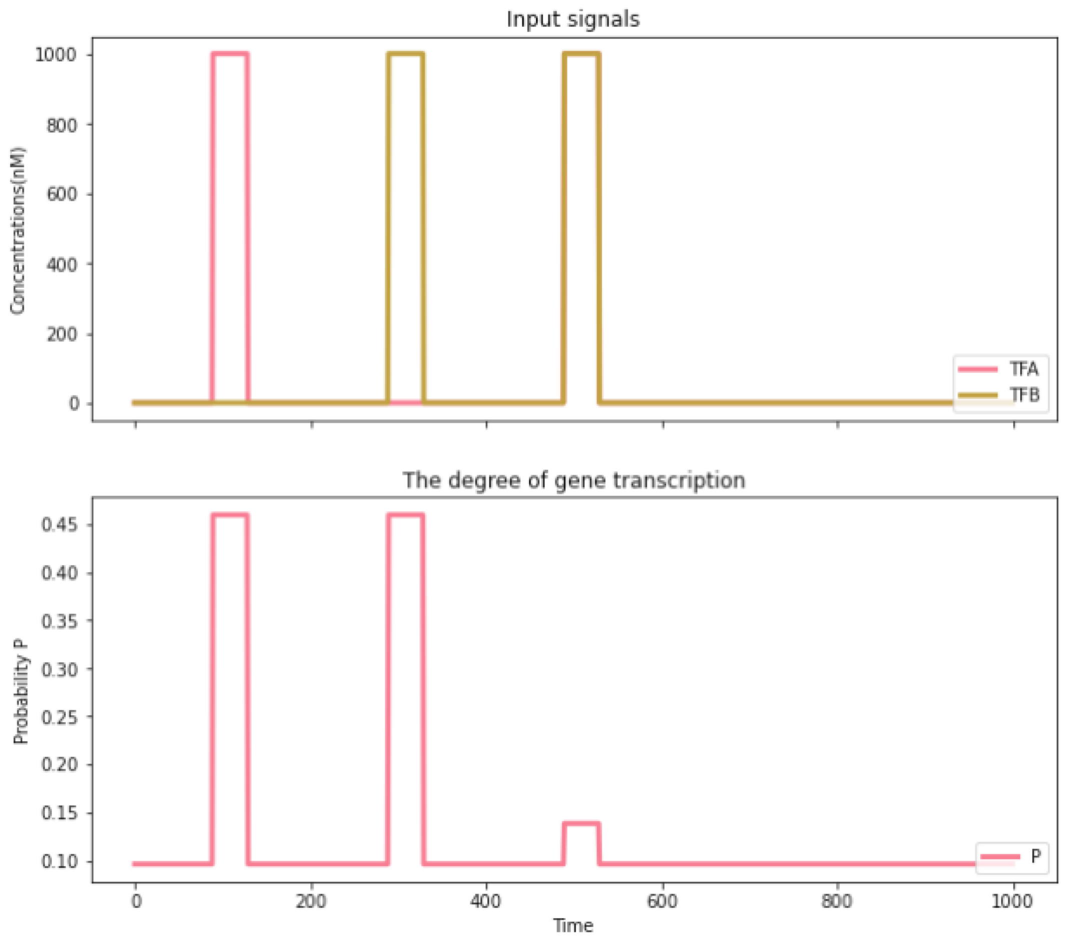

Figure 18: XOR gate with double promoter's probability plot for $K_{A1} = K_{B2} = 500$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$ under changing input states

The chosen formula that is given below for XOR gate with double promoter is the same formula used for XOR gate with single promoter, which is given in Equation 29.

$$a\,[TF_A] + b\,[TF_B] + c\,\frac{[TF_A]}{[TF_B]} + d$$

"curve_fit" [35] function form SciPy library is used to calculate the coefficients, just as it is described in Section 4.1.2.

The calculated coefficients could be seen in Table 13, and the calculated R score for this fit is 1.0. R score drops down to 0.97, if these coefficients are tested under the trial input concentrations given in the middle plot of Figure 19. The original concentrations used to calculate coefficients, the trial concentrations used for testing purposes, and the real and the fitted probabilities calculated for trial concentrations are shown in Figure 19.

Table 13: XOR gate with double promoter's fit formula coefficients

a	b	c	d
0,00032	0,00036	-0,00068	0,095476

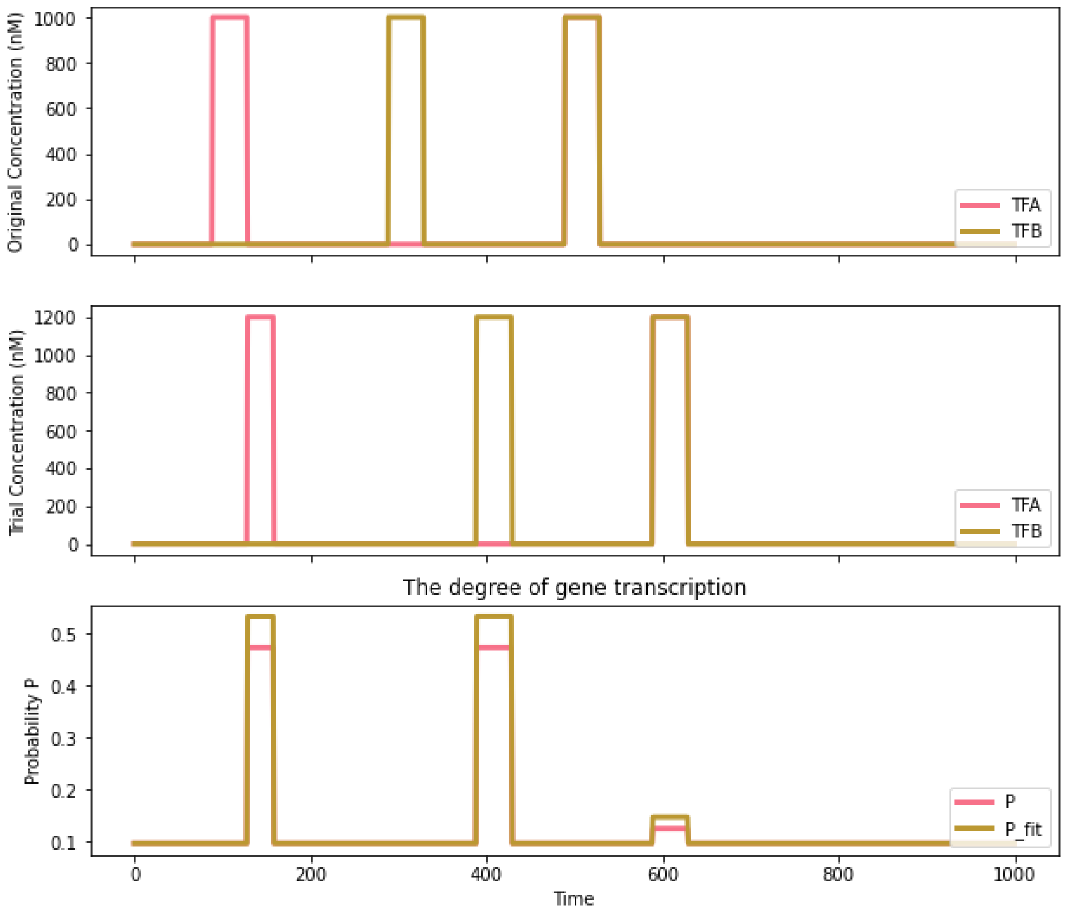

Figure 19: XOR gate with double promoter's original and trial input concentration graphs and P plots for trial concentrations for $K_{A1} = K_{B2} = 500$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$

4.3.2.3 XOR Gate Double Promoter Protein Generation Simulation

The procedure explained in Section 4.1.3 is followed for XOR gate with double promoter, as well. In Figure 20, the protein generation of XOR gate with double promoter can be seen.

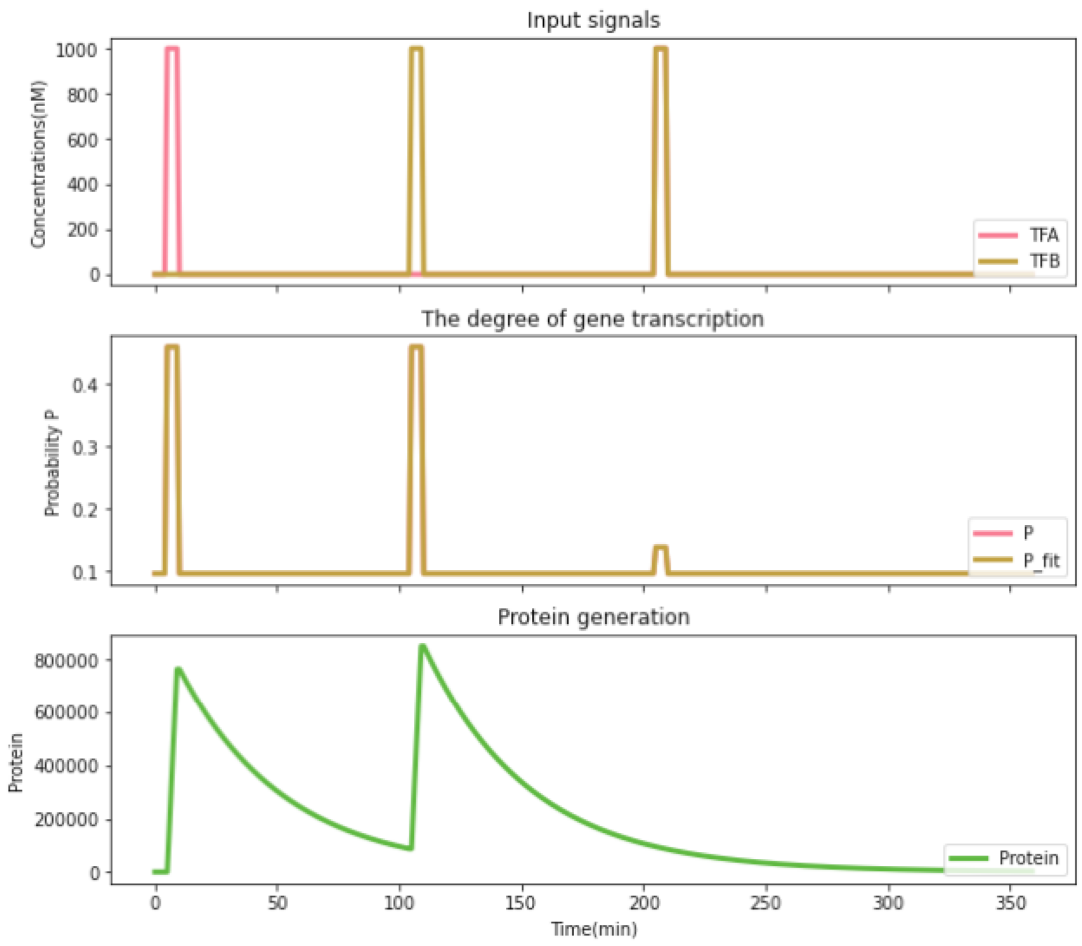

Figure 20: Input concentrations of TF_A and TF_B, P plot and protein generation of XOR gate with double promoter for $K_{A1} = K_{B2} = 500$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$

CHAPTER 5

RESULTS

In this part of the thesis, results found while investigating the characteristic of each logic gate covered in Chapter 4 are shared.

As it could be seen in all of the figures that shows the protein generation characteristics, transcription happens much faster than translation that it is negligible.

5.1 AND Gate Results

When one looks at the equilibrium probability of RNAp binded to promoter given in the bottom part of Figure 4, it is seen that when one of the transcription factor is absent, there is still more than 10% probability that gene is "ON". However, Figure 7 shows that even though there is more than 10% probability, protein generation is absent. That is because gene is "OFF" as it is accepted in the beginning of thesis that the gene would not considered to be ON below 40% of probability. Thus, AND gate could be considered liable.

However, if the dissociation parameters are changed as $K_A = 100$, $K_B = 100$, and q_p and ω are kept as they are given in Section 4.1.1, it can be seen that AND gate turns into OR gate. This could seen in the heat map in Figure 21.

Similarly, if K_A, K_B and ω are kept as they are given in Section 4.1.1, and q_p is changed to 35, AND gate would turn into OR gate. The heat map of AND gate with these parameters is given in Figure 22.

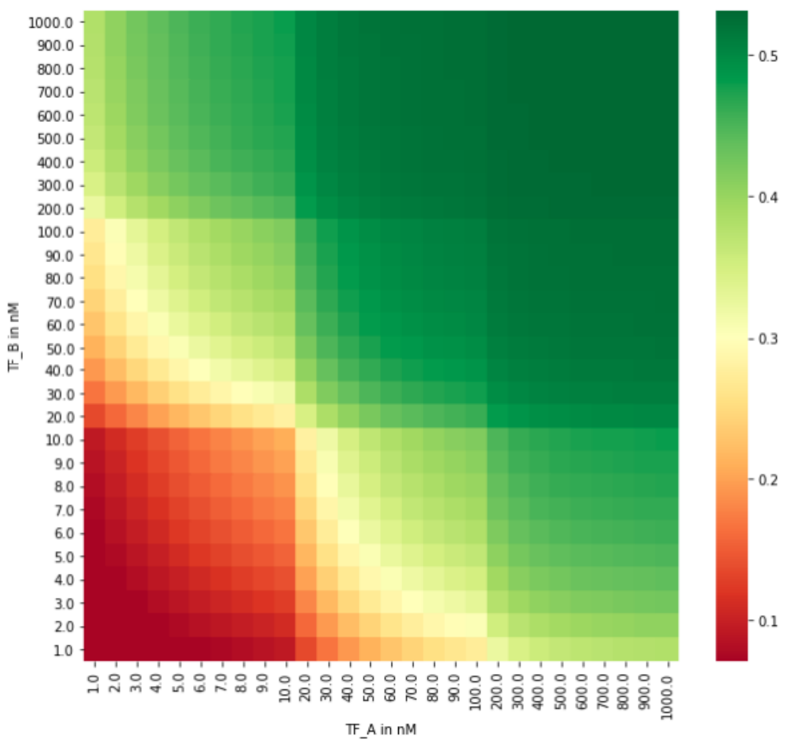

Figure 21: Heat map of AND gate for $K_A = K_B = 100$, $q_p = 1/35$ and $w = 20$

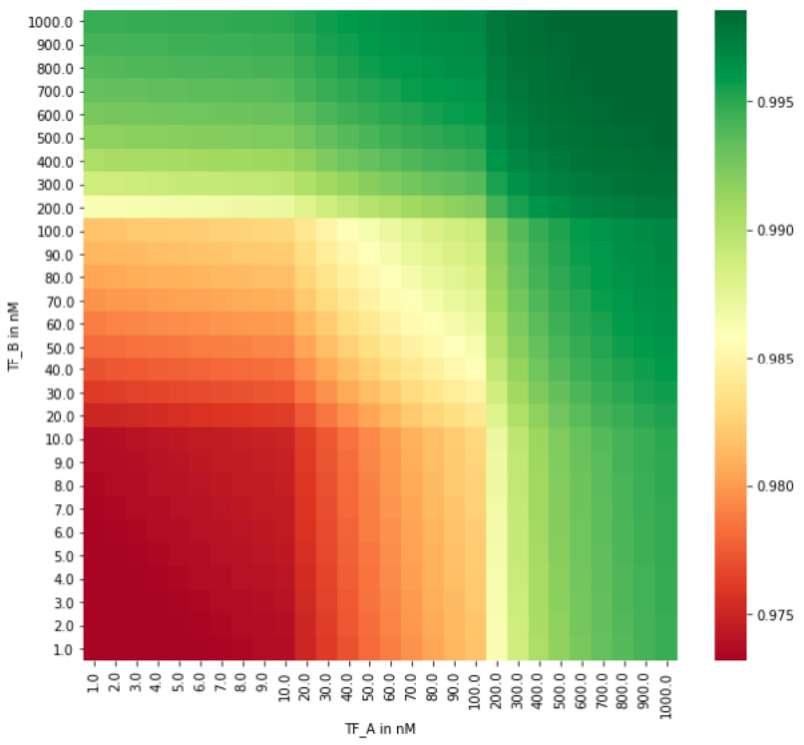

Figure 22: Heat map of AND gate for $K_A = K_B = 3500$, $q_p = 35$ and $w = 20$

5.2 OR Gate Results

As seen in Figure 10, OR Gate gives more than 40% equilibrium probability whenever either one of the dissociation constants or both of them are present. Also, it is worth to notice that equilibrium probability is more than 60%, when both of the dissociation constants are present. However, as it could be seen in Figure 12, protein generation curve does not speed up or down. That is because, as given in Equation 12, response time, τ, does not depend on the equilibrium probability.

As it is described in Section 4.2.1, the parameters for OR Gate are as follows: $K_A = K_B = 100$, $q_p = 1/20$, and $\omega = 20$. With this parameter set, the formulas provided in Section 4.2.1 works as a typical OR gate. However, if parameters are changed as $K_A = 3500$, $K_B = 3500$, and, q_p, and ω are kept the same, OR gate shows itself only for higher concentrations of TF_A and TF_B as seen in Figure 23.

On the other hand, if K_B is changed as $K_B = 4000$, while K_A, q_p, and ω are kept the same, OR gate follows the change of $[TF_A]$. The heat map generated with these parameters is given in Figure 24.

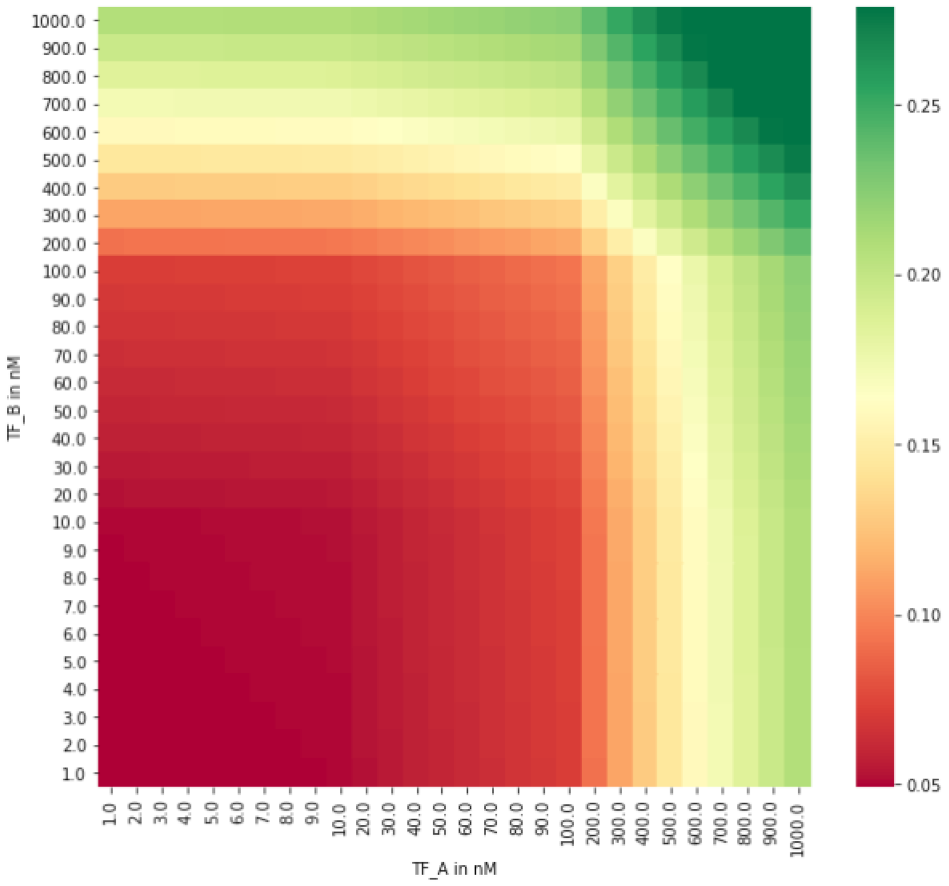

Figure 23: Heat map of OR gate for $K_A = K_B = 3500$, $q_p = 1/20$ and $w = 20$

Similarly, if K_A is changed as $K_A = 4000$, while K_B, q_p, and ω are kept the same, OR gate follows the change of $[TF_B]$. The heat map generated with these parameters is given in Figure 25.

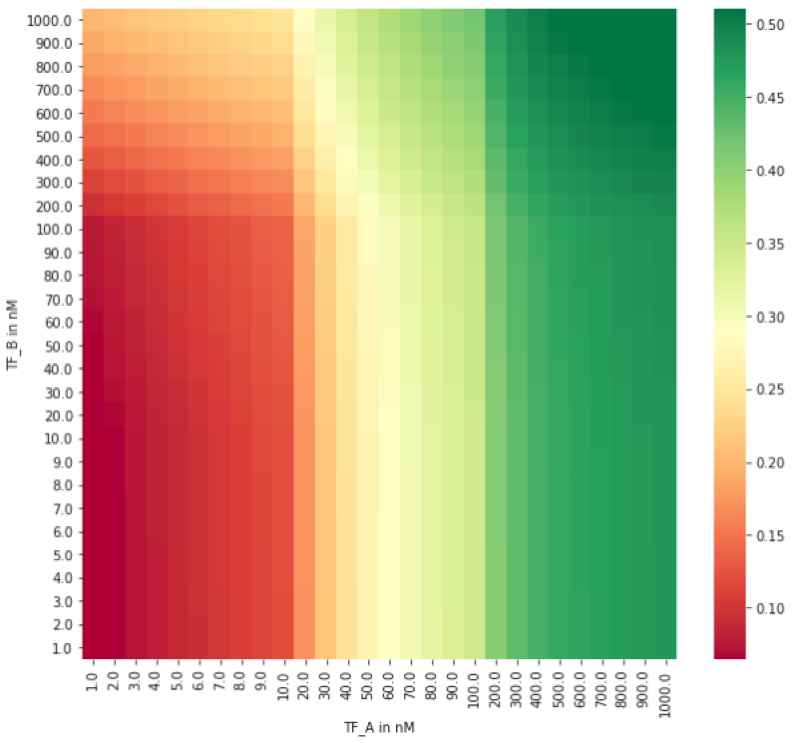

Figure 24: Heat map of OR gate for $K_A = 100$, $K_B = 4000$, $q_p = 1/20$ and $w = 20$

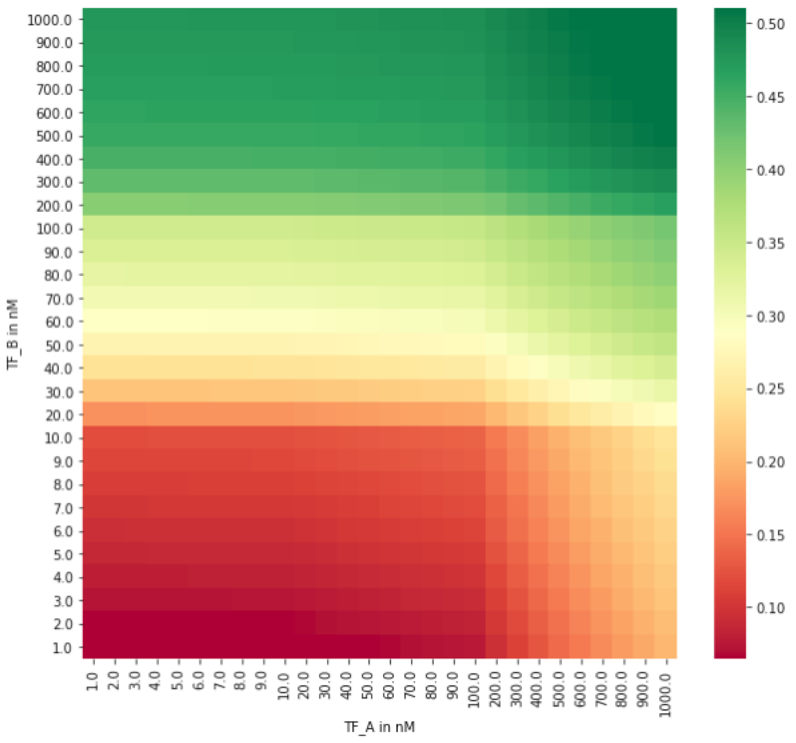

Figure 25: Heat map of OR gate for $K_A = 4000$, $K_B = 100$, $q_p = 1/20$ and $w = 20$

5.3 XOR Gate Single Promoter Results

XOR Gate Single Promoter probability graph given in Figure 14 shows that there is approximately 50% probability that the gene is "ON" whenever any of the transcription factors is present. Also, it could be seen in the heat map given in Figure 13 that gene has more "OFF" area when both of the transcription factors are absent compared to when both of them are present.

The heat maps in Figure 26 and Figure 27 are constructed using the same formulation and parameters given in the XOR Gate Single Promoter except K_{A2} and K_{B2} values.

For Figure 26, the values of K_{A2} and K_{B2} are increased to 1000. Similarly, for Figure 27, the values of K_{A2} and K_{B2} are increased to 1500. It could be seen that as K_{A2} and K_{B2} increases, XOR gate single promoter loses its "OFF" state at higher concentrations of transcription factors.

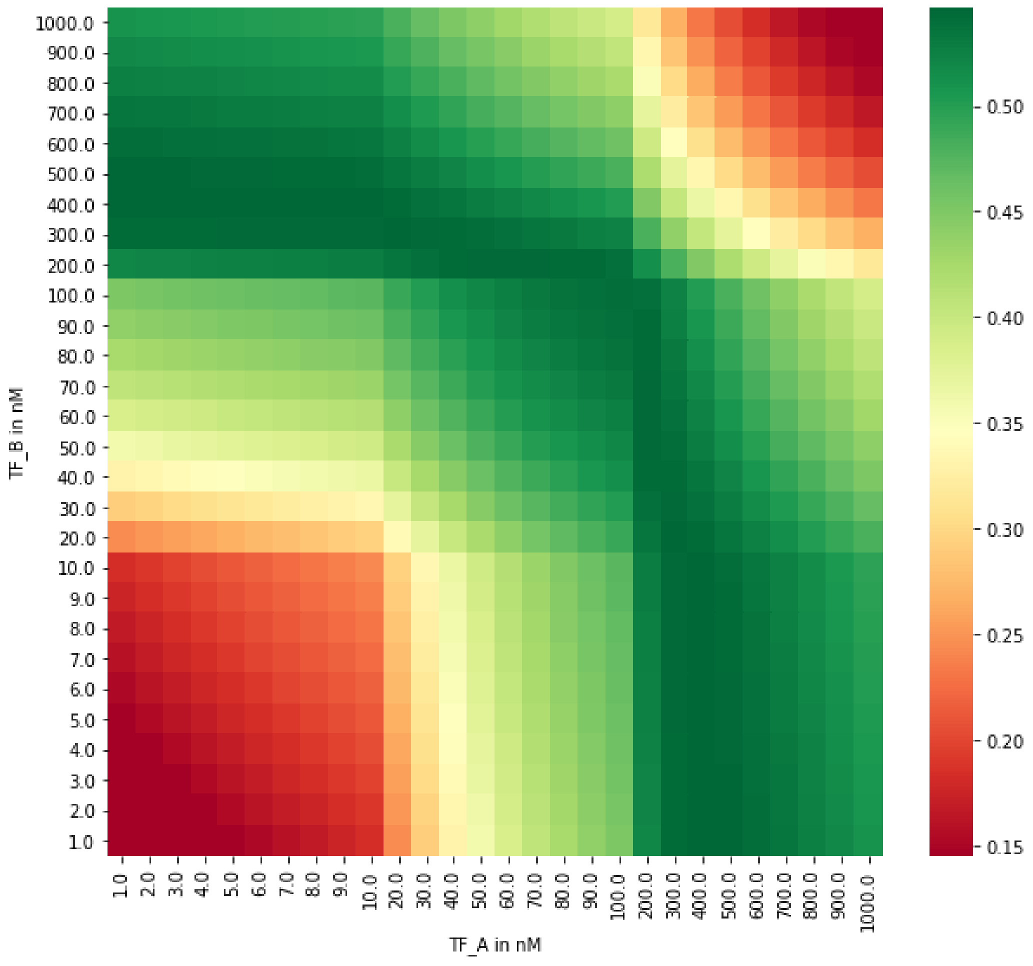

Figure 26: Heat map of XOR gate with single promoter for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 1000$, $q_p = 1/10$ and $w = 25$

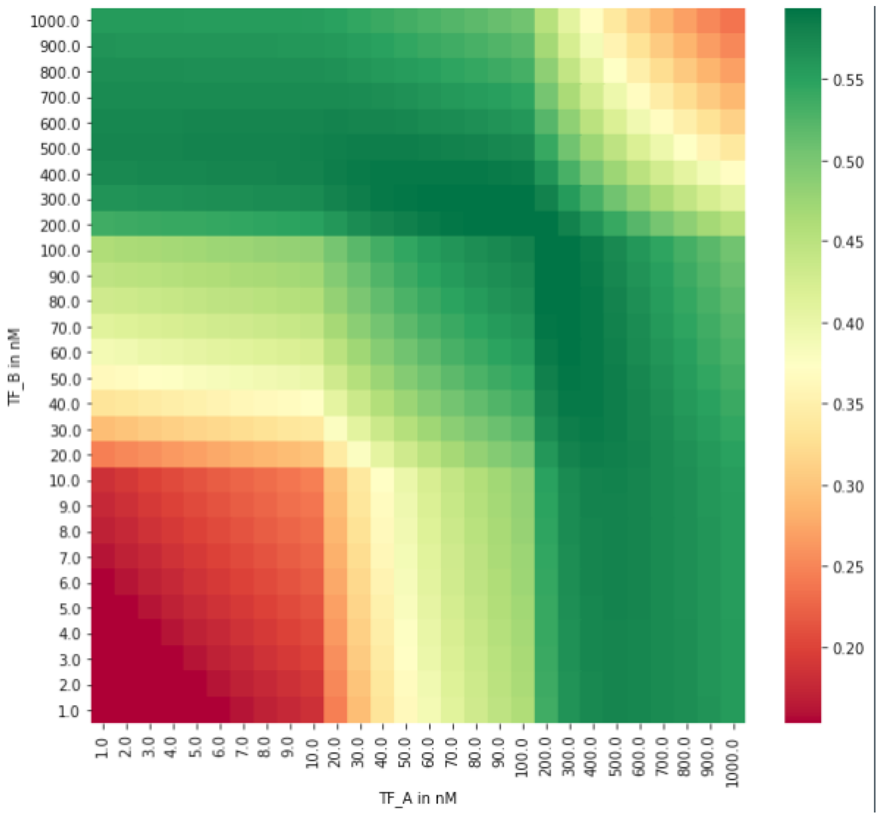

Figure 27: Heat map of XOR gate with single promoter for $K_{A1} = K_{B1} = 200$, $K_{A2} = K_{B2} = 1500$, $q_p = 1/10$ and $w = 25$

5.4 XOR Gate Double Promoter Results

As seen in Figure 18, probability of gene activation exceeds 40% whenever one of the concentrations of TF_A and TF_B is high and the other one is low. However, it could also be seen in the same figure that when both transcription concentrations are high, there is approximately 15% probability that gene is "ON". When looked at the XOR gate single promoter probability plot given in Figure 14 for the same case, it is seen that XOR gate single promoter works better than double promoter. Therefore, it could be said that double promoter is leakier than single promoter. However, even if there is slight increase in the probability, it does not affect the protein concentration as seen in Figure 16 since gene is taken as "ON" when probability exceeds 40%.

As given in Section 4.3.2.1, default parameters given by Buchler et al. [29] for XOR Gate Double Promoter are as follows; $K_{A1} = K_{B2} = 500$, $K_{A2} = K_{B1} = 100$ and $q_{p1} = q_{p2} = 1/20$. If dissociation constants for A_1 and B_2 are changed as $K_{A1} = K_{B2} = 900$ while keeping the other parameters unchanged the plots given in Figure 28 is obtained. It seen that probabilities for "ON" states are decreased to almost 40%.

If the same dissociation constants are further increased as $K_{A1} = K_{B2} = 1000$, probability falls below 40% for every state and protein generation stops. This case is given in Figure 29.

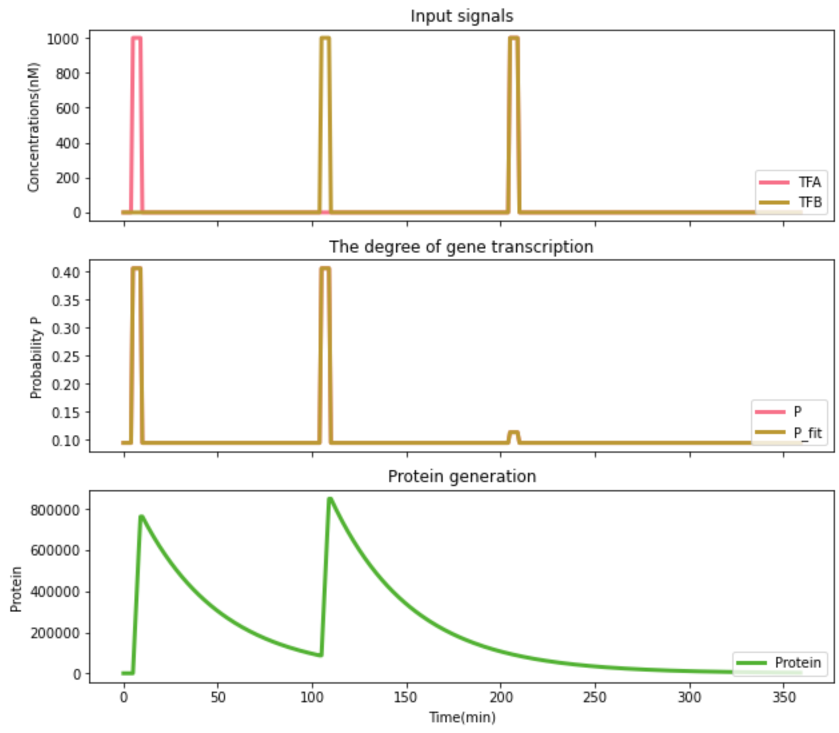

Figure 28: Input concentrations of TF_A and TF_B, P plot and protein generation of XOR gate with double promoter for $K_{A1} = K_{B2} = 900$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$

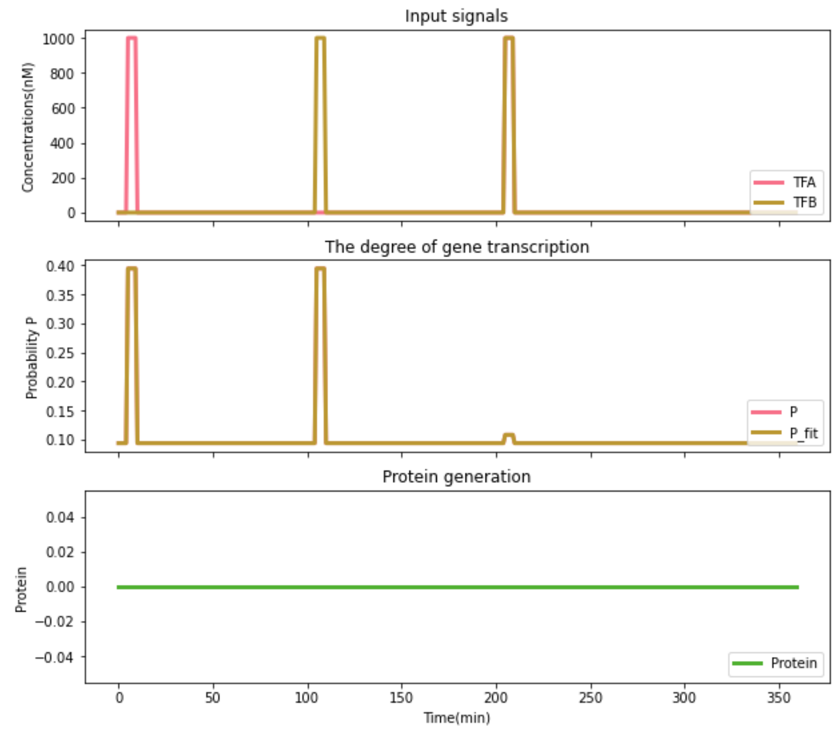

Figure 29: Input concentrations of TF_A and TF_B, P plot and protein generation of XOR gate with double promoter for $K_{A1} = K_{B2} = 1000$, $K_{A2} = K_{B1} = 100$, $q_{p1} = q_{p2} = 1/20$ and $w = 25$

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis; AND, OR and XOR logic gates in a bacteria cell are modelled by using the Shea and Ackers's [30] thermodynamic theory for a bacteria. In order to model these logic gates, formulations laid out by Buchler et al. [29] are implemented.

The probability of a RNA polymerase (RNAp) bound to a specific promoter is the core element in thermodynamic modeling. The main assumption behind this approach is that gene expression level is proportional to the probability of a specific promoter binding to RNA polymerase. The idea is that the equilibrium probability, P, of the binding between RNA polymerase and promoter resulting from their interactions with bound transcription factors could be used to measure the gene transcription degree.

Another assumption in this thesis is when modelling AND, OR and XOR gates in bacteria, promoter occupancy larger than 40% is excepted as sufficient for a gene to be “ON”, any occupancy less then 40% is excepted as "OFF".

With these assumptions; AND and OR gates are modelled for a single promoter system and XOR gate is modelled for both single and double promoter system. Similar heatmaps provided by Buchler et al. are constructed [29]. However, if the default parameter values are changed, it is seen that logic gates might change behaviours. As seen in Section 5.1, it is possible to turn AND gate into OR gate by changing a few parameters and as seen in Section 5.2, OR gate might mirror the changes in the concentration of either one of the transcription factors. As for the XOR gates, it is discovered that double promoter is leakier than single promoter for the current input combinations. However, this does not mean that for each configuration, single promoter is going to perform better than double promoter. Further investigations might disproof this conclusion.

Buchler et al.[29] described the formulation for logic gates, and created heat maps accordingly. As a contribution to the study simpler functions are sought as a fit in this thesis to achieve fast and accurate results. For AND gate Equation 18 is used; for OR gate Equation 23 is used and for XOR gates Equation 29 is used as fit functions. Transcription factor concentrations are changed in different combinations of step function to represent all values that corresponds to the rows of Table 7 and coefficients of the aforementioned equations are calculated using the "curve_fit" function [35] from SciPy library. In order to calculate the overall performance of fit functions, R score is calculated for both the training input data set and the trial input data set. The trial data set is still a combination of step functions; yet, it's amplitude and time values are different than the training input data set. When R scores are compared, it is seen that from training input sets to trial input datasets, there is a slight decrease in the R score performances which might indicate that fit functions might not be as robust as wanted. Optimization of the overall performance of fit functions and finding ways to improve them could be the work of another study.

A cell produces a protein y, whenever a gene Y is activated as explained by Uri Alon [33]. Rate of protein production, protein degradation and protein dilution are the 3 variables that affects the protein concentration in the cell.

In this thesis, protein generation simulation of an average E.coli bacteria is simulated using the formulation provided by Uri Alon [33] to see how it might differ from gate to gate. As expected; for AND gate, protein generation observed only when both transcription factors are present. For OR gate, whenever either one of the transcription factor is present or both them are present, protein generation is observed. As for the XOR gates, protein generation observed only when either one of the transcription factor concentration is high but not when both of them are high. However, since the protein generation is assumed to be started whenever the promoter occupancy exceeded 40%, the probable affects of higher promoter occupancy values are not observed on protein generation.

It is possible to support high-scaled projects such as cleaning environmental damages, increasing agricultural efficiency and producing personalized medicine based on controlled bacteria activity. All of the conducted study in this thesis implies that gene regulation in bacteria is the key element in controlling its activity and creating live logic gates. Creating a complex genetic circuits and designing algorithms based on live logic gates are left as a future work.

REFERENCES

[1] G. Faccio, *Proteins as Nanosized Components of Biosensors*, pp. 229–255. 01 2019.

[2] L. Learning, "Biology for majors I." `https://courses.lumenlearning.com/suny-wmopen-biology1/chapter/outcome-prokaryotic-gene-regulation/`.

[3] D. Ellenby, "The five: New uses for bacteria." `https://www.theguardian.com/technology/2019/jul/21/five-new-uses-for-bacteria-graphene-plastic-waste-oil-spills-tumours`, Jul 2019.

[4] S. B. Carroll, J. K. Grenier, and S. D. Weatherbee, *From DNA to diversity: Molecular genetics and the evolution of Animal Design*. Blackwell, 2010.

[5] X. He, M. A. Samee, C. Blatti, and S. Sinha, "Thermodynamics-based models of transcriptional regulation by enhancers: The roles of synergistic activation, cooperative binding and short-range repression," *PLoS Computational Biology*, vol. 6, no. 9, 2010.

[6] L. Hetherington, "What is RNA." `https://www.rnasociety.org/what-is-rna`.

[7] M. J. Lopez and S. S. Mohiuddin, *Biochemistry, Essential Amino Acids*. Treasure Island (FL): StatPearls Publishing, 2022.

[8] A. LaPelusa and R. Kaushik, *Physiology, Proteins*. Treasure Island (FL): StatPearls Publishing, 2022.

[9] G. Wu, "Amino acids: Metabolism, functions, and nutrition," *Amino Acids*, vol. 37, no. 1, p. 1–17, 2009.

[10] "Biomolecules: Proteins 2." `https://www2.chem.wisc.edu/deptfiles/genchem/netorial/modules/biomolecules/modules/protein2/prot28.htm`.

[11] R. J. Ouellette and J. D. Rawn, "Amino acids, peptides, and proteins," *Principles of Organic Chemistry*, p. 371–396, 2015.

[12] S. Sen K., "Ribonucleic acid (RNA)." `https://www.genome.gov/genetics-glossary/RNA-Ribonucleic-Acid`.

[13] "Deoxyribonucleic acid (DNA) fact sheet." `https://www.genome.gov/about-genomics/fact-sheets/Deoxyribonucleic-Acid-Fact-Sheet`.

[14] E. Green, "Gene." `https://www.genome.gov/genetics-glossary/Gene`.

[15] "What is the Human Genome Project?." `https://www.genome.gov/human-genome-project/What`.

[16] "What is a gene?: Medlineplus genetics." `https://medlineplus.gov/genetics/understanding/basics/gene/`.

[17] J. Segre, "Gene regulation." `https://www.genome.gov/genetics-glossary/Gene-Regulation`.

[18] G. Gill, K. Chapman, and S. Higgins, "Regulation of the initiation of eukaryotic transcription," *Essays in Biochemistry*, vol. 37, p. 33–43, 2001.

[19] "George boole develops boolean algebra." `https://historyofinformation.com/detail.php?entryid=565`.

[20] K. Levitz and H. Levitz, *Logic and boolean algebra*. Barron's Educational Series, 1979.

[21] J. E. Whitesitt, *Boolean algebra and its Applications*. Dover Publications; Illustrated edition (March 18, 2010).

[22] W. Kenton, "Boolean algebra." `https://www.investopedia.com/terms/b/boolean-algebra.asp`, Jun 2022.

[23] "Basic logic gates with truth tables - digital logic circuits." `https://www.elprocus.com/basic-logic-gates-with-truth-tables/`, Dec 2020.

[24] J. C. Anderson, C. A. Voigt, and A. P. Arkin, "Environmental signal integration by a modular and gate," *Molecular Systems Biology*, vol. 3, no. 1, p. 133, 2007.

[25] B. Wang, R. I. Kitney, N. Joly, and M. Buck, "Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology," *Nature Communications*, vol. 2, no. 1, 2011.

[26] NTNU Trondheim, "Creating logic gates in living organisms." `https://2016.igem.org/Team:NTNU_Trondheim/Description`, 2016.

[27] "iGEM." `https://igem.org/`.

[28] A. Ay and D. N. Arnosti, "Mathematical modeling of gene expression: A guide for the perplexed biologist," *Critical Reviews in Biochemistry and Molecular Biology*, vol. 46, no. 2, p. 137–151, 2011.

[29] N. E. Buchler, U. Gerland, and T. Hwa, "On schemes of combinatorial transcription logic," *Proceedings of the National Academy of Sciences*, vol. 100, no. 9, p. 5136–5141, 2003.

[30] M. A. Shea and G. K. Ackers, "The or control system of bacteriophage lambda," *Journal of Molecular Biology*, vol. 181, no. 2, p. 211–230, 1985.

[31] A. González, S. Jafari, A. Zenere, M. Alenius, and C. Altafini, "Thermodynamic model of gene regulation for the OR59B olfactory receptor in drosophila," *PLOS Computational Biology*, vol. 15, no. 1, 2019.

[32] L. Bintu, N. E. Buchler, H. G. Garcia, U. Gerland, T. Hwa, J. Kondev, and R. Phillips, "Transcriptional regulation by the numbers: Models," *Current Opinion in Genetics and Development*, vol. 15, no. 2, p. 116–124, 2005.

[33] U. Alon, *An introduction to systems biology: Design principles of biological circuits*. CRC Press/Taylor and Francis Group, 2020.

[34] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[35] "Scipy.optimize.curve-fit." `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve-fit.html`, May 2022.

APPENDIX A

TABLES FOR RELATED WORK CHAPTER

A.1 And Gate Heat Map

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 27 18:57:04 2022

@author: irem
"""
import matplotlib.pyplot as plt
import numpy as np
import math as math
import seaborn as sn
import pandas as pd

def TF(TF_values):

    for i in range(0,3):
        for j in np.arange(10**(i), 10**(i+1), 10**(i), dtype=float):
            TF_values.append(j)

    TF_values.append(float(10**(i+1)))

def Binding_Affinity(q_i, TF_i, K_i):
    for x in TF_i:
        q_i.append(x / K_i )

def GATE_AND(q_A1, q_B1, w, q_p):

    Z_OFF = (1 + q_A1 + q_B1 + w*q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2 * w * w * q_A1 * q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))


time = np.linspace(1, 100, 100000)

K_A = 3500
K_B = 3500
q_p = 1/35
w = 20
```

```
q_A = []
q_B = []

TFA_values = []
TFB_values = []

TF(TFA_values)
Binding_Affinity(q_A, TFA_values, K_A)

TF(TFB_values)
TFB_values.reverse()
Binding_Affinity(q_B, TFB_values, K_B)

P =[]
for qb in q_B:
    for qa in q_A:
        P.append(GATE_AND(qa, qb, w, q_p))

delta_P = []
P_min = 1.00 #min(P)
for p in P:
    delta_P.append(p /P_min)

delta_P=np.reshape(delta_P,(np.size(TFA_values), np.size(TFB_values)))
plt.clf()
fig, fig_ax = plt.subplots(figsize=(10, 9))
df_cm = pd.DataFrame(delta_P)
fig_ax = sn.heatmap(df_cm, cmap = 'RdYlGn', annot=False, robust=True,
↪  yticklabels=TFB_values, xticklabels=TFA_values, cbar=True)
plt.title('AND GATE. P(i)/Pmin plot for w:' + str(w) +' K_A:' + str(K_A) +
↪  ' K_B:' + str(K_B) + ' q_p:' + str(q_p) + ' and Pmin:' + str(P_min),
↪  loc="center")
plt.xlabel('TF_A in nM')
plt.ylabel('TF_B in nM')

plt.show()
plt.close()
```

A.2 And Gate Function Fitting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May 13 17:43:15 2022

@author: irem
"""

import matplotlib.pyplot as plt
import numpy as np
import math as math
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
```

```
def GATE_AND(TFA,TFB):
    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    Z_OFF = (1 + q_A1 + q_B1 + w*q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2 * w * w * q_A1 * q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))


def GATE_AND_FIT(TF, a, b, c, d, e, f):
    (TFA, TFB) = TF

    return ( a/(b*TFA+c) + d/(e*TFB+f))

K_A = 3500
K_B = 3500
q_p = 1/35
w = 20#cooperativity factor

time = []
time = np.arange(0, 1000, 1, dtype=None)
"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000

TFA_values2 = []
TFB_values2 = []
TFA_values2 = np.ones(1000)
TFA_values2[129:159]=1200
TFA_values2[589:629]=1200
TFB_values2 = np.ones(1000)
TFB_values2[389:429]=1200
TFB_values2[589:629]=1200


P1 = []

for i in time:

    P1.append(GATE_AND(TFA_values[i], TFB_values[i]))
```

```
plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('AND GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪  str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.legend(loc="lower right")
ax1.set_ylabel('TFA Concentration (nM)')
ax1.set_yscale('linear')

ax2.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('TFB Concentration (nM)')
ax2.set_yscale('linear')


ax3.plot(time, P1,    c=sn.color_palette('husl')[2], label='P',
↪  linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()

[m, cov] = curvefit(GATE_AND_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_AND_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_AND_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))


plt.clf()
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('AND GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪  str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
```

```
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()


Ptrial =[]
Ptrialfit =[]
for i in time:
    x = TFA_values2[i]
    y = TFB_values2[i]

    Ptrial.append(GATE_AND(x, y))
    f  = GATE_AND_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Ptrialfit.append(f)
print('R^2 score for P1:', r2_score(Ptrial, Ptrialfit))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('AND GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪   str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪   linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪   linewidth=3)
ax1.legend(loc="lower right")
ax1.set_ylabel('Original Concentration (nM)')
ax1.set_yscale('linear')

ax2.plot(time, TFA_values2, c=sn.color_palette('husl')[0], label='TFA',
↪   linewidth=3)
ax2.plot(time, TFB_values2, c=sn.color_palette('husl')[1], label='TFB',
↪   linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('Trial Concentration (nM)')
ax2.set_yscale('linear')


ax3.plot(time, Ptrial,    c=sn.color_palette('husl')[0], label='P',
↪   linewidth=3)
ax3.plot(time, Ptrialfit, c=sn.color_palette('husl')[1], label='P_fit',
↪   linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()
```

A.3 And Gate Protein Generation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May 13 19:18:33 2022

@author: irem
"""


import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
from scipy import signal


def GATE_AND(TFA,TFB):
    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    Z_OFF = (1 + q_A1 + q_B1 + w*q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2 * w * w * q_A1 * q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))


def GATE_AND_FIT(TF, a, b, c, d, e, f):
    (TFA, TFB) = TF

    return ( a/(b*TFA+c) + d/(e*TFB+f))

K_A = 3500
K_B = 3500
q_p = 1/35
w = 20#cooperativity factor

duration = 360 # dakika
time = []
time = np.arange(0, duration, 1, dtype=None)  # 40 dklik
"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
```

```
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000

P1 = []

for i in time:
    p = GATE_AND(TFA_values[i], TFB_values[i])
    P1.append(p)

[m, cov] = curvefit(GATE_AND_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_AND_FIT, (TFA_values, TFB_values), P1, p0=m)

TFA_values = []
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000
NewProt = []
NewProtTemp = 0


P1 = []
Pcurved = []
beta = 200000.0 # tunable parameter
alpha = 0.023104906018664842 #math.log(2)/30 timescale for 50% change in
↪  concentration of the stable protein is appr. 30 min and alpha_deg=0
Protss = beta/alpha;
ProtInt1 = 0
ProtInt2 = 0
timer1=0
timer2=0

for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_AND_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)

    p = GATE_AND(TFA_values[i], TFB_values[i])
    P1.append(p)
    if p>0.4:
        NewProtTemp = ProtInt1 + Protss*(1-math.exp(1)**(-alpha*timer1))
        ProtInt2 = NewProtTemp
        timer1 = timer1+1
        timer2 = 0
    else:
        NewProtTemp = ProtInt2*math.exp(1)**(-alpha*timer2)
        ProtInt1 = NewProtTemp
        timer2 = timer2+1
        timer1 = 0

    if NewProtTemp <0:
        NewProtTemp = 0
    NewProt.append(NewProtTemp)

print('R^2 score for P1:', r2_score(P1, Pcurved))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('AND GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪   str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪   linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪   linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪   linewidth=3)
#ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪   linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')


ax3.plot(time, NewProt, c=sn.color_palette('husl')[2], label='Protein',
↪   linewidth=3)
ax3.set_title('Protein generation', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Protein')
ax3.set_yscale('linear')
ax3.set_xlabel('Time(min)')

plt.show()
```

A.4 Or Gate Heat Map

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 31 18:51:46 2022

@author: irem
"""

import matplotlib.pyplot as plt
import numpy as np
import math as math
import seaborn as sn
import pandas as pd


def TF(TF_values):

    for i in range(0,3):
        for j in np.arange(10**(i), 10**(i+1), 10**(i), dtype=float):
            TF_values.append(j)

    TF_values.append(float(10**(i+1)))

def Binding_Affinity(q_i, TF_i, K_i):
    for x in TF_i:
        q_i.append(x / K_i )


def GATE_OR(q_A1, q_B1, w, q_p):

    Z_OFF  = (1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON   = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))


time = np.linspace(1, 200, 1000)

K_A = 100
K_B = 100
q_p = 1/20
w = 20


q_A = []
q_B = []

TFA_values = []
TFB_values = []

TF(TFA_values)
Binding_Affinity(q_A, TFA_values, K_A)

TF(TFB_values)
TFB_values.reverse()
Binding_Affinity(q_B, TFB_values, K_B)

P =[]
for qb in q_B:
    for qa in q_A:
        P.append(GATE_OR(qa, qb, w, q_p))

delta_P = []
P_min = 1.00 #min(P)
for p in P:
    delta_P.append(p /P_min)

delta_P=np.reshape(delta_P,(np.size(TFA_values), np.size(TFB_values)))
```

```
plt.clf()
fig, fig_ax = plt.subplots(figsize=(10, 9))
df_cm = pd.DataFrame(delta_P)
fig_ax = sn.heatmap(df_cm, cmap = 'RdYlGn', annot=False, robust=True,
↪  yticklabels=TFB_values, xticklabels=TFA_values, cbar=True)
plt.title('OR GATE. P(i)/Pmin plot for w:' + str(w) +' K_A:' + str(K_A) + '
↪  K_B:' + str(K_B) + ' q_p:' + str(q_p) + ' and Pmin:' + str(P_min),
↪  loc="center")
plt.xlabel('TF_A in nM')
plt.ylabel('TF_B in nM')

plt.show()
plt.close()
```

A.5 Or Gate Function Fitting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 20 15:21:05 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
from scipy import signal


def GATE_OR(TFA,TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B

    Z_OFF  = (1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON   = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))

def GATE_OR_FIT(TF, a, b, c, d):
    (TFA, TFB) = TF

    return ( a*TFA + b*TFB + c*TFA*TFB + d)

def POLY(p, x, deg):
    y = 0
    for i in range(0,deg+1):
        y = y + p[i] * x**(deg-i)
    return y


K_A = 100
K_B = 100
q_p = 1/20
w1 = 0.04 #frequency
w2 = 0.02 #frequency
w = 20#cooperativity factor
deg = 3

time = []
time = np.arange(0, 1000, 1, dtype=None)
"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000

TFA_values2 = []
TFB_values2 = []
TFA_values2 = np.ones(1000)
TFA_values2[129:159]=1200
TFA_values2[589:629]=1200
TFB_values2 = np.ones(1000)
TFB_values2[389:429]=1200
TFB_values2[589:629]=1200

P1 = []

for i in time:

    P1.append(GATE_OR(TFA_values[i], TFB_values[i]))

plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('OR GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪ str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪ linewidth=3)
ax1.legend(loc="lower right")
ax1.set_ylabel('TFA Concentration (nM)')
ax1.set_yscale('linear')
```

```
ax2.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('TFB Concentration (nM)')
ax2.set_yscale('linear')


ax3.plot(time, P1,    c=sn.color_palette('husl')[2], label='P',
↪  linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()

[m, cov] = curvefit(GATE_OR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_OR_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_OR_FIT((x,y), m[0], m[1], m[2], m[3])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))

Ptrial =[]
Ptrialfit =[]
for i in time:
    x = TFA_values2[i]
    y = TFB_values2[i]

    Ptrial.append(GATE_OR(x, y))
    f  = GATE_OR_FIT((x,y), m[0], m[1], m[2], m[3])
    Ptrialfit.append(f)
print('R^2 score for P1:', r2_score(Ptrial, Ptrialfit))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('OR GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪  str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.legend(loc="lower right")
ax1.set_ylabel('Original Concentration (nM)')
ax1.set_yscale('linear')

ax2.plot(time, TFA_values2, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax2.plot(time, TFB_values2, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('Trial Concentration (nM)')
```

```
ax2.set_yscale('linear')


ax3.plot(time, Ptrial,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax3.plot(time, Ptrialfit, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()


z = np.polyfit(time, P1, deg)


Poly = []
for t in time:

    f = POLY(z, t, deg)
    Poly.append(f)

plt.clf()
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('OR GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪  str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
#ax2.plot(time, Poly, c=sn.color_palette('husl')[2], label='Polyfit',
↪  linewidth=3)

ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()
```

A.6 Or Gate Protein Generation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 27 00:37:27 2022

@author: irem
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 20 15:21:05 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
from scipy import signal


def GATE_OR(TFA,TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B

    Z_OFF = (1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)

    return ( (Z_ON) / (Z_ON + Z_OFF))

def GATE_OR_FIT(TF, a, b, c, d,e,f):
    (TFA, TFB) = TF

    return ( a*(TFA+b) + c*(TFB+d) + e*(TFA*TFB+f))

K_A = 100
K_B = 100
q_p = 1/20
w1 = 0.04 #frequency
w2 = 0.02 #frequency
w = 20#cooperativity factor

duration = 360 # dakika
time = []
time = np.arange(0, duration, 1, dtype=None)  # 40 dklik
"""impulse
TFA_values = []
```

```
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000

P1 = []

for i in time:
    p = GATE_OR(TFA_values[i], TFB_values[i])
    P1.append(p)

[m, cov] = curvefit(GATE_OR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_OR_FIT, (TFA_values, TFB_values), P1, p0=m)

TFA_values = []
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000
NewProt = []
NewProtTemp = 0


P1 = []
Pcurved = []
beta = 200000.0 # tunable parameter
alpha = 0.023104906018664842 #math.log(2)/30 timescale for 50% change in
↪  concentration of the stable protein is appr. 30 min and alpha_deg=0
Protss = beta/alpha;
ProtInt1 = 0
ProtInt2 = 0
timer1=0
timer2=0

for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_OR_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)

    p = GATE_OR(TFA_values[i], TFB_values[i])
    P1.append(p)
    if p>0.4:
        NewProtTemp = ProtInt1 + Protss*(1-math.exp(1)**(-alpha*timer1))
        ProtInt2 = NewProtTemp
        timer1 = timer1+1
        timer2 = 0
    else:
        NewProtTemp = ProtInt2*math.exp(1)**(-alpha*timer2)
        ProtInt1 = NewProtTemp
        timer2 = timer2+1
        timer1 = 0

    if NewProtTemp <0:
        NewProtTemp = 0
    NewProt.append(NewProtTemp)

print('R^2 score for P1:', r2_score(P1, Pcurved))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('OR GATE for w:' + str(w) +' K_A:' + str(K_A) + ' K_B:' +
↪  str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
#ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')


ax3.plot(time, NewProt, c=sn.color_palette('husl')[2], label='Protein',
↪  linewidth=3)
ax3.set_title('Protein generation', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Protein')
ax3.set_yscale('linear')
ax3.set_xlabel('Time(min)')

plt.show()
```

A.7 Xor Gate Single Promoter Heat Map

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 31 20:34:49 2022

@author: irem
"""

import matplotlib.pyplot as plt
import numpy as np
import math as math
import seaborn as sn
import pandas as pd


def TF(TF_values):

    for i in range(0,3):
        for j in np.arange(10**(i), 10**(i+1), 10**(i), dtype=float):
            TF_values.append(j)

    TF_values.append(float(10**(i+1)))


def GATE_XOR_SINGLE_PROMOTER(q_A1, q_B1, q_A2, q_B2, w, q_p):

    Z_OFF = (1 + q_A2 + q_B2 + w*q_A2*q_B2)*(1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)
    return ( (Z_ON) / (Z_ON + Z_OFF))



time = np.linspace(1, 200, 1000)


K_A = 200
K_B = 200

K_A2 = 900
K_B2 = 900

q_p = 1/10
w = 25


TFA_values = []
TFB_values = []

TF(TFA_values)

TF(TFB_values)
TFB_values.reverse()

P =[]
for y in TFB_values:
    q_B1 = y/K_B
    q_B2 = y/K_B2
    for x in TFA_values:
        q_A1 = x/K_A
        q_A2 = x/K_A2
        P.append(GATE_XOR_SINGLE_PROMOTER(q_A1, q_B1, q_A2, q_B2, w, q_p))

delta_P = []
P_min = 1.00 #min(P)
for p in P:
    delta_P.append(p /P_min)

delta_P=np.reshape(delta_P,(np.size(TFA_values), np.size(TFB_values)))
plt.clf()
fig, fig_ax = plt.subplots(figsize=(10, 9))
df_cm = pd.DataFrame(delta_P)
fig_ax = sn.heatmap(df_cm, cmap = 'RdYlGn', annot=False, robust=True,
↪  yticklabels=TFB_values, xticklabels=TFA_values, cbar=True)
plt.title('XOR GATE SINGLE PROMOTER. P(i)/Pmin plot for w:' + str(w) +'
↪  K_A:' + str(K_A) + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:'
↪  + str(K_B2) + ' q_p:' + str(q_p), loc="center")
plt.xlabel('TF_A in nM')
plt.ylabel('TF_B in nM')

plt.show()
plt.close()
```

A.8 Xor Gate Single Promoter Function Fitting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 20 16:54:32 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score

def GATE_XOR_SINGLE_PROMOTER(TFA, TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    q_A2 = TFA/K_A2
    q_B2 = TFB/K_B2

    Z_OFF = (1 + q_A2 + q_B2 + w*q_A2*q_B2)*(1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)
    return ( (Z_ON) / (Z_ON + Z_OFF))

def GATE_XOR_FIT(TF, a, b, c, d):
    (TFA, TFB) = TF

    return  ( a*TFA + b*TFB + c*TFA/TFB + d)

time = []
time = np.arange(0, 1000, 1, dtype=None)

K_A = 200
K_B = 200

K_A2 = 900
K_B2 = 900

q_p = 1/10
w = 25   #cooperativity factor
w1 = 0.04 #frequency
w2 = 0.02 #frequency

"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000

P1 = []
for t in time:

    P1.append(GATE_XOR_SINGLE_PROMOTER(TFA_values[t], TFB_values[t]))

[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))
```

```
plt.clf()
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE SINGLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪  q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,      c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)

ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()

TFA_values2 = []
TFB_values2 = []
TFA_values2 = np.ones(1000)
TFA_values2[129:159]=1200
TFA_values2[589:629]=1200
TFB_values2 = np.ones(1000)
TFB_values2[389:429]=1200
TFB_values2[589:629]=1200

Ptrial =[]
Ptrialfit =[]
for i in time:
    x = TFA_values2[i]
    y = TFB_values2[i]

    Ptrial.append(GATE_XOR_SINGLE_PROMOTER(x, y))
    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3])
    Ptrialfit.append(f)
print('R^2 score for P1:', r2_score(Ptrial, Ptrialfit))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE SINGLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪  q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
```

```
ax1.legend(loc="lower right")
ax1.set_ylabel('Original Concentration (nM)')
ax1.set_yscale('linear')

ax2.plot(time, TFA_values2, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax2.plot(time, TFB_values2, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('Trial Concentration (nM)')
ax2.set_yscale('linear')


ax3.plot(time, Ptrial,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax3.plot(time, Ptrialfit, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()
```

A.9 Xor Gate Single Promoter Protein Generation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat May 14 12:46:32 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
from scipy import signal


def GATE_XOR_SINGLE_PROMOTER(TFA, TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    q_A2 = TFA/K_A2
    q_B2 = TFB/K_B2

    Z_OFF = (1 + q_A2 + q_B2 + w*q_A2*q_B2)*(1 + q_A1 + q_B1 + q_A1*q_B1)
    Z_ON  = q_p*(1 + w*q_A1 + w*q_B1 + 2*w*q_A1*q_B1)
    return ( (Z_ON) / (Z_ON + Z_OFF))

def GATE_XOR_FIT(TF, a, b, c, d,e,f):
    (TFA, TFB) = TF

    return ( a*(TFA+b) + c*(TFB+d) + e*(TFA*TFB+f))


K_A = 200
K_B = 200

K_A2 = 900
K_B2 = 900

q_p = 1/10
w = 25   #cooperativity factor

ime = []
time = np.arange(0, 1000, 1, dtype=None)

"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000

P1 = []
for t in time:

    P1.append(GATE_XOR_SINGLE_PROMOTER(TFA_values[t], TFB_values[t]))

[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))

plt.clf()
```

```
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE SINGLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪   + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪   q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪   linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪   linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,      c=sn.color_palette('husl')[0], label='P',
↪   linewidth=3)
#ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪   linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()

duration = 360 # dakika
time = []
time = np.arange(0, duration, 1, dtype=None)  # 40 dklik

TFA_values = []
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000
NewProt = []
NewProtTemp = 0


P1 = []
Pcurved = []
beta = 200000.0 # tunable parameter
alpha = 0.023104906018664842 #math.log(2)/30 timescale for 50% change in
↪   concentration of the stable protein is appr. 30 min and alpha_deg=0
Protss = beta/alpha;
ProtInt1 = 0
ProtInt2 = 0
timer1=0
timer2=0

for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)
    p = GATE_XOR_SINGLE_PROMOTER(TFA_values[i], TFB_values[i])
    P1.append(p)
    if p>0.4:
        NewProtTemp = ProtInt1 + Protss*(1-math.exp(1)**(-alpha*timer1))
        ProtInt2 = NewProtTemp
        timer1 = timer1+1
        timer2 = 0
    else:
        NewProtTemp = ProtInt2*math.exp(1)**(-alpha*timer2)
        ProtInt1 = NewProtTemp
        timer2 = timer2+1
        timer1 = 0

    if NewProtTemp <0:
        NewProtTemp = 0
    NewProt.append(NewProtTemp)

print('R^2 score for P1:', r2_score(P1, Pcurved))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE_SINGLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪   + ' K_B:' + str(K_B) + ' q_p:' + str(q_p))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪   linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪   linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪   linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪   linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')


ax3.plot(time, NewProt, c=sn.color_palette('husl')[2], label='Protein',
↪   linewidth=3)
ax3.set_title('Protein generation', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Protein')
ax3.set_yscale('linear')
ax3.set_xlabel('Time(min)')

plt.show()
```

A.10 Xor Gate Double Promoter Heat Map

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 31 20:58:17 2022

@author: irem
"""


import matplotlib.pyplot as plt
import numpy as np
import math as math
import seaborn as sn
import pandas as pd


def TF(TF_values):

    for i in range(0,3):
        for j in np.arange(10**(i), 10**(i+1), 10**(i), dtype=float):
            TF_values.append(j)

    TF_values.append(float(10**(i+1)))


def GATE_XOR_DOUBLE_PROMOTER(q_A1, q_B1, q_A2, q_B2, w, q_p1, q_p2):

    Z_OFF1 = (1 + q_A1)*(1 + q_B1)
    Z_OFF2 = (1 + q_A2)*(1 + q_B2)
    Z_ON1  = q_p1*(1 + w*q_A1)
    Z_ON2  = q_p2*(1 + w*q_B2)

    return ( (Z_ON1*Z_OFF2 + Z_OFF1*Z_ON2 + Z_ON1*Z_ON2) / (Z_ON1*Z_OFF2 +
    ↪  Z_OFF1*Z_ON2 + Z_ON1*Z_ON2 + Z_OFF1*Z_OFF2))

time = np.linspace(1, 200, 1000)

K_A = 500
K_B = 100

K_A2 = 100
K_B2 = 500

q_p1 = 1/20
q_p2 = 1/20
w = 25


TFA_values = []
TFB_values = []

TF(TFA_values)

TF(TFB_values)
TFB_values.reverse()
```

```
P =[]
for y in TFB_values:
    q_B1 = y/K_B
    q_B2 = y/K_B2
    for x in TFA_values:
        q_A1 = x/K_A
        q_A2 = x/K_A2
        P.append(GATE_XOR_DOUBLE_PROMOTER(q_A1, q_B1, q_A2, q_B2, w, q_p1,
        ↪  q_p2))

delta_P = []
P_min = 1.00 #min(P)
for p in P:
    delta_P.append(p /P_min)

delta_P=np.reshape(delta_P,(np.size(TFA_values), np.size(TFB_values)))

plt.clf()

fig, fig_ax = plt.subplots(figsize=(10, 9))
df_cm = pd.DataFrame(delta_P)
fig_ax = sn.heatmap(df_cm, cmap = "RdYlGn", annot=False, robust=True,
↪  yticklabels=TFB_values, xticklabels=TFA_values, cbar=True)
plt.title('XOR GATE DOUBLE PROMOTER. P(i) plot for w:' + str(w) +' K_A:' +
↪  str(K_A) + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' +
↪  str(K_B2) + ' q_p1:' + str(q_p1) + ' q_p2:' + str(q_p2), loc="center")
plt.xlabel('TF_A in nM')
plt.ylabel('TF_B in nM')
plt.show()
```

A.11 Xor Gate Double Promoter Function Fitting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 20 17:05:45 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score


def GATE_XOR_DOUBLE_PROMOTER(TFA,TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    q_A2 = TFA/K_A2
    q_B2 = TFB/K_B2

    Z_OFF1 = (1 + q_A1)*(1 + q_B1)
    Z_OFF2 = (1 + q_A2)*(1 + q_B2)
    Z_ON1  = q_p1*(1 + w*q_A1)
    Z_ON2  = q_p2*(1 + w*q_B2)

    return ( (Z_ON1*Z_OFF2 + Z_OFF1*Z_ON2 + Z_ON1*Z_ON2) / (Z_ON1*Z_OFF2 +
    ↪  Z_OFF1*Z_ON2 + Z_ON1*Z_ON2 + Z_OFF1*Z_OFF2))

def GATE_XOR_FIT(TF, a, b, c, d):
    (TFA, TFB) = TF

    return ( a*TFA + b*TFB + c*TFA/TFB + d)

time = []
time = np.arange(0, 1000, 1, dtype=None)

K_A = 500
K_B = 100

K_A2 = 100
K_B2 = 500

q_p1 = 1/20
q_p2 = 1/20
w = 25   #cooperativity factor
w1 = 0.04 #frequency
w2 = 0.02 #frequency

"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000


P1 = []
for t in time:
    P1.append(GATE_XOR_DOUBLE_PROMOTER(TFA_values[t], TFB_values[t]))
```

```
[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))

plt.clf()
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE DOUBLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪  q_p1:' + str(q_p1) + ' q_p2:' + str(q_p2))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)

ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()


TFA_values2 = []
TFB_values2 = []
TFA_values2 = np.ones(1000)
TFA_values2[129:159]=1200
TFA_values2[589:629]=1200
TFB_values2 = np.ones(1000)
TFB_values2[389:429]=1200
TFB_values2[589:629]=1200

Ptrial =[]
Ptrialfit =[]
for i in time:
    x = TFA_values2[i]
    y = TFB_values2[i]

    Ptrial.append(GATE_XOR_DOUBLE_PROMOTER(x, y))
    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3])
    Ptrialfit.append(f)
```

```
print('R^2 score for P1:', r2_score(Ptrial, Ptrialfit))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE DOUBLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪  q_p1:' + str(q_p1) + ' q_p2:' + str(q_p2))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.legend(loc="lower right")
ax1.set_ylabel('Original Concentration (nM)')
ax1.set_yscale('linear')

ax2.plot(time, TFA_values2, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax2.plot(time, TFB_values2, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax2.legend(loc="lower right")
ax2.set_ylabel('Trial Concentration (nM)')
ax2.set_yscale('linear')


ax3.plot(time, Ptrial,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax3.plot(time, Ptrialfit, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax3.set_title('The degree of gene transcription', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Probability P')
ax3.set_yscale('linear')
ax3.set_xlabel('Time')
plt.show()
```

A.12 Xor Gate Double Promoter Protein Generation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat May 14 13:02:34 2022

@author: irem
"""

import numpy as np
import math  as math
import sympy as sym

import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from scipy.optimize import curve_fit as curvefit
from sklearn.metrics import r2_score
```

```python
from scipy import signal


def GATE_XOR_DOUBLE_PROMOTER(TFA,TFB):

    q_A1 = TFA/K_A
    q_B1 = TFB/K_B
    q_A2 = TFA/K_A2
    q_B2 = TFB/K_B2

    Z_OFF1 = (1 + q_A1)*(1 + q_B1)
    Z_OFF2 = (1 + q_A2)*(1 + q_B2)
    Z_ON1  = q_p1*(1 + w*q_A1)
    Z_ON2  = q_p2*(1 + w*q_B2)

    return ( (Z_ON1*Z_OFF2 + Z_OFF1*Z_ON2 + Z_ON1*Z_ON2) / (Z_ON1*Z_OFF2 +
    ↪  Z_OFF1*Z_ON2 + Z_ON1*Z_ON2 + Z_OFF1*Z_OFF2))

def GATE_XOR_FIT(TF, a, b, c, d,e,f):
    (TFA, TFB) = TF

    return ( a*(TFA+b) + c*(TFB+d) + e*(TFA*TFB+f))

time = []
time = np.arange(0, 1000, 1, dtype=None)

K_A = 500
K_B = 100

K_A2 = 100
K_B2 = 500

q_p1 = 1/20
q_p2 = 1/20
w = 25   #cooperativity factor


"""impulse
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89]=1000
TFA_values[489]=1000
TFB_values = np.ones(1000)
TFB_values[289]=1000
TFB_values[489]=1000
"""
"""step"""
TFA_values = []
TFB_values = []
TFA_values = np.ones(1000)
TFA_values[89:129]=1000
TFA_values[489:529]=1000
TFB_values = np.ones(1000)
TFB_values[289:329]=1000
TFB_values[489:529]=1000

P1 = []
```

```
for t in time:

    P1.append(GATE_XOR_DOUBLE_PROMOTER(TFA_values[t], TFB_values[t]))

[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=None)
[m, cov] = curvefit(GATE_XOR_FIT, (TFA_values, TFB_values), P1, p0=m)

Pcurved =[]
for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)
print('R^2 score for P1:', r2_score(P1, Pcurved))

plt.clf()
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE DOUBLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) +' K_A2:' + str(K_A2) + ' K_B2:' + str(K_B2) + '
↪  q_p1:' + str(q_p1)+ ' q_p2:' + str(q_p2))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,      c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
#ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')
ax2.set_xlabel('Time')
plt.show()

duration = 360 # dakika
time = []
time = np.arange(0, duration, 1, dtype=None)  # 40 dklik

TFA_values = []
TFB_values = []
TFA_values = np.ones(duration)
TFA_values[5:10]=1000
TFA_values[205:210]=1000
TFB_values = np.ones(duration)
TFB_values[105:110]=1000
TFB_values[205:210]=1000
NewProt = []
NewProtTemp = 0
```

```python
P1 = []
Pcurved = []
beta = 200000.0 # tunable parameter
alpha = 0.023104906018664842 #math.log(2)/30 timescale for 50% change in
↪  concentration of the stable protein is appr. 30 min and alpha_deg=0
Protss = beta/alpha;
ProtInt1 = 0
ProtInt2 = 0
timer1=0
timer2=0

for i in time:
    x = TFA_values[i]
    y = TFB_values[i]

    f  = GATE_XOR_FIT((x,y), m[0], m[1], m[2], m[3], m[4], m[5])
    Pcurved.append(f)

    p = GATE_XOR_DOUBLE_PROMOTER(TFA_values[i], TFB_values[i])
    P1.append(p)
    if p>0.4:
        NewProtTemp = ProtInt1 + Protss*(1-math.exp(1)**(-alpha*timer1))
        ProtInt2 = NewProtTemp
        timer1 = timer1+1
        timer2 = 0
    else:
        NewProtTemp = ProtInt2*math.exp(1)**(-alpha*timer2)
        ProtInt1 = NewProtTemp
        timer2 = timer2+1
        timer1 = 0

    if NewProtTemp <0:
        NewProtTemp = 0
    NewProt.append(NewProtTemp)

print('R^2 score for P1:', r2_score(P1, Pcurved))


plt.clf()
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 9))
fig.suptitle('XOR GATE_DOUBLE PROMOTER for w:' + str(w) +' K_A:' + str(K_A)
↪  + ' K_B:' + str(K_B) + ' q_p1:' + str(q_p1) + ' q_p2:' + str(q_p2))
ax1.plot(time, TFA_values, c=sn.color_palette('husl')[0], label='TFA',
↪  linewidth=3)
ax1.plot(time, TFB_values, c=sn.color_palette('husl')[1], label='TFB',
↪  linewidth=3)
ax1.set_title('Input signals', loc="center")
ax1.legend(loc="lower right")
ax1.set_ylabel('Concentrations(nM)')
ax1.set_yscale('linear')


ax2.plot(time, P1,    c=sn.color_palette('husl')[0], label='P',
↪  linewidth=3)
ax2.plot(time, Pcurved, c=sn.color_palette('husl')[1], label='P_fit',
↪  linewidth=3)
ax2.set_title('The degree of gene transcription', loc="center")
ax2.legend(loc="lower right")
```

```
ax2.set_ylabel('Probability P')
ax2.set_yscale('linear')


ax3.plot(time, NewProt, c=sn.color_palette('husl')[2], label='Protein',
↪  linewidth=3)
ax3.set_title('Protein generation', loc="center")
ax3.legend(loc="lower right")
ax3.set_ylabel('Protein')
ax3.set_yscale('linear')
ax3.set_xlabel('Time(min)')

plt.show()
```