VRIJE UNIVERSITEIT

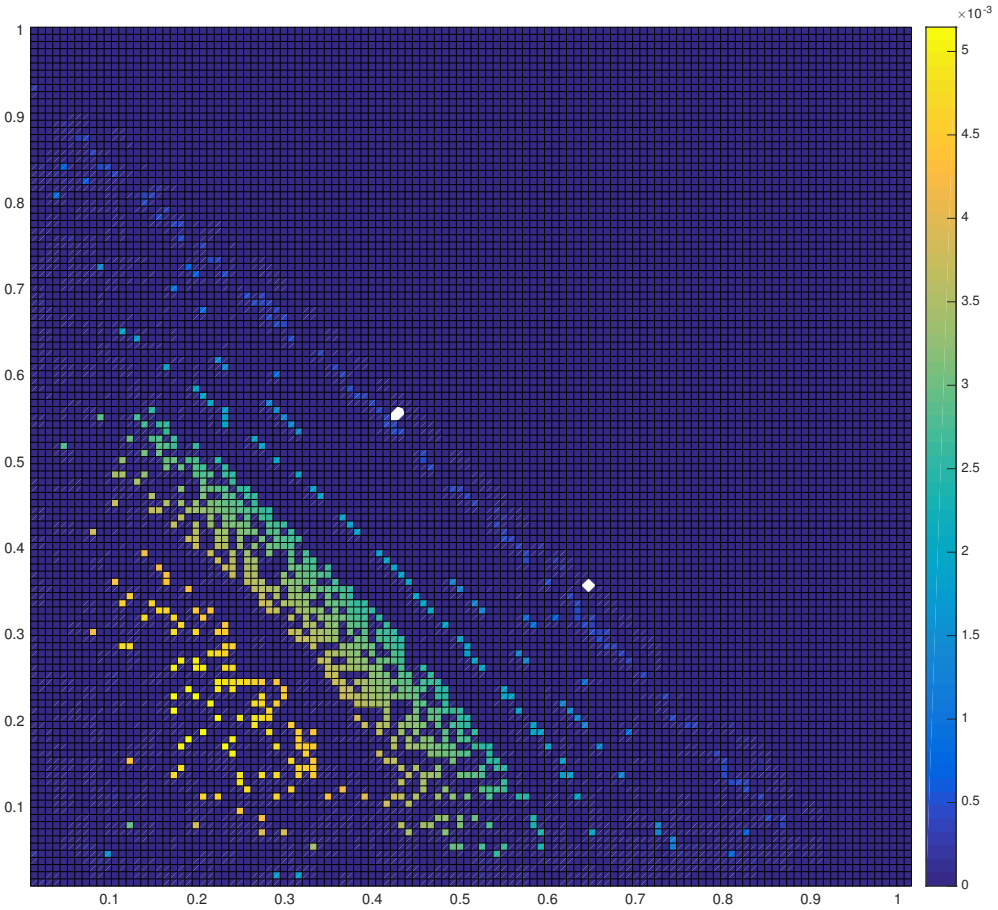
MASTER THESIS

Numerical Experiments On A Model For Bacteria Metabolism : Stress Response In The Presence Of Toxins

Author:
Martino Pitruzzella

Supervisor:
Bob Planqué

Second Examiner:
Jan Bouwe van den Berg



November 1, 2018

"While the telescope revealed deep simplicities in the cosmos, the microscope revealed previously unseen complexities in life. The same dichotomy between the simple and the complex has bedevilled the biological sciences ever since. Biologists, with some justification, argue that the life sciences are fundamentally harder than the physical sciences"

Ian Stewart

Abstract

Metabolism is the sum of chemical transformations that allow life to sustain itself. It can be described by a network of chemical reactions that happen inside the cells. In this thesis a model for the metabolism of bacteria is investigated numerically. Several examples of it are considered which include the presence of stress factors such as toxins. The problem is modeled as a set of differential equations describing the change of the concentrations of the chemical compounds present inside cell. The system is composed of $2n + 1$ differential equations, n metabolite variables and $n + 1$ enzyme variables. The system is supplied with one additional algebraic equation which specifies the growth rate which is equivalent to assuming constant osmotic pressure. The growth rate is optimised to find the optimal allocation of ribosome among the enzymes which gives rise to maximal growth. Several scenarios and specific systems are considered and metabolic adaptation is observed. This study shows the versatility and applicability of this model and that a growth rate optimisation approach also allows to understand microbial adaptation and processes that restrain growth such as stress management.

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Bob Planqué, for his patience, motivation and enthusiasm, which I had a chance to appreciate while working on this master project as well as during two of his courses I attended at VU university. I am glad I had a chance to know him and work on this research under his supervision. His research motivated me to study for a mater degree at VU university even before starting this master. Secondly, I would like to thank the master coordinator Corrie Quant for advice during the time of my master. Finally, I would like to thank my family for the support and love during the time of my studies, in particular my parents and my grandfather Aldo.

Contents

Chapter 1

Introduction

Biology is the natural science that studies life. It deals not only with the structure, function, development and reproduction of living organisms but also with their interaction with the environment around them. It is not easy to state a universal definition of life but there are some characteristics that are observed in all living beings.

They possess some kind of dynamic, physical and chemical organisation that unlikely would occur randomly and that is not found in inanimate objects. Although some inanimate things such as crystals often have some kind or regularity and structural organisation, they do not possess a dynamic kind of organisation: if regularity is broken it is not restored. To some extent regularity tends to be restored in living organisms.

Inanimate things also usually lack clear bounds. In contrast living organisms always posses some kind of clearly identifiable boundary. Moreover, living organisms not only possess internal organisation but most importantly posses homeostasis, a feature which allows them to maintain a stable state and organisation by means of biological processes such as signalling and feedback mechanisms.

Other important and characterising features of living organisms are their ability to grow, evolve, reproduce and adapt. All living organisms are composed of cells. Each cell contains genes which store genetic information. This allows the cells to carry out all the necessary life-sustaining processes including replication.

Living organisms can be divided into two subsets: unicellular and multicellular organisms. It is not fully understood how life first developed on earth but it is known that unicellular organisms have preceded by far multicellular organisms which have developed from them.

Microbes are unicellular organisms which have been discovered only recently after the invention of the microscope. They are at the basis of the food chain. They are eaten by bigger organisms and their metabolic waste products often contitute nutriment for plants. They are the first forms of life that have appeared on the planet and since then they have been ubiquitous. Wherever there is life microbes are also present. They not only outnumber macroscopic multicellular organisms in number but also form a comparable part of the total living mass on the planet.

Microbes can be either prokaryotes or eukaryotes. Eukaryotes contain a defined nucleus while prokaryotes do not. Unicellular microbes can be divided into protists, bacteria and archea. Protists are eukaryote organisms while bacteria and archea are prokaryotes. All multicellular organisms are composed of eucaryote cells. Prokaryotes are the most simple forms of life and also the first to have appeared on earth. Eukaryotes have then probably developed from prokaryotes by symbiosis (margulis, 1998). It is this more simple kind of organisms (prokaryote bacteria) that is modeled in this thesis.

Biology has made great progresses by means of a reductionistic approach: functioning of parts of the organism or of the cell are studied in detail. The different parts are studied separately from each other with the idea that the more complex whole can be described as a sum of the different parts. This idea has undoubtedly been successful to explain lots of phenomena but sometimes has also failed to explain more complex and global ones.

Only in recent times, and very recently with the advent of systems biology and bioinformatics, also a more holistic approach has been more widely used: the complex system needs to be approached as a whole in order to explain some of its global properties. The two approaches are not necessarily in contraposition. In the light of deterministic dynamical systems which showed how simple deterministic rules can lead to complex and unexpected phenomena such as emergence and pattern formation, it is possible that the considered system is indeed not more than the sum of its parts, but still not all its global features can be described by only studying its composing parts separately. Progress in this direction have also been possible due to the availability of computers.

We will now briefly describe some general biological facts about bacteria metabolism that are relevant to this model. Bacteria are unicellular organisms which are virtually found in any environment where there is presence of water. They are confined by a cell membrane. The cell membrane is constituted usually by a double layer of lipids and some transport proteins that allow the passage only of selected nutrients from the exterior. The interior of the bacteria contains the cytoplasm, a highly concentrated liquid composed of water in which all the chemical compounds that are used for its functions and growth are diluted. Inside the cell membrane most of the compounds wander randomly and react with each other only if they run into the ones the can react with and at the angle that allow them to magnetically interact and bind. Although it might at first it might be surprising that the reactions happen randomly, they occur regularly given their presence in high concentrations. The nucleoid is present In the center of the cell. The nucleoid contains one or more chromosomes. Chromosomes contain the information that allows the cell to built proteins and all the macromolecules needed for growth. The genetic information present in the nucleotid control the expression of proteins that catalyse metabolic reactions. As more nutrients are imported from outside the cell membrane and are converted in precursor metabolites and later into macromolecules, the pressure inside the cell rises which leads to the intake of water and eventually allows the cell to grow. While the cell grows the cell membrane is extended and the nucleoid containing the chromosome(s) is duplicated. The end result is the splitting of the cell in two nearly identical cells which contain the same genetic information. An important role in the whole process is the one of enzymes and of the the ribosome. Enzymes act as catalysers for specific reactions. Ribosomes are complex molecules which contain RNA information aswell as protein parts, it catalyses the production of proteins including enzymes and of itself. Image 1.1 shows a simplified version of the prokaryote cell structure. In the presence of enzymes reactions can speed up by a factor of several orders of magnitude. The cell implements regulatory mechanisms which allow to choose the amount of ribosome which is used to catalyse the production of each enzyme. In this way the cell regulates genes expression which in turn depends also on the availability of external nutrients. An average bacterial cell contains about 500 different metabolites in its interior and a much greater number of reactions is possible. However not all the reactions need to take place at the same time. Which reaction subnetworks actually are expressed depends on which nutrients are available. We will see that in the presented model, even if we restrict to a system with only few metabolite variables, we will be able to observe relatively complex behaviour like adaptation and stress response.

The quick growth of bacteria is an evolutionary strategy. When a minimum sufficient set of nutrients for growth is available bacteria always maximise their growth rate. Most of their metabolism is addressed towards growth. This is because bacteria usually find themselves competing for the nutrients with other microorganisms. Being able to grow quicker than other competitors makes the difference for survival. This has lead to evolutionary preference for quickly growing strains of bacteria. When the needed nutrients are available, after a short settling stage, bacteria optimise their growth rate and grow exponentially quick until the sources are depleted. This is well documented both in vitro and in nature [2].

In this master thesis a model of bacteria metabolism is investigated numerically. The model is derived in a top-down manner: the whole cell is modeled based on first principles. The model is kept as simple as possible but specified enough so that feedback mechanisms such as gene regulation and expression are included. Metabolism is the sum of all the chemical reactions that

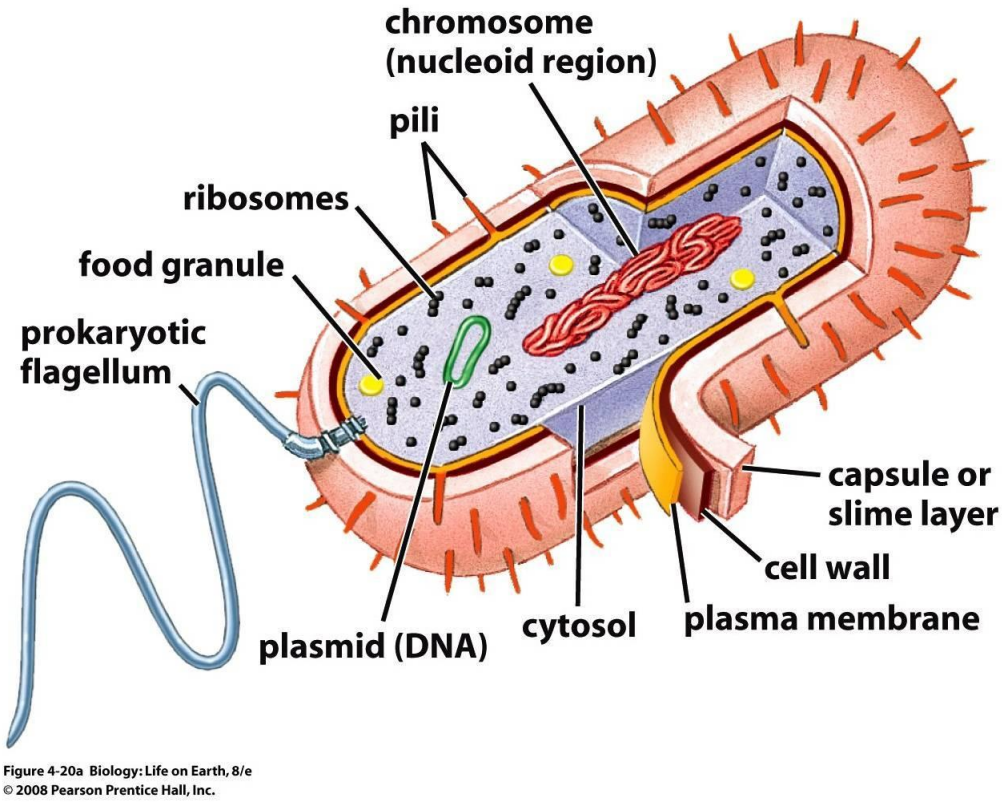


FIGURE 1.1: This diagram shows the simplified version of the structure of a prokaryote cell

happen inside the cell. It can be described by a network of reactions that can occur inside the cell membrane. The presented model consists of a set of differential equations that describe the change of the concentrations of the compounds which are present inside the cell. Also, the system of differential equations is supplied with one algebraic equation with describes the growth rate and hence allows its maximisation.

In the model that we present we are interested in the exponential growth stage of bacterial growth. Hopefully this research will give theoretical insight not only on how to choose the best substrate concentrations which allow bacteria to grow as fast as possible in the applications, but also on how the bacteria are able to successfully deal with growth maximisation themselves. Possible biotechnology applications include the ones where bacteria are selected or driven to produce some specific compound or to fulfil some specific task. Insight on how bacteria deal with growth optimisation can be useful in applications where it is needed that bacteria grow as fast as possible for example the production of yoghourts or the purification of waters and oils.

Previous mathematical studies of bacteria [5], [6], [7], [9], [10], have showed that cell implements feedback mechanisms in order to face limited resources. Even in a growth permissive environment cells have finite resources available so they have to implement strategies that allow them to fine tune how these resources are used. Cells need then to possess mechanisms that allow them to quickly change gene expression and ribosome allocation according to changing external conditions. For example if one substrate runs out, the enzyme that metabolises it does not need to be produced anymore. Possibly other enzymes that catalyses other substrates present in the environment that was not used before needs to start to be sythesised. In another scenario, a toxin gets introduced in the environment and the cell needs to start producing the toxin-removing enzyme, at the expense of the production of other enzymes. Previous mathematical models have found out that in the light of the tradeoffs and economy of resources that the cell needs to implement, metabolic networks can be decomposed into minimal subnetworks called elementary flux modes (EFM's) [7]. Every flux profile which maximise the flux through the entire network can be decomposed into a convex sum of these minimal networks. As we have seen, not all the reactions in the network have to happen at the same time, some fluxes can be zero at some point in time.

For example if some substrate nutrient is not present, the flux of the reaction that metabolises that particular substrate will be zero. EFM's are minimal because they can not be decomposed into smaller subnetworks by discarding some reaction. They are the minimal networks that allow steady state solutions. EFMs are often used to maximise biomass synthesis flux. The model that we present introduces an additional algebraic equation which defines the growth rate which is equivalent to assuming constant osmotic pressure. This additional equation which specifies the growth in terms of the metabolites' molecular volume allows its maximisation. In the setting that this model proposes maximal growth rate is also attained in an equivalent concept of EFM's. We will call these elementary self-replicating networks Elementary Growth Modes (EGM's).

The main goal of this thesis is to undestand if it is possible to understand stress management of cells from a fitness perspective. The usual approach is that cells have to choose either to invest resources in their growth or in their stress management. Stresses such as toxin, radiation and temperature prevent growth and hence are considered 'orthogonal' to growth. For this reason models of stress management and models of growth are usually been developed separately. In this thesis we wish to show that a growth rate optimization framework can be used to explain stress management. Both these aspects of bacterial life are studied in one unified theory where the cell uses gene expression to change dynamically the allocation of its recources according to varying environmental conditions.

This thesis is composed of 6 sections. In section 2 we introduce the model in which the growth rate is to be maximised. Maximizing the growth rate leads to minimal self-replicating networks, EGMs. For a given EGM, the identity of the reactions is known. Within one EGM, the metabolite and enzyme concentrations are tuned to find the maximal growth rate. In section 3 we maximise the growth rate numerically within one EGM. We are interested to see if this theory can also be applied to stress management systems. In this section we describe the numerical problem that we wish to study and present four different examples. The first one does not involve the presence of toxins while the following three model the presence of a toxin in three different ways. In section 4 we present the results relative to the examples introduced in section 3 and some background tests that have been performed to check the functionality of the programs. In section 5 we describe the Matlab codes that have been developed in order to numerically investigate the examples. In section 6 we conclude and describe what we have learned from this study and recount what this model was able to predict.

Chapter 2

EGM theory: Derivation Of The Model Equations

In this chapter, after having explored the relevant biological background in the previous one, we derive the main model that we will study and investigate. Several examples of this model will be investigated numerically in the following chapters. This section is based on the paper "Metabolism and Growth Rate in Microbiology" (Planqué and Bruggerman, 2017) [1]. Before deriving the model from general principles we make a summary of the content of this chapter and of its sections.

In section 2.1 we consider some definitions and first principles and use them to derive the modeling equations of our problem. The model is composed of a system of differential equations and an algebraic equation which specifies the growth rate and allows its maximisation. In section 2.2 we further specify the model (we will call it the whole cell model) by making a distinction between metabolites and enzymes. We do this because we want the system to be able to model mechanisms which involve the allocation of the ribosome among the different enzymes. In the whole cell model the number of metabolites and enzymes is in general different, however, in the examples that we will introduce in the next chapter we consider only systems where the number of metabolites equals the number of enzymes. This is because we will consider only systems composed of only one Elementary Growth Mode or EGM. We will define EGM's as minimal self-replicating networks which can not be reduced to smaller networks by discarding reactions. We will see that these minimal networks share the property that the number of metabolites equals the number of enzymes.

We wish to solve the maximisation problem of finding the metabolite and enzyme concentrations which are such that the system attains maximum growth. We know that cells attain maximum growth when they are in balanced exponential growth where all copy number of the compounds grow at the same rate. For this reason in section 2.3 we derive a set of algebraic equations, we will call them the Balanced Growth Equations, that need to hold when the system is in steady state. Hence by requiring that the system is in steady state, keeping in mind that we want to maximise the growth rate, we derive a static set of algebraic equations from the set of differential equations which describe a dynamical system. The resulting set of equations (the Balanced Growth Equations) will be non-linear in the concentrations variables and in the growth rate and hence difficult to tackle.

We then will proceed in steps and 'decompose' the maximisation problem into non-linear and linear parts. First we fix the metabolite concentrations, then we will treat the growth rate which appears in the denominator differently from the one that appears in the numberator in the equations. This will allow to transform the problem into a standard linear programming problem. We will then use a fixed-point argument to find the maximum growth rate for fixed metabolites concentrations. Lastly, we will maximise over the metabolites concentrations to find the maximum growth rate of the problem. In section 2.5 we derive another set of algebraic equations, the Optimality Equations. This set of equations characterize critical points in the space of solutions of the Balanced Growth Equations (for fixed metabolites concentrations) and is derived by imposing that the a vector tangent to the solution set of the Balanced Growth Equations must be normal to the gradients of the functions in the Balanced Growth Equations. We will use this set of equations

in two ways, firstly to check that the point we have found with the maximisation procedure is indeed a critical point and hence a candidate to be the point where maximum growth rate is attained (if the point has such property it must be a solution of the Optimality Equations). Secondly we will use the Optimality Equations to continue the solution respect to one external concentration. We can not use only the Balanced Growth Equations to do this continuation step. The algebraic system which includes only the Balanced Growth Equations is underdetermined and hence will not allow to do so. We will instead continue the solution in the space of both Balanced Growth and Optimality Equations. This will allow to find a one-parameter curve of solutions parametrized by one external concentration. It is important to notice that to derive the Optimality Equations we make the assumption that for fixed metabolites concentrations the Balanced Growth Equations have a unique local solution. Althougth this seems reasonable it has not been proved and we take it as an assumption.

In section 2.6 we introduce the μORAC framework. We know that bacteria are able to change quickly ribosome allocation and enzymes expression according to changes in the external concentrations. The μORAC setting attempts to model this ability of bacteria to adapt to changing external conditions. We assume that the cell uses some internal metabolites as sensor metabolites. Also we assume that it possesses some feedback mechanism that allows it to use the internal sensor information (the concentration of the sensor metabolites) to predict external evironmental changes and quickly change gene espression accordingly. Specifically we will supply the differential equations system with one or more functions, which we have calculated using the continuation, and which allow to supply the system with predicted optimal ribosomial fractions. We assume that cells have gene regulatory networks that allow to make such predictions. The controlling circuit (which allows to estimate optimal ribosomial fractions using sensor metabolites) does not use any external concentrations as known parameters but only internal ones. We will then run the system of differential equations and continously change the ribosome allocation according to the predictions offered by the sensor metabolites concentrations. We will see that in this setting the system is then able to steer itself to the steady state of maximum growth and steer itself to a new maximum after the external substrate concentrations are changed. All this will be further explained in the respective sections.

2.1 General Model Derivation

We are now ready to derive the model based on first principles. We start by considering the following definitions [2]:

- Steady state growth: when all the intensive properties of the cell, such as relative volume increase and concentrations, are time-independent.
- Balanced growth: when all extensive properties of the cell, such as copy number of compounds and volume, increase by the same factor over a given time interval.
- Exponential growth: when the multiplication factor (of the number of cells) is time independent. Steady state growth is equivalent to exponential balanced growth.

We can recall these modeling assumptions and observations:

- Growth of bacteria is defined as an increase of the bacterial biomass. This happens when selected nutrients (or in general any chemical compound including water) cross the cell membrane and reach the cell interior.
- In order to sustain growth, cells need to make the enzymes which in turn catalyse the reactions that import and transform the nutrients into all the needed organic compounds of which the cell is composed. This means that the reaction network has to have some degree of self-consistency: if some reaction is present and catalysed by one enzyme, the reaction which creates that specific enzyme has to be present further downstream in the reaction network. This is because, differently from smaller molecules like aminoacids, macromolecules

such as proteins and in particular enzymes can not be imported from the cell exterior and have to be metabolised internally from smaller molecules.

- Dilution by growth has to be taken into account: as the chemical compounds are transported inside the cell, the osmotic pressure inside the cell grows. This leads to the intake of water by osmosis. As a results the cell grows. On the other hand this means that as the cell grows, the concentrations are diluted and the reaction rates drop. This observation shows that dilution by growth needs to be included in the model. Cells therefore need to keep making the enzymes needed to maintain reaction rates constant.
- Lastly, since we are interested in the maximisation of the growth rate, we assume that the cell is in exponential balanced growth. Which we will usually refer to simply as balanced growth.

In what follows we consider the growth of an individual cell rather than a population. Our model is one of average cells in a population. We make this assumption because balanced growth is the only state in which extensive properties of the cell such as composition and growth rate are well-defined. Cells populations converge to balanced growth when conditions allow growth, this is the only well-defined time-invariant state in microbial growth.

After having stated some of the modeling assumptions we can further proceed in the derivation of the model based on these assumptions. We will at first derive a model where we do not make distinction between metabolites, enzymes and ribosome. We will introduce this distinction later when we further specify the model in the next section. We consider the vector $n = (n_1, \dots, n_N)$ of copy number of N compounds in a cell volume V. The concentrations inside the cell are then defined as: $c_k = n_k/V$, the concentration of compound k. We assume that the change in copy numbers of the compounds inside the cell change only according to the reactions that import and export molecules from and to the exterior of the cell and to the reactions that occur inside the cell. Furthermore we assume that the compounds can be consumed when they take part in some reaction (when they come to form other compounds or are transformed into some form of energy) but do not degrade due to other reasons.

Since the concentrations change only due to the reactions that occur inside the cell we assume that

$$\dot{n}_k = V \sum_j N_{kj} v_j(\boldsymbol{c}) \quad \forall k$$

where N is the stoichiometry matrix and $v_j(\boldsymbol{c})$ is the j-th reaction rate, which is assumed to be a function of the concentrations inside the cell. It is reasonable to assume that the volume V is a function of the compounds: $V = V(\boldsymbol{n})$. We choose to specify this function as

$$V(\boldsymbol{n}) = \sum_k \rho_k n_k$$

where the ρ_l are the osmotic activities of the compounds (the molar volumes). So the volume is a sum of the copy numbers of each compound multiplied with the corresponding osmotic ativity (hence multiplied by the volume each compound occupies in the cell). We make this assumption as it seems to be the most reasonable physically. Furthermore, we define the instantaneous growth rate as

$$\mu(t) := \frac{\dot{V}}{V}$$

it is defined as the rate of change of volume per volume unit (so it is defined similarly to the istantaneous speed in physics). Now, since we defined the concentrations as $c_k = n_k/V$, we have that $n_k = V c_k \quad \forall k$. Hence, if we calculate the derivative $\dot{n}_k$ we have that

$$\dot{n}_k = \frac{d}{dt}(Vc_k) = \dot{V}c_k + V\dot{c}_k = V\sum_j N_{kj}v_j(\boldsymbol{c}) \quad \forall k$$

Using the last two terms of the previous chain of equalities we obtain the first main equation of the model

$$\dot{c}_k = \sum_j N_{kj}v_j(\boldsymbol{c}) - \frac{\dot{V}}{V}c_k \quad \forall k$$

which, using the definition of the istantaneous growth rate can be rewritten as

$$\dot{c}_k = \sum_j N_{kj}v_j(\boldsymbol{c}) - \mu c_k \quad \forall k \tag{2.1}$$

and in vector notation reads as

$$\dot{\boldsymbol{c}} = N\boldsymbol{v}(\boldsymbol{c}) - \mu\boldsymbol{c}.$$

We notice that the concentration of each compound is diluted by volume growth (this is accounted by the last term). We derive now the second main equation of the model, a closed form for the growth rate which will allow later its maximization. Using the formula for the volume $V(\boldsymbol{n}) = \sum_l \rho_l n_l$ and dividing by V, we can easily see that

$$1 = \sum_k \rho_k c_k \tag{2.2}$$

this means that if we take the derivative respect to time and substitute the formula in equation (2.1) we get

$$\begin{aligned}
0 &= \frac{d}{dt}\left(\sum_k \rho_k c_k\right) \\
&= \sum_k \rho_k \dot{c}_k \\
&= \sum_k \rho_k \left(\sum_j N_{kj}v_j(\boldsymbol{c}) - \mu c_k\right) \\
&= \sum_k \rho_k \sum_j N_{kj}v_j(\boldsymbol{c}) - \sum_k \rho_k(\mu c_k) \\
&= \sum_k \rho_k \sum_j N_{kj}v_j(\boldsymbol{c}) - \mu\sum_k \rho_k c_k \\
&= \sum_k \rho_k \sum_j N_{kj}v_j(\boldsymbol{c}) - \mu
\end{aligned}$$

where in the last but one step we have used equation (2.2) . Hence we have derived the second main equation of the model. A closed form for the growth rate

$$\mu = \sum_k \rho_k \sum_j N_{kj} v_j(\boldsymbol{c}). \tag{2.3}$$

Equation (2.3) specifies the growth rate in terms of the metabolic reactions that happen inside the cell. It is a weighted sum of all the chemical reactions, each weighted according the molecules' molar volumes. Equations (2.1) and (2.3) together are the main modeling equations. We notice that, at this point, equation (2.1) does not allow any further analysis, it is too general.

Since we know that cells attain maximum growth when they are in (exponential) balanced growth, we now attempt to impose that the system is in (exponential) balanced growth. Hence we impose that the system is in steady state. We start from the general relation for the concentrations (2.1)

$$\dot{c}_k = \sum_j N_{kj} v_j(\boldsymbol{c}) - \mu c_k \quad \forall k$$

by imposing that the system is in steady state, hence by imposing $\dot{c}_k = 0$ we get the equation for μ

$$\mu = \frac{\sum_j N_{kj} v_j(\boldsymbol{c})}{c_k}.$$

We now combine this last equation with the other equation for μ that we have previously found, equation (2.3)

$$\mu = \sum_k \rho_k \sum_j N_{kj} v_j(\boldsymbol{c})$$

so we get

$$\sum_k \rho_k \sum_j N_{kj} v_j(\boldsymbol{c}) = \frac{\sum_j N_{kj} v_j(\boldsymbol{c})}{c_k}.$$

This leads to the following closed identity that involves the concentrations

$$\sum_j \left(N_{kj} - c_k \sum_k \rho_k N_{kj}\right) v_j(\boldsymbol{c}) = 0 \quad \forall k.$$

This is a non-linear set of equations in the concentrations that needs to hold in steady state in the general case. We notice that this set of equations has too little structure to work with, solving for the concentrations using this equation is not an easy task so this set of equations is not useful in this form. In order to be able to find balanced growth solutions, and in particular the one which has maximum growth rate among these, we need to solve the equations for both the concentrations and the fluxes at the same time. This is in contrast with EFM models where, since the growth rate is not considered, it is possible to first solve the equations for the fluxes and then for the concentrations.

In the next section we will hence specify the system further and introduce more biological structure. This will both make the equations more manageable and solvable and make the model more realistic and able to account for regulatory mechanisms. In particular we will make a distinction between metabolites, enzymes and ribosomes. We will call the resulting system the Whole Cell Model.

2.2 Derivation of the Whole Cell Model

After having derived the main general equations for the model (2.1) and (2.3), we further specify it by making a distinction between metabolites, enzymes and ribosomes. We make this distinction because we want the model to be specified enough so that mechanisms which involve how the ribosome is allocated among the enzymes can be observed. We assume that to each reaction is associated one and only one enzyme. The ribosome is associated to the synthesis of each of the enzymes and of itself. So we have now that the vector of concentrations is $\boldsymbol{c} = (\boldsymbol{x}, \boldsymbol{e}, r)$, where $\boldsymbol{x}$ is a vector of length m and $\boldsymbol{e}$ is a vector of length n (so in total we have $n + m + 1$ concentations). We make also a distinction between the $\boldsymbol{\rho}$ and the $\boldsymbol{\sigma}$, respectively the osmotic activities of the metabolites and of the the enzymes and σ_r the osmotic activity of the ribosome (these were previously all called $\boldsymbol{\rho}$). Finally, we also assume that there are more reactions than metabolites (in order to keep the network connected).

The stoichiometry matrix N is now divided into blocks. The P block is the metabolites reactions block while the M block is relative to the enzyme synthesis reactions

$$N = \left[\begin{array}{c|c} P & -M \\ \hline 0 & I \end{array}\right]$$

P is an $m \times n$ matrix with $m < n$ with both positive and negative entries, M is an $m \times (n+1)$ matrix with positive or zero entries and I is a $(n+1) \times (n+1)$ identity matrix. The P block has entries of different signs because it accounts for reactions which use metabolites to produce other metabolites, the M block has a minus sign in front because metabolites are consumed to produce the enzymes.

The vector of reaction rates is now specified as:

$$\vec{v} = (v_1, \ldots, v_{2n+1}) = (v_1, \ldots, v_n, w_1, \ldots, w_n, w_r)$$

where the $(v_1, \ldots, v_n)$ are the metabolic reaction rates, $(w_1, \ldots, w_n)$ are the enzyme synthesis rates and w_r is the ribosome synthesis rate (we will then set $v_{n+j} = w_j$ and use subscript r in the place of $n+1$ for the w reaction rates).

Since reaction j is catalyzed by enzyme j we have that

$$v_j = e_j f_j(\boldsymbol{x})$$

where the functions $f_j(\boldsymbol{x})$ depend on the metabolites concentrations. Also, for the enzymes synthesis we have

$$w_j = r\alpha_j g_j(\boldsymbol{x})$$

where the functions $g_j(\boldsymbol{x})$ are such that $g_j(\boldsymbol{x}) \geq 0$ and dependent on the metabolites concentrations while the α_j are the fractions of ribosome allocated to catalyse the production of enzyme j. The e_j and r appear as a product to model the catalysing property of the enzymes and ribosomes [11]. Also, their linear dependence introduces the linear structure in the model that we will exploit later when we will derive the Balanced Growth and Optimality Equations.

The properties of the enzymes and of the ribosome dictate the form of the kinetics functions $f_j(\boldsymbol{x})$ and $g_j(\boldsymbol{x})$. In this thesis we consider functions approximately proportional to the metabolites involved but saturating for high concentrations of the metabolites (in particular we consider the standard reversible Michaelis-Menten form for these functions).

The complete whole cell model can now be stated as

$$\begin{cases} \dot{x}_k = \sum_{j=1}^{n} P_{kj} e_j f_j(\boldsymbol{x}) - \sum_{j=1}^{n+1} M_{kj} r \alpha_j g_j(\boldsymbol{x}) - \mu x_k \quad , \quad \forall k = 1, \ldots, m \\ \dot{e}_j = r \alpha_j g_j(\boldsymbol{x}) - \mu e_j \qquad\qquad , \qquad \forall j = 1, \ldots, n \\ \dot{r} = r \alpha_{n+1} g_{n+1}(\boldsymbol{x}) - \mu r \\ \mu = \sum_{k=1}^{m} \rho_k \sum_{j=1}^{n} P_{kj} v_j + \sum_{j=1}^{n+1} \left(\sigma_j - \sum_{j=1}^{m} \rho_k M_{kj} \right) w_j \\ \alpha_1 + \ldots + \alpha_n + \alpha_{n+1} = 1 \end{cases} \tag{2.4}$$

where the first $n + m + 1$ are differential equations, the last but one equation specifies the growth rate and the last one states that all the ribosome is occupied all the time to produce the enzymes. The α_j in the equation

$$\alpha_1 + \ldots + \alpha_n + \alpha_{n+1} = 1 \tag{2.5}$$

are the fractions of the ribosome that is used to synthesise each enzyme. The terms $-\mu x_k$, $-\mu e_j$ and $-\mu r$ account for the dilution by growth. As previously mentioned, we have made a distinction between the $\boldsymbol{\rho}$, the $\boldsymbol{\sigma}$ and σ_r. These are the molar volumes of the metabolites, enzymes and ribosome respectively.

2.3 Deriving the Balanced Growth Equations

In balanced growth the dynamical system is in steady state. We wish to find the steady state with maximal growth rate, since cells seem able to find such maxima experimentally [12,13,14,15,16,17]. In this section we derive a set of equations that need to hold when the system is in steady state for the Whole Cell Model, we will call these the Balanced Growth Equations. At the end of section 2.1 we have imposed that the system is in steady state for the general model. As we have seen this has lead to closed form equations that do not allow much analysis. We will see now that after having introduced the biology in the whole cell model, this derivation will allow to simplify the resulting equations. However, this is non-trivial because of the definition of μ in equation (2.3). By imposing that the system is in steady state (hence by imposing that the left hand side of the differential equations in the system definition (2.4) equals zero), we wish to derive a simplified algebraic set of equations which depends only on the metabolite concentrations $\boldsymbol{x}$ and on the growth rate μ and not anymore on the enzyme $\boldsymbol{e}$ and ribosome r concentrations. If we manage to obtain a sytem that is linear we will then be able to use the techniques from linear programming to maximise the growth rate. We will find out that the resulting set of equations will be linear in all the variables except the growth rate μ. This will allow to split the algebraic system into a linear and a non-linear part. The linear optimization problem is then solved using linear programming taking care of the non-linear part using a fixed-point technique which is reported in section 2.4.

We are now ready to derive the Balanced Growth Equations for the Whole Cell Model. We will see that it is indeed possible to simplify the set of equations that we will find and get rid of some of the variables (the $\boldsymbol{x}$ metabolite concentrations and the r). This will make the system of algebraic equations more manageable and easy to solve.

In what follows, in order to make the notation more concise, we introduce the constants a_j, b_j and the functions $\beta_j(\boldsymbol{x})$.

$$a_j = \sum_{k=1}^{m} \rho_k P_{kj}$$

$$b_j = \sum_{k=1}^{m} \rho_k M_{kj}$$

$$\beta_j(\boldsymbol{x}) = \alpha_j g_j(\boldsymbol{x}).$$

We will see that this notation will allow to separate the $f_j(\boldsymbol{x})$ and $g_j(\boldsymbol{x})$ better in the resulting equations. Notice that we have $\alpha_j \geq 0 \forall j$ as the α_j are fractions that express how the ribosome is allocated and sum up to 1. Since we have also that $g_j(\boldsymbol{x}) \geq 0$, ($g_j(\boldsymbol{x}) < 0$ is biologically impossible), we have also that the $\beta_j(\boldsymbol{x}) \geq 0$. Now, since we assume the system is in steady state we have that $\dot{\boldsymbol{c}} = 0$, in particular we have for the ribosome that

$$\dot{r} = 0$$

which implies

$$\mu = \beta_r(\boldsymbol{x})$$

and for the enzymes

$$\dot{e}_j = 0$$

which implies

$$\mu e_j = r\beta_j(\boldsymbol{x})$$

so we have

$$e_j = \frac{r\beta_j(\boldsymbol{x})}{\mu}.$$

Using the last identities and the introduced notation we have also then that

$$v_j = e_j f_j(\boldsymbol{x}) = \frac{r\beta_j}{\mu} f_j(\boldsymbol{x})$$

and

$$w_j = r\alpha_j g_j(\boldsymbol{x}) = r\beta_j(\boldsymbol{x}).$$

Consider now the equation for the growth rate in the Whole Cell Model which explicitly reads as

$$\mu = \sum_{k=1}^{m} \rho_k \sum_{j=1}^{n} P_{kj} v_j + \sum_{j=1}^{n+1} \left(\sigma_j - \sum_{k=1}^{m} \rho_k M_{kj} \right) w_j.$$

We can rewrite this equation as

$$\mu = \sum_{k=1}^{m} \rho_k \sum_{j=1}^{n} P_{kj} \frac{r\beta_j(\boldsymbol{x})}{\mu} f_j(\boldsymbol{x}) + \sum_{j=1}^{n+1} \sigma_j r\beta_j(\boldsymbol{x}) - \sum_{j=1}^{n+1} r\beta_j(\boldsymbol{x}) \sum_{j=1}^{m} \rho_k M_{kj}$$

then, dividing by r we get

$$\frac{\mu}{r} = \sum_{k=1}^{m} \rho_k \sum_{j=1}^{n} P_{kj} \frac{\beta_j(\boldsymbol{x})}{\mu} f_j(\boldsymbol{x}) + \sum_{j=1}^{n+1} \sigma_j \beta_j(\boldsymbol{x}) - \sum_{j=1}^{n+1} \beta_j(\boldsymbol{x}) \sum_{j=1}^{m} \rho_k M_{kj}$$

which, using the notation introduced above, reads as

$$\frac{\mu}{r} = \sum_{j=1}^{n} \left(\frac{a_j f_j(\boldsymbol{x})}{\mu} + \sigma_j - b_j \right) \beta_j(\boldsymbol{x}) + (\sigma_r - b_r)\mu. \tag{2.6}$$

We will use this equation later in this section. We have previously imposed that the derivative of the ribosome concentration and of the enzymes concentrations is zero (we impose that the system is in steady state). We now do the same for the first m equations in the system of differential equations, those relative to the metabolites. So we set $\dot{x}_k = 0$ and in the model equations and we get the m equations

$$\mu x_k = \sum_{k=1}^{n} P_{kj} e_j f_j(\boldsymbol{x}) - \sum_{k=1}^{n+1} M_{kj} r\alpha_j g_j(\boldsymbol{x})$$

which, using the new notation for the a_j, b_j and for the $\beta_j(\boldsymbol{x})$, reads as

$$\mu x_k = \sum_{k=1}^{n} P_{kj} \frac{r\beta_j(\boldsymbol{x})}{\mu} f_j(\boldsymbol{x}) - \sum_{k=1}^{n+1} M_{kj} r\alpha_j g_j(\boldsymbol{x}).$$

Dividing by r we get

$$\frac{\mu}{r} x_k = \sum_{k=1}^{n} P_{kj} \frac{\beta_j(\boldsymbol{x})}{\mu} f_j(\boldsymbol{x}) - \sum_{k=1}^{n+1} M_{kj} \alpha_j g_j(\boldsymbol{x}).$$

Now, recalling that $\alpha_r g_r(\boldsymbol{x}) = \beta_r(\boldsymbol{x}) = \mu$ we get

$$\frac{\mu}{r} x_k = \sum_{j=1}^{n} \left(\frac{P_{kj} f_j(\boldsymbol{x})}{\mu} - M_{kj} \right) \beta_j(\boldsymbol{x}) - M_{kr}\mu \quad \forall k = 1, \ldots, m.$$

Substituting the expression in equation (2.6) we get the set of equations

$$x_k \left[\sum_{j=1}^{n} \left(\frac{a_j f_j(\boldsymbol{x})}{\mu} + \sigma_j - b_j \right) \beta_j(\boldsymbol{x}) + (\sigma_r - b_r)\mu \right] = \sum_{j=1}^{n} \left(\frac{P_{kj} f_j(\boldsymbol{x})}{\mu} - M_{kj} \right) \beta_j(\boldsymbol{x}) - M_{kr}\mu \quad \forall k = 1, \ldots, m. \tag{2.7}$$

The ribosome allocation equation (2.5) now reads

$$\sum_{j=1}^{n} \frac{\beta_j(\boldsymbol{x})}{g_j(\boldsymbol{x})} + \frac{\mu}{g_r(\boldsymbol{x})} = 1 \tag{2.8}$$

Equations (2.7) and (2.8) together form the *Balanced Growth Equations*. Our goal is to find a solution of these equations for which the growth rate of the modeled cell is maximum. The result of the manipulations in this section is that in the equations that we have derived the β_j appear linearly everywhere, except for $\beta_r = \mu$, which appears quadratically. This allows, as we will see in the next section, to set up a Linear Programming Problem.

2.4 Maximising the Growth Rate

Now that we have derived a set of equations that need to hold when the cell is in balanced growth, the Balanced Growth Equations, in this section we focus on maximising the growth rate. We will do this by splitting the optimization procedure into three steps.

First we fix the $\boldsymbol{x}$ vector of concentrations and solve the resulting system of algebraic equations for the β_j.

Second we use a trick, we treat the μ that appears at the denominator in the equations differently from the ones that appear linearly and call it $\tilde{\mu}$, to set up a Linear Programming Problem. We hence obtain a function H which, for fixed concentrations $\boldsymbol{x}$, takes as argument $\tilde{\mu}$ and gives as image the maximum μ relative to these values. At this step, in general we will have that $H(\tilde{\mu}) \neq \tilde{\mu}$ so the vector we have found does not correspond to a balanced growth solution and will not satisfy equation (2.7). For this reason next we look for a fixed point $\hat{\mu}$ of the function H, that is a point where $H(\hat{\mu}) = \hat{\mu}$. This point will satisfy the Balanced Growth Equations (2.7) and (2.8) and will be the maximiser of the problem (for fixed $\boldsymbol{x}$) that we are looking for.

The third and last step is to vary the $\boldsymbol{x}$ values and repeat this procedure for each combination of $\boldsymbol{x}$ (we restrict to the range of $\boldsymbol{x}$ where the concentrations are positive and such that the fluxes are also positive). With the last step we maximise over the $\boldsymbol{x}$ values and find the set of concentrations such that the full system attains maximum growth.

In what follows we will explain these steps in more detail. We will consider steps one and two in this section and step three in the following one. Recall the Balanced Growth Equations (2.7) and (2.8)

$$x_k\left[\sum_{j=1}^{n}\left(\frac{a_j f_j(\boldsymbol{x})}{\mu} + \sigma_j - b_j\right)\beta_j(\boldsymbol{x}) + (\sigma_r - b_r)\mu\right] = \sum_{j=1}^{n}\left(\frac{P_{kj} f_j(\boldsymbol{x})}{\mu} - M_{kj}\right)\beta_j(\boldsymbol{x}) - M_{kr}\mu$$

$$\sum_{j=1}^{n} \frac{\beta_j(\boldsymbol{x})}{g_j(\boldsymbol{x})} + \frac{\mu}{g_r(\boldsymbol{x})} = 1.$$

This is a set of non-linear equations, because the $f_j(\boldsymbol{x})$ and the $g_j(\boldsymbol{x})$ are non-linear in the $\boldsymbol{x}$ vector. We wish to solve this set of algebraic equations and find the set of concentrations such that the system attains maximum growth rate μ. The difficulty of this problem lies mainly in the non-linearity in the concentrations $\boldsymbol{x}$ and in the non-linearity in the growth rate μ which appears in the denominator in the first m equations. The first step, now, is to fix the $\boldsymbol{x}$ concentrations. For fixed $\boldsymbol{x}$ we can treat the functions $\beta_j(\boldsymbol{x})$ as variables (for fixed $\boldsymbol{x}$ they are just a rescaling of the α_j). The equations then become the following set of equations where we have dropped the dependancy on $\boldsymbol{x}$ in the β_j, in the f_j and g_j (hence we use the notation $\beta_j = \beta_j(\boldsymbol{x})$, $f_j = f_j(\boldsymbol{x})$ and $g_j = g_j(\boldsymbol{x})$ for fixed $\boldsymbol{x}$)

$$x_k\left[\sum_{j=1}^{n}\left(\frac{a_j f_j}{\mu}+\sigma_j-b_j\right)\beta_j+(\sigma_r-b_r)\mu\right]=\sum_{j=1}^{n}\left(\frac{P_{kj}f_j}{\mu}-M_{kj}\right)\beta_j-M_{kr}\mu$$

$$\sum_{j=1}^{n}\frac{\beta_j}{g_j}+\frac{\mu}{g_r}=1.$$

We notice that the equations are linear in the β_j but non-linear in μ. They would be linear in μ except for the μ terms that appear in the denominator. We therefore decide to treat the μ that appears in the denominator differently from the one that appears linearly. We call it $\tilde{\mu}$

$$x_k\left[\sum_{j=1}^{n}\left(\frac{a_j f_j}{\tilde{\mu}}+\sigma_j-b_j\right)\beta_j+(\sigma_r-b_r)\mu\right]=\sum_{j=1}^{n}\left(\frac{P_{kj}f_j}{\tilde{\mu}}-M_{kj}\right)\beta_j-M_{kr}\mu \tag{2.9}$$

$$\sum_{j=1}^{n}\frac{\beta_j}{g_j}+\frac{\mu}{g_r}=1. \tag{2.10}$$

For fixed $\tilde{\mu}$ and fixed $\boldsymbol{x}$ concentrations this problem is now linear in the β_j and in μ. We also have that the $\beta_j \geq 0$ because they relate to ribosomial fractions. We can then solve this problem for the β_j and μ as a linear optimisation problem using standard linear programming methods. We wish to solve the linear optimisation problem for the vector $\boldsymbol{\beta}=(\beta_1,\beta_2,\cdots\beta_n,\beta_{n+1})=(\beta_1,\beta_2,\cdots\beta_n,\beta_r)=(\beta_1,\beta_2,\cdots\beta_n,\mu)$ where we have substituted, $\mu=\beta_{n+1}=\beta_r$. The linear programming problem is then expressed as

$$\max_{\boldsymbol{\beta}}\{\mu|A(\boldsymbol{x},\tilde{\mu})\boldsymbol{\beta}=0,\beta_j\geq 0\}$$

where as we said $\boldsymbol{\beta}$ is the vector $\boldsymbol{\beta}=(\beta_1,\beta_2,\cdots\beta_n,\beta_r)$ and A is the $\tilde{\mu}$ dependent (and $\boldsymbol{x}$ dependent) matrix associated with the linear problem and defined using equations (2.9) and (2.10). That is, A is the matrix whose j-th row contains the coefficients of the $\boldsymbol{\beta}$ that appear in the j-th equation of (2.9), and the last row contains the coefficients of the $\boldsymbol{\beta}$ relative to equation (2.10). The matrix A is explicitly expressed as follows (notice that the $\tilde{\mu}$ is still present in the matrix)

$$A=\begin{bmatrix} x_1\left(\frac{a_1f_1}{\tilde{\mu}}+\sigma_1-b_1\right)-\frac{P_{11}f_1}{\tilde{\mu}}+M_{11} & x_1\left(\frac{a_2f_2}{\tilde{\mu}}+\sigma_2-b_2\right)-\frac{P_{12}f_2}{\tilde{\mu}}+M_{12} & \dots & x_1(\sigma_r-b_r)-M_{1r} \\ x_2\left(\frac{a_1f_1}{\tilde{\mu}}+\sigma_1-b_1\right)-\frac{P_{21}f_1}{\tilde{\mu}}+M_{21} & x_2\left(\frac{a_2f_2}{\tilde{\mu}}+\sigma_2-b_2\right)-\frac{P_{22}f_2}{\tilde{\mu}}+M_{22} & \dots & x_2(\sigma_r-b_r)-M_{2r} \\ \vdots & \vdots & & \vdots \\ \frac{1}{g_1} & \frac{1}{g_2} & \dots & \frac{1}{g_r} \end{bmatrix}$$

For each fixed $\tilde{\mu}$ (and for fixed concentrations $\boldsymbol{x}$), either the linear programming problem has no feasible solutions, or it does. If it does have solutions, we can find a maximiser, the maximized growth rate relative to $\tilde{\mu}$ and we call it $\mu^{opt}_{\tilde{\mu}}$. For each $\tilde{\mu}$ the maximizer lies on an extreme ray of the cone spanned by the matrix operator A and the constraints $\beta_j \geq 0$. This solution can in general have a μ value different from $\tilde{\mu}$ and therefore will not be a solution of the Balanced Growth Equations. So, in order to find the maximum growth (for fixed $\boldsymbol{x}$) we need to find a point such that $\mu^{opt}_{\tilde{\mu}}=\tilde{\mu}$. In order to do this we consider the function $H:\Re\to\Re$, $H(\tilde{\mu})=\mu^{opt}_{\tilde{\mu}}$. If we can find a fixed point of the function H, that is some $\tilde{\mu}$ such that $\tilde{\mu}=\mu^{opt}_{\tilde{\mu}}$, then we will have a maximiser which is also a solution of the Balanced Growth Equations (2.9) and (2.10). This particular $\tilde{\mu}$ lies

also on an extreme ray of the cone associated with the linear problem (since every such $\tilde{\mu}$ does lie on an extreme ray of the cone associated with the linear programming problem, also this particular $\tilde{\mu}$ does). This particular $\tilde{\mu}$ corresponds also to a solution of the Balanced Growth Equations (for fixed $\boldsymbol{x}$), hence it is the maximum growth rate that we were looking for.

Any extreme ray of the cone will have many $\beta_j = 0$. Hence, the corresponding metabolic network has only those enzymes for which $\beta_j \geq 0$. From Linear Programming theory we know that the number of nonzero entries in the maximiser is minimal [12]. We call such pathways *Elementary Growth Modes*, (in correspondence with the Elementary Flux Modes). They are minimal pathways in which it is not possible to discard any reaction without making the network disconnected. We will explore the last step of this procedure in the following section.

2.5 Deriving the Optimality Equations

In this section we continue with the last step from the previous one, namely maximizing within one EGM. Then we derive a set of equations, the *Optimality Equations*, which need to hold for a point in the $\boldsymbol{x}$ and $\boldsymbol{\beta}$ space which has the maximum growth rate.

The last maximizing step of the procedure we have seen in the previous section is to maximise the growth rate μ within one EGM. We hence wish to maximise over the $\boldsymbol{x}$ concentrations in order to find the maximum growth rate of the full problem. For each $\boldsymbol{x}$ we find a set of β_j such that the growth rate μ is maximum. We repeat this procedure for each combination of $\boldsymbol{x}$ in the range where the $\boldsymbol{x}$ concentrations are positive and allow positive fluxes and choose the maximum growth rate μ over all the combinations (clearly we first discretize the space of concentrations $\boldsymbol{x}$). If this maximisation procedure will prove out to be effective, the maximum growth rate μ over the $\boldsymbol{x}$ that we will find will be the maximum growth rate of the problem.

We are now ready to derive the *Optimality Equations*. In order to do this we restate the problem in more abstract terms. We are looking for the maximum

$$\max_{\boldsymbol{x},\boldsymbol{\beta}} \{\mu = \beta_r | F_i(\boldsymbol{x}, \beta_1, \cdots, \beta_n, \beta_r), \quad i = 1, \cdots, n+1\}$$

where the $F_i(\boldsymbol{x}, \boldsymbol{\beta})$ functions are the Balanced Growth Equations (2.7) and (2.8)

$$F_i(\boldsymbol{x}, \boldsymbol{\beta}) = x_i \left[\sum_{j=1}^{n} \left(\frac{a_j f_j(\boldsymbol{x})}{\mu} + \sigma_j - b_j \right) \beta_j(\boldsymbol{x}) + (\sigma_r - b_r)\mu \right] - \sum_{j=1}^{n} \left(\frac{P_{ij} f_j(\boldsymbol{x})}{\mu} + M_{ij} \right) \beta_j(\boldsymbol{x}) - M_{kr}\mu$$

$$\forall i = 1, \cdots, n$$

$$F_r(\boldsymbol{x}, \boldsymbol{\beta}) = F_{n+1}(\boldsymbol{x}, \boldsymbol{\beta}) = \sum_{j=1}^{n} \frac{\beta_j(\boldsymbol{x})}{g_j(\boldsymbol{x})} + \frac{\mu}{g_r(\boldsymbol{x})} - 1.$$

We are looking for a point in the $(\boldsymbol{x}, \boldsymbol{\beta})$ space which has maximum $\mu = \beta_r$. This means that if we consider the set

$$S := \{(\boldsymbol{x}, \boldsymbol{\beta}) \in \Re^{2n+1} | F_i(\boldsymbol{x}, \boldsymbol{\beta}) = 0, \quad i = 1, \cdots, n+1\}$$

we look for clitical points on the surface S. Each vector that is tangent to a point in this set is normal to all the gradients of the functions $F_i(\boldsymbol{x}, \boldsymbol{\beta})$. Specifically, since we want to maximise μ, we

look for points where all tangent vectors do not have a μ component. In particular every such vector which is normal to a point in this set which has maximum $\mu = \beta_r$ will have 0 value in the last component: $v = (v_1, \cdots, v_{2n}, 0)$. So we have the relation

$$v_1 \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{x_1} + \cdots + v_{2n} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\beta_n} = 0$$

where the subscripts mean the partial derivative respect to that variable. This means that these column vectors are linearly dependent, that is, the matrix C with these $2n$ vectors as columns has not maximal rank. The matrix C has dimensions $(n+1) \times 2n$ so we have that $rank(C) \leq n$.

Now, to obtain a set of algebraic equations we need to make a further assumption. We assume that, for fixed $\boldsymbol{x}$ the Balanced Growth Equations have a unique local solution. Then, by the Implicit Function Theorem, the matrix

$$B = \left(\frac{\partial F_i}{\partial \beta_j} \right)_{i,j=1,\cdots,n}$$

which occurs as a submatrix of C, is invertible and has maximal rank ($rank(B) = n$). In the light of this we can say that the matrix C has rank n, but since C has $n+1$ rows it means that all the $(n+1) \times (n+1)$ submatrices of C have zero determinant. This is equivalent to saying that the following matrices must have zero determinant at the maximal point

$$D_i = \left[\begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{x_i} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\beta_1} \cdots \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\beta_n} \right]$$

We have then derived the *Optimality Equations*

$$detD_i = 0, \quad \forall i = 1, \cdots, n \tag{2.11}$$

These are a set of n equations that, in addition to the Balanced Growth Equations, need to hold when a point is a critical point of the problem. We note that the whole set of Balanced Growth and Optimality Equations is a square system: it has as many equations as unknowns, hence its solutions are generically isolated. We will use the Optimality Equations to double check that the point we have found is actually a maximum and to continue the solution respect to some parameters in the problem.

2.6 The μORAC framework

In this section we introduce the μORAC framework which is associated to the qORAC framework (specific flux Optimisation by Robust Adaptative Control [10]) but in the setting of growth rate optimisation. We know that bacteria are able to quickly adapt to changes in external conditions. For example when a substrate has been consumed, the cell has to be able to readily change the ribosome allocation in order to produce the enzymes used to metabolise other possible substrates. The cell has the ability to sense some external conditions but it is inconvenient to rely only on this kind of sensitivity. It instead relies on internal sensor metabolites which are used to predict

optimal ribosomial fractions α_j. This ribosome allocation is then used to steer the cell to a new optimum relative to the new external concentrations. The predicted optimal ribosomal fractions are steady state optimal fractions, this means that if the system is not yet in steady state, it is not yet optimal. As long as there are dynamics, the ribosomal fractions also change, and it seems plausible that the only possible steady state the system can attain is the real optimum, corresponding to external concentrations (which were not used in the predictions of the ribosomal allocation).

We now construct an adaptive control, in which predicted optimal ribosomal allocation is integrated with the Whole Cell Model. We restrict ourselves to system that contain only one EGM. To do this we consider the Whole Cell Model (2.4), where we have substituted the $\boldsymbol{\alpha}$ with the corresponding $\boldsymbol{\beta}$ expressions

$$\begin{cases} \dot{x}_k = \sum_{j=1}^{n} P_{kj} e_j f_j(\boldsymbol{x}) - \sum_{j=1}^{n+1} M_{kj} r \alpha_j g_j(\boldsymbol{x}) - \mu x_k \quad , \quad \forall k = 1, \dots, m \\ \dot{e}_j = r\beta_j(\boldsymbol{x}) - \mu e_j \qquad\qquad , \qquad\qquad \forall j = 1, \dots, n \\ \dot{r} = r\beta_r(\boldsymbol{x}) - \mu r \\ \mu = \sum_{k=1}^{m} \rho_k \sum_{j=1}^{n} P_{kj} v_j + \sum_{j=1}^{n+1} \left(\sigma_j - \sum_{j=1}^{m} \rho_k M_{kj} \right) w_j \end{cases}$$

In order to complete the Whole Cell Model, we need to supply the ribosomal allocation, $\boldsymbol{\beta}$, which need to be determined by internal sensor metabolite values $\boldsymbol{x}_S$, so we set $\boldsymbol{\beta} = \boldsymbol{\beta}(x_S(t))$. These ribosomal allocations are to be the optimal ribosomal allocation parameters in steady state, if $\boldsymbol{x}_S$ is optimal itself. They may be computed by solving the Balanced Growth Equations (2.7) and (2.8) and the Optimality Equations (2.11) in which the metabolite concentrations have been substituted by dummy variables $\boldsymbol{\xi}$ (including the external concentrations, which are unknown to the adaptive control we are constructing here). The $\boldsymbol{\xi}$ variables contain both the $\boldsymbol{x}$ and external substrate variables. We will then prescribe some of the variables to make the system square again.

$$\xi_k \left[\sum_{j=1}^{n} \left(\frac{a_j f_j(\boldsymbol{\xi})}{\mu} + \sigma_j - b_j \right) \beta_j(\boldsymbol{\xi}) + (\sigma_r - b_r)\mu \right] = \sum_{j=1}^{n} \left(\frac{P_{kj} f_j(\boldsymbol{\xi})}{\mu} - M_{kj} \right) \beta_j(\boldsymbol{\xi}) - M_{kr}\mu$$

$$\frac{\beta_1}{g_1(\boldsymbol{\xi})} + \cdots + \frac{\beta_n}{g_n(\boldsymbol{\xi})} + \frac{\beta_{n+1}}{g_{n+1}(\boldsymbol{\xi})} = 1$$

$$D_i = \begin{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\xi_i} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\beta_1} \cdots \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n+1} \end{bmatrix}_{\beta_n} \end{bmatrix}$$

$$detD_i = 0, \quad \forall i = 1, \cdots, n$$

$$\boldsymbol{\xi}_S = \boldsymbol{x}_S$$

In order to estimate the optimal $\boldsymbol{\beta}$ values we have introduced dummy variables which allow to estimate the optimal solution from the values of the internal metabolites. The dummy variables $\boldsymbol{\xi}$ correspond to an optimal solution as estimated by the sensor values, under the assumption that the system is in steady state. The differential equation is then run and the ribosome allocation is allowed to dynamically change according to the sensor concentrations and relative optima we have supplied it with. At each time step, the system of Balanced Growth and Optimality Equations is a square system if we supply sensor values from the metabolic system $\boldsymbol{\xi}_S(t) = \boldsymbol{x}_S(t)$. The external concentrations are variables in these equations, and are therefore determined (predicted)

by $\boldsymbol{x}_S(t)$. They are the values at which $\boldsymbol{x}_S(t)$ would be the optimal steady state values. The predicted step gives the optimal steady state ribosomial fractions, which are fed into the metabolic system. This gives rise to changing enzyme levels, and hence changing metabolite levels. In this way, the sensor changes again, and we repeat. We assume that the cell stores this information in the genetic code and implements regulatory networks to make such predictions. This models how the cell is able to steer itself to the optimum while adapting to changing external conditions. Because of limited time we will implement the μORAC for the example without toxin only. In the numerical simulations, we first compute the steady state optimisers $(\boldsymbol{\xi}, \boldsymbol{\beta})$ and store them. Then we look up and update the correct value at each time point, on the basis of the sensor values.

Chapter 3

Maximising The Growth Rate μ within one EGM

In this thesis we wish to investigate numerically several low dimensional examples of the model described by equations (2.4). We will include the presence of stress factors such as toxins. We investigate these examples with the use of programs written in MATLAB. We wish to maximise the growth rate in this setting with the methods and procedure explained in chapter 2 and we expect that the cell will invest more resources into the enzymes that clean up the toxins as the toxin level increases in order to maintain high growth rate. Because of this we expect the toxin level to be low and the toxin-removing enzymes to have higher concentrations than the other ones in the optimum steady state.

We start with one example without toxins. The first example contains five variables: two metabolites, two enzymes and the ribosome. The following example contains seven variables: three metabolites, three enzymes and the ribosome. In the following example we model the toxin differently from the other ones, as a parameter. This example also contains seven variables: three metabolites, three enzymes and the ribosome. In the last example we attempt to investigate a case where the toxin diffuses over the cellular membrane. This scenario differs from the others as there will not be an enzyme responsible for importing the toxin, if such an enzyme was present, the optimal solution would be one where that enzyme is not produced. All the examples that we investigate consist of one EGM only, so in each of the three first examples we have that the number of metabolites equals the number of enzymes wich equals the number of import reactions (there is one additional reaction for the production of the ribosome). The last example deviates from this setting as the number of reactions does not equal the number of metabolites.

For each example, for fixed parameters that describe the system, we wish to find the optimal metabolite and enzyme concentrations. This means we wish to find metabolites and enzymes concentrations for which the system is in steady state and which has maximum μ value. This is a static problem, dynamics of the differential equations are not involved. We wish instead to find a specific steady state of the system.

This is done with the methods we have introduced in chapter 2. We first create a function which solves the linear programming problem $\max_{\boldsymbol{\beta}}\{\mu|\boldsymbol{A}(\boldsymbol{x},\tilde{\mu})\boldsymbol{\beta} = 0, \beta_j \geq 0\}$. That is a function that solves the linear optimization problem for fixed $\tilde{\mu}$. We have then a function $H : \Re \to \Re$, $H(\tilde{\mu}) = \mu_{\tilde{\mu}}^{opt}$ which associates an optimum solution $\mu_{\tilde{\mu}}^{opt}$ to each $\tilde{\mu}$.

We then find a fixed point of this function $\hat{\mu}$. This is a point such that $H(\tilde{\mu}) = \tilde{\mu}$. In this way we have found the optimal growth rate for fixed metabolite concentrations. We then use an optimisation routine which finds the maximum growth rate in the metabolites space. We restrict our search in the range of metabolites which make the fluxes positive so that they happen from the exterior to the interior of the cell (because we want the cell to grow). We also impose that the cell has constant osmotic pressure so we include equation (2.3) and we have $\boldsymbol{\rho} \cdot \boldsymbol{x} < 1$. Once we find the optimal solution we check for optimality using the Balanced Growth Equations (2.7) and the Optimality Equations (2.8). That is, we check if the found solution is a zero of both sets of equations. At the end of the first example we use pseudo-arc length continuation to continue the solution and find

a curve of different optima at different substrate concentrations. We then include the continuation data in the system of differential equations to implement the μORAC framework for this example.

In order to solve the problem we have used some inbuilt functions, in particular one that solves the linear optimisation problem, one that finds the minimum of a non linear function and two root finding functions. We found out that the algorithm relies heavily on the parameters in these functions.

During this work we have encountered some numerical difficulties. We will now briefly explain the main numerical techniques we have used and where the numerical difficulties have arisen.

In order to investigate the examples numerically we wish to create an algorithm that maximises the growth rate, that is, an algorithm that finds the optimal metabolite and enzyme concentrations for which the system attains optimal growth rate. To fulfil this task we will at first make a function that maximises the growth rate for fixed metabolite concentrations (using linear programming as we have seen in Chapter 2) and then we will maximize the growth rate over all possible metabolite concentrations making a grid in the $\boldsymbol{x}$. The first maximisation function makes use of linear programming and of the inbuilt function *linprog*. The second one is a composite routine that uses a grid method and the inbuilt minimisation function *fminsearch*. We will see that the solution will depend on the parameters in the model.

While working on the minimisation routine some difficulties have arisen. We explain these in what follows. In order for the minimisation routine function *fminsearch* to find a solution we need to supply it with a starting point to start the search from. If the initial point is not enough close to the maximum, *fminsearch* might just fail to find a solution, or may be attracted to a different basin of attraction. We found out that this is a crucial step in the algorithm architecture because *fminsearch* highly depends on the choice of the initial point. We could provide this initial point in some cases but we are left with the general problem to find a solution for every problem.

In order to overcome this difficulty of finding a starting point (in the metabolite space) we first use a routine that creates a grid in the metabolite concentrations. We describe it now in the setting of the first example with five variables, where the grid is then two dimensional (we have two metabolites $\boldsymbol{x}$ in the example without toxin). We create a grid $n \times n$ in the rectangle of metabolite concentrations where all the fluxes are positive. We tried with different values of n but we found out by running some tests that the value that is more efficient computationally is $n = 5$ so in what follows we will always use a grid which divides each coordinate range in 4 equal segments and 5 grid points. We then calculate the growth rate in each of the n^2 grid points points and choose the maximum. We then continue this process by choosing a small box around the found grid point of length $\frac{1}{n}$ (the point is in the interior of the grid and taking care to define it properly if the point lies on the edge of the box). We then continue this process iteratively until the maximum length of the small box we found at the current step is smaller than a chosen precision value. We have then found an approximation of the solution with the desired precision. After that we use the point we have found to further maximise the solution using the minimisation routine *fminsearch*. We found out that the minimising routine *fminsearch* relies heavily on the grid routine. If the the grid refining of the solution is stopped too early (that is the required precision is low), *fminsearch* often fails to find the optimal growth rate and settles on a lower value. One hypotesis is that function we wish to minimise has several isolated local minima, however, as we will see in the results chapter, it is probable that the maximisation routine that we have used is not fully reliable. The composite routine seems to be able to find a solution but only to some degree of approximation, if we try to magnify the H function around the maximum in the grid routine we will find a pattern of points where the value of μ is close to zero and points where μ is close to the maximum. According to this the plot of the function H seems to be noisy. At the end of the results section we will see that this function is probably not noisy at all and instead the it appears noisy because the algorithm that we have used (the optimisation routine) is not reaiable. Moreover the complexity of the routine is exponential.

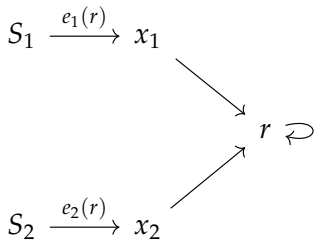
We have tried to find a better routine to find the initial point as it is clear that the complexity of the grid method is exponential respect to the number of (metabolite) variables. Here we briefly

describe another attempt we tried to improve the grid method, this is a random search method. The function still makes a grid 5×5 but then visits these points randomly until it finds a point where the growth rate is 'big enough' to use as a starting point. Clearly the problem here is that we do not know what is a good candidate for 'big enough'. In particular we found out that in the second example where the grid is $5 \times 5 \times 5$, if we choose the 5 points where the growth rate is the biggest (which happen to be close to each other), and we use each of these points as a starting point we get significantly different results for each of the points. The growth rate that we find when we supply *fminsearch* with the point that has biggest growth rate is significantly bigger than the growth rate we find when we supply the other 4 points. It appears that the minimisation routine *fminsearch* depends on the starting point when used to find a minimum for this problem. These computational difficulties will be more evident when we will plot the function in the results chapter.

In the remainder of this Chapter we introduce and state the full description of the four examples of the model that we have studied. In Chapter 4 we describe the results we have found by numerically investigating them.

3.1 Example Without Toxin

The first example we describe contains two metabolites, two enzymes and the ribosome. With this example we wish illustrate the model we have introduced in Chapter 2 with a simple system that contains at least two different enzymes. The reaction network is depicted by the diagram below:



where the arrows correspond to the reactions. Also the expression $e_1(r)$ is not a functional dependency but expresses that the production of that enzyme is catalysed by (a fraction) of the ribosome. In this case the total ribosome is split into three parts: $\alpha_1 + \alpha_2 + \alpha_r = 1$. The first two catalyses the production of the two enzymes that catalyse each import reaction and the rest catalyses the production of the ribosome itself. To be noticed is that the cell membrane is not showed but it is understood that in this case x_1, x_2 and r lie inside the cell while the substrates S_1 and S_2 lie outside the cell.

We see that we have two different substrates which are imported and metabolised into two metabolites, each import reaction is catalysed by one enzyme whose production is catalysed by a fraction of the ribosome, while the rest of the ribosome is used to catalyse the production of itself. Also we see that to produce both the ribosome r and the other enzymes e_1 and e_2 both metabolites are used.

The differential equations system associated to this example is then

$$\begin{cases} \dot{x}_1 = v_1 - M_{11}w_1 - M_{12}w_2 - M_{1r}w_r - \mu x_1 \\ \dot{x}_2 = v_2 - M_{21}w_1 - M_{22}w_2 - M_{2r}w_r - \mu x_2 \\ \dot{e}_1 = w_1 - \mu e_1 \\ \dot{e}_2 = w_2 - \mu e_2 \\ \dot{r} = w_r - \mu r \end{cases}$$

and the matrix associated to it is

$$\left(\begin{array}{cc|ccc} 1 & 0 & -M_{11} & -M_{12} & -M_{1r} \\ 0 & 1 & -M_{21} & -M_{22} & -M_{2r} \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c} v_1 \\ v_2 \\ w_1 \\ w_2 \\ w_r \end{array}\right) = \mu \boldsymbol{x}.$$

We have chosen also the formula for the functions f_1 and f_2. The shape of these functions could be arbitrary, we assume them to be direct proportional respect to the metabolite(s) but also saturating respect to both metabolite(s) and substrate(s). The formula we have chosen is the classic reversible Michaelis-Menten reaction formula. The functions f_1 and f_2 are

$$f_1(S_1, x_1) = k_{cat} \frac{S_1 - k_{f1} x_1}{1 + S_1 + x_1} \qquad v_1 = e_1 f_1(x_1, x_2)$$

$$f_2(S_2, x_2) = k_{cat} \frac{S_2 - k_{f2} x_2}{1 + S_2 + x_2} \qquad v_2 = e_2 f_2(x_1, x_2).$$

The functions g_1, g_2 and g_r are chosen to be approximately proportional respect to the two metabolites x_1 and x_2 but also saturating for high concentrations of these. The reaction rate grows linearly for small increasing concentrations of x_1 and x_2 but this dependence slows down and tends to some constant for higher concentrations of the metabolites

$$g_1(x_1, x_2) = \frac{k_{g1} x_1 x_2}{1 + x_1 + x_2 + x_1 x_2} \qquad w_1 = r\alpha_1 g_1(x_1, x_2)$$

$$g_2(x_1, x_2) = \frac{k_{g2} x_1 x_2}{1 + x_1 + x_2 + x_1 x_2} \qquad w_2 = r\alpha_2 g_2(x_1, x_2)$$

$$g_r(x_1, x_2) = \frac{k_r x_1 x_2}{1 + x_1 + x_2 + x_1 x_2} \qquad w_1 = r\alpha_1 g_r(x_1, x_2)$$

with:

$$\alpha_1 + \alpha_2 + \alpha_r = 1$$

We have chosen the constants $M_{11} = 2$, $M_{12} = 4$, $M_{1r} = 3$, $M_{21} = 3$, $M_{22} = 4$ and $M_{2r} = 2$. From the network we have $P_{11} = P_{22} = 1$ and $P_{12} = P_{21} = 0$. With this choice the formula for the growth rate is:

$$\mu = \rho_1 \cdot v_1 + \rho_2 \cdot v_2 + (\sigma_1 - 2\rho_1 - 3\rho_2) w_1 + (\sigma_2 - 4\rho_1 - 4\rho_2) w_2 + (\sigma_r - 3\rho_1 - 2\rho_2) w_r$$

Finally the Balanced Growth Equations for this example are

$$F_1(x_1, x_2, \beta_1, \beta_2, \beta_r) = x_1 \left[\left(\frac{a_1 f_1}{\mu} + \sigma_1 - b_1 \right) \beta_1 + \left(\frac{a_2 f_2}{\mu} + \sigma_2 - b_2 \right) \beta_2 + (\sigma_r - b_r)\mu \right] + \\ - \left(\frac{P_{11} f_1}{\mu} + M_{11} \right) \beta_1 - \left(\frac{P_{12} f_2}{\mu} + M_{12} \right) \beta_2 - M_{1r} \mu$$

$$F_2(x_1, x_2, \beta_1, \beta_2, \beta_r) = x_2\left[\left(\frac{a_1 f_1}{\mu} + \sigma_1 - b_1\right)\beta_1 + \left(\frac{a_2 f_2}{\mu} + \sigma_2 - b_2\right)\beta_2 + (\sigma_r - b_r)\mu\right] + \\ -\left(\frac{P_{21} f_1}{\mu} + M_{21}\right)\beta_1 - \left(\frac{P_{22} f_2}{\mu} + M_{22}\right)\beta_2 - M_{2r}\mu$$

$$F_r(x_1, x_2, \beta_1, \beta_2, \beta_r) = \frac{\beta_1}{g_1(x_1, x_2)} + \frac{\beta_2}{g_2(x_1, x_2)} + \frac{\mu}{g_r(x_1, x_2)} - 1$$

3.2 Examples Modeling The Presence Of A Toxin

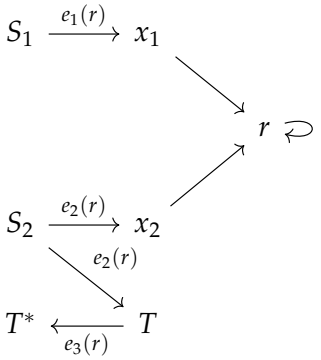
We will now investigate three examples that model the presence of a toxin in three different ways. The first one has 3 metabolites (one of them is the toxin), 3 enzymes, the ribosome and two external substrates. Two enzymes import and catalyse the metabolic reactions of the two substrates respectively. In this case the toxin import is enzymatic, it is 'involuntarily' imported by the second enzyme while importing the second substrate (we can imagine that the toxin binds to the substrate and then the substrate is imported or that the import reaction changes the substrate, so that a toxin is created), the third enzyme catalyses then the elimination of the toxin and the the ribosome that catalyses the production of each enzyme and of itself as usual.

In the second example, the toxin is modeled as a parameter that affects the efficiency of one of the import reactions. In this case we have 3 metabolites, 3 enzymes which catalyse the import and metabolism of one substrate each (hence we have 3 substrates) and the ribosome that catalyses the production of each enzyme. We suppose that in this case the cell membrane is completely permeable for the toxin. That is, as the concentration of the toxin changes outside the cell it instantaneously changes inside. The toxin parameter affects the efficiency of the import of one of the substrates: we have 3 substrates S_1, S_2 and S_3. The idea is that the cell can sustain itself using just two of the substrates, S_1 and S_2 or S_2 and S_3, S_1 and S_3 are interchangeable. However, in absence of the toxin, the import of the substrate S_1 is much more efficient while, for increasing values of the toxin, the import of substrate S_1 becomes less and less efficient. We will expect that for increasing values of the toxin, the cell will adapt and use less ribosome for the production of the enzyme that catalyses the import of S_1 and more for the production of the enzyme that catalyses the import of S_3. We will model this example in two slightly different ways, one where the cell needs all three metabolites to survive and one where it needs just a pair of them.

In the last example we still have 3 metabolites (one is the toxin) and 3 enzymes and the ribosome but the toxin now diffuses over the membrane and is not imported by any enzyme. One of the enzymes is again responsible for the elimination of the toxin while the other two catalyse the import of the two substrates. Lastly we have the ribosome that is responsible of catalysing the production of each enzyme and of itself. We will see that this example presents unforeseen difficulties because it deviates from the setting we have been working in, where each reaction is catalysed by exactly one enzyme. For this reason the equations of this example are derived but it is not investigated numerically.

3.2.1 Enzymatic Toxin Transport

The first example of this section is depicted by the following diagram:



The network is similar to the one we have studied in the previous section, except that we have one additional metabolite, namely the toxin, which is imported 'involuntarily' by the reaction which imports and metabolises the second substrate, we have then also one additional enzyme whose task is to remove the toxin from the interior of the cell and to transport it back to the outside across the cell membrane (possibly in a metabolised form). As we will see from the reaction rates functions, the modeled cell needs both metabolites to survive. They are both used to synthesise each of the enzymes. The system of differential equations for this example is the following

$$\begin{cases} \dot{x}_1 = v_1 - M_{11}w_1 - M_{12}w_2 - M_{13}w_3 - M_{1r}w_r - \mu x_1 \\ \dot{x}_2 = v_2 - M_{21}w_1 - M_{22}w_2 - M_{23}w_3 - M_{2r}w_r - \mu x_2 \\ \dot{T} = v_2 - v_3 - \mu T \\ \dot{e}_1 = w_1 - \mu e_1 \\ \dot{e}_2 = w_2 - \mu e_2 \\ \dot{e}_3 = w_3 - \mu e_3 \\ \dot{r} = w_r - \mu r \end{cases}$$

and the matrix associated to it is

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 0 & -M_{11} & -M_{12} & -M_{13} & -M_{1r} \\ 0 & 1 & 0 & -M_{21} & -M_{22} & -M_{23} & -M_{2r} \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \\ w_r \end{array}\right) = \mu \boldsymbol{x}$$

The functions that describe the reactions are then

$$f_1(S_1, x_1) = k_{cat} \frac{1}{K_T + T} \frac{S_1 - k_{f_1} x_1}{1 + S_1 + x_1} \qquad v_1 = e_1 f_1(x_1, x_2)$$

$$f_2(S_2, x_2, T) = k_{cat} \frac{S_2 - k_{f_2} T x_2}{1 + T + S_2 + x_2 + T x_2} \qquad v_2 = e_2 f_2(x_1, x_2, T)$$

$$f_3(T) = k_d \frac{T}{1 + T} \qquad v_3 = e_3 f_3(T)$$

$$g_1(x_1, x_2) = \frac{k_{g_1} x_1 x_2}{1 + x_1 + x_2 + x_1 x_2} \qquad w_1 = r \alpha_1 g_1(x_1, x_2)$$

$$g_2(x_1,x_2)=\frac{k_{g_2}x_1x_2}{1+x_1+x_2+x_1x_2} \qquad w_2=r\alpha_2 g_2(x_1,x_2)$$

$$g_3(x_1,x_2)=\frac{k_{g_3}x_1x_2}{1+x_1+x_2+x_1x_2} \qquad w_3=r\alpha_2 g_3(x_1,x_2)$$

$$g_r(x_1,x_2)=\frac{k_r x_1x_2}{1+x_1+x_2+x_1x_2} \qquad w_1=r\alpha_1 g_r(x_1,x_2)$$

with

$$\alpha_1+\alpha_2+\alpha_3+\alpha_r=1$$

We have added a term to the first function f_1 so that we have an inverse proportional relation with the toxin: the toxin slows down the import and metabolism of the first substrate. Also, the third function f_3 models the depletion of the toxin, it is chosen to be approximately proportional respect to the toxin but also saturating for increasing concentration of it.

For this example we have chosen the constants $M_{11}=3$, $M_{12}=4$, $M_{13}=2$, $M_{1r}=2$, $M_{21}=2$, $M_{22}=4$, $M_{23}=3$, $M_{2r}=2$ and $M_{31}=M_{32}=M_{33}=M_{3r}=0$ (since no metabolites are used to produce the toxin). From the network we have $P_{11}=P_{22}=P_{32}=1$, $P_{12}=P_{13}=P_{21}=P_{23}=P_{31}=0$ and $P_{33}=-1$ (since the transport happens from the inside to the outside of the cell membrane) so the formula for the growth rate is

$$\mu=\rho_1\cdot v_1+\rho_2\cdot v_2+\rho_3(v_2-v_3)+$$
$$+(\sigma_1-3\rho_1-2\rho_2)w_1+(\sigma_2-4\rho_1-4\rho_2)w_2+(\sigma_3-2\rho_1-3\rho_2)w_3+(\sigma_r-2\rho_1-2\rho_2)w_r.$$

Lastly we have that the Balanced Growth Equations for this example are

$$F_1(x_1,x_2,T,\beta_1,\beta_2,\beta_3,\beta_r)=$$
$$x_1\left[\left(\frac{a_1f_1}{\mu}+\sigma_1-b_1\right)\beta_1+\left(\frac{a_2f_2}{\mu}+\sigma_2-b_2\right)\beta_2+\left(\frac{a_3f_3}{\mu}+\sigma_3-b_3\right)\beta_3+(\sigma_r-b_r)\mu\right]+$$
$$-\left(\frac{P_{11}f_1}{\mu}+M_{11}\right)\beta_1-\left(\frac{P_{12}f_2}{\mu}+M_{12}\right)\beta_2-\left(\frac{P_{13}f_3}{\mu}+M_{13}\right)\beta_3-M_{1r}\mu$$

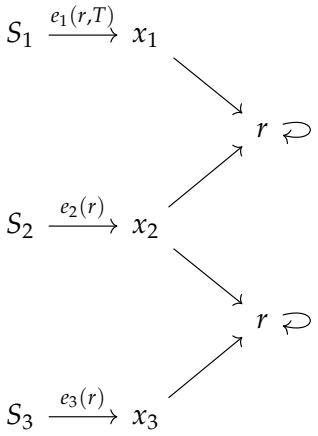
$$F_2(x_1,x_2,T,\beta_1,\beta_2,\beta_3,\beta_r)=$$
$$x_2\left[\left(\frac{a_1f_1}{\mu}+\sigma_1-b_1\right)\beta_1+\left(\frac{a_2f_2}{\mu}+\sigma_2-b_2\right)\beta_2+\left(\frac{a_3f_3}{\mu}+\sigma_3-b_3\right)\beta_3+(\sigma_r-b_r)\mu\right]+$$
$$-\left(\frac{P_{21}f_1}{\mu}+M_{21}\right)\beta_1-\left(\frac{P_{22}f_2}{\mu}+M_{22}\right)\beta_2-\left(\frac{P_{23}f_3}{\mu}+M_{23}\right)\beta_3-M_{2r}\mu$$

$$F_3(x_1,x_2,T,\beta_1,\beta_2,\beta_3,\beta_r)=$$
$$T\left[\left(\frac{a_1f_1}{\mu}+\sigma_1-b_1\right)\beta_1+\left(\frac{a_2f_2}{\mu}+\sigma_2-b_2\right)\beta_2+\left(\frac{a_3f_3}{\mu}+\sigma_3-b_3\right)\beta_3+(\sigma_r-b_r)\mu\right]+$$
$$-\left(\frac{P_{31}f_1}{\mu}+M_{31}\right)\beta_1-\left(\frac{P_{32}f_2}{\mu}+M_{32}\right)\beta_2-\left(\frac{P_{33}f_3}{\mu}+M_{33}\right)\beta_3-M_{3r}\mu$$

$$F_r(x_1,x_2,x_3,\beta_1,\beta_2,\beta_3,\beta_r)=\frac{\beta_1}{g_1(x_1,x_2)}+\frac{\beta_2}{g_2(x_1,x_2)}+\frac{\beta_3}{g_3(x_1,x_2)}+\frac{\mu}{g_r(x_1,x_2)}-1$$

3.2.2 Toxin Presence Leads To Substrate Use Switch

The second example that we consider models the toxin as a parameter. The network for this example is depicted by the following diagram:



The two r's are both the same ribosome, the diagram expresses that the ribosome can be produced using just metabolites x_1 and x_2 or just metabolites x_2 and x_3, our intention is to set the reaction functions so that that the cell can 'survive' using only the first two metabolites or only the second two. Moreover, for low concentrations of toxin, the import of the first substrate is more efficient than the import of the others, while for increasing values of toxin, the first import reaction becomes less and less efficient. We will see how this is expressed in the functions f's and g's. The differential equations system for this examples is

$$\begin{cases} \dot{x}_1 = v_1 - M_{11}w_1 - M_{12}w_2 - M_{13}w_3 - M_{1r}w_r - \mu x_1 \\ \dot{x}_2 = v_2 - M_{21}w_1 - M_{22}w_2 - M_{23}w_3 - M_{2r}w_r - \mu x_2 \\ \dot{x}_3 = v_3 - M_{31}w_1 - M_{32}w_2 - M_{33}w_3 - M_{3r}w_r - \mu x_3 \\ \dot{e}_1 = w_1 - \mu e_1 \\ \dot{e}_2 = w_2 - \mu e_2 \\ \dot{e}_3 = w_3 - \mu e_3 \\ \dot{r} = w_r - \mu r \end{cases}$$

and the matrix associated to it is

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 0 & -M_{11} & -M_{12} & -M_{13} & -M_{1r} \\ 0 & 1 & 0 & -M_{21} & -M_{22} & -M_{23} & -M_{2r} \\ 0 & 0 & 1 & -M_{31} & -M_{32} & -M_{33} & -M_{3r} \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \\ w_r \end{array}\right) = \mu \boldsymbol{x}$$

We have chosen the same form as before for the import functions of each of the substrates/metabolites, each of them is direct proportional in the substrate and inverse proportional in both the substrate and the metabolite, however, in order to model the effect of the toxin we add one term $\frac{5}{1+T}$ to the first import reaction f_1. We can see that when the toxin is not present $T = 0$, the first import reaction is 5 times more efficient than the others, for $T = 4$ the reaction is as efficient as the other two, while for increasing values of $T \geq 4$ the efficiency of this reaction becomes less and less. So the chosen import reactions are

$$f_1(S_1, x_1) = k_{cat}\frac{5}{1+T}\frac{S_1 - k_{f_1}x_1}{1+S_1+x_1} \qquad v_1 = e_1 f_1(x_1)$$

$$f_2(S_2, x_2) = k_{cat}\frac{S_2 - k_{f_2}x_2}{1+S_2+x_2} \qquad v_2 = e_2 f_2(x_2)$$

$$f_3(S_3, x_3) = k_{cat}\frac{S_3 - k_{f_3}x_3}{1+S_3+x_3} \qquad v_3 = e_2 f_2(x_3)$$

We have used two slightly different sets of enzyme production functions g's, the first set is

$$g_1(x_1, x_2, x_3) = \frac{k_{g_1}x_1x_2x_3}{1+x_1+x_2+x_3+x_1x_2+x_1x_3+x_2x_3+x_1x_2x_3} \qquad w_1 = r\alpha_1 g_1(x_1, x_2, x_3)$$

$$g_2(x_1, x_2, x_3) = \frac{k_{g_2}x_1x_2x_3}{1+x_1+x_2+x_3+x_1x_2+x_1x_3+x_2x_3+x_1x_2x_3} \qquad w_2 = r\alpha_2 g_2(x_1, x_2, x_3)$$

$$g_3(x_1, x_2, x_3) = \frac{k_{g_3}x_1x_2x_3}{1+x_1+x_2+x_3+x_1x_2+x_1x_3+x_2x_3+x_1x_2x_3} \qquad w_3 = r\alpha_3 g_3(x_1, x_2, x_3)$$

$$g_r(x_1, x_2, x_3) = \frac{k_r x_1x_2}{1+x_1+x_2+x_1x_2} + \frac{k_r x_2x_3}{1+x_2+x_3+x_2x_3} \qquad w_r = r\alpha_r g_r(x_1, x_2, x_3)$$

with

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_r = 1$$

In this set of equations we can see that the cell is able to synthesise the ribosome using only two metabolites (x_1 and x_2 or x_2 and x_3), however it needs all three metabolites to synthesise the other enzymes, we will see in the simulation that for high values of the toxin the cell adapts but also the growth rate becomes smaller as metabolite x_1 becomes less available. This is why we have then substituted the equations with the set

$$g_1(x_1, x_2, x_3) = \frac{k_{g_1}x_1x_2}{1+x_1+x_2+x_1x_2} + \frac{k_{g_1}x_2x_3}{1+x_2+x_3+x_2x_3} \qquad w_1 = r\alpha_1 g_1(x_1, x_2, x_3)$$

$$g_2(x_1, x_2, x_3) = \frac{k_{g_2}x_1x_2}{1+x_1+x_2+x_1x_2} + \frac{k_{g_2}x_2x_3}{1+x_2+x_3+x_2x_3} \qquad w_2 = r\alpha_2 g_2(x_1, x_2, x_3)$$

$$g_3(x_1, x_2, x_3) = \frac{k_{g_3}x_1x_2}{1+x_1+x_2+x_1x_2} + \frac{k_{g_3}x_2x_3}{1+x_2+x_3+x_2x_3} \qquad w_3 = r\alpha_3 g_3(x_1, x_2, x_3)$$

$$g_r(x_1, x_2, x_3) = \frac{k_r x_1x_2}{1+x_1+x_2+x_1x_2} + \frac{k_r x_2x_3}{1+x_2+x_3+x_2x_3} \qquad w_r = r\alpha_r g_r(x_1, x_2, x_3)$$

with

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_r = 1$$

In this second set of equations we can see that now the variables are separated in each function, hence the cell is able to use just a pair of metabolites to synthesise each enzyme. We should expect

that with this modified set of equations the modeled cell will adapt without significantly reducing the growth rate for high values of toxin. All the following simulations except the first one have been done with this set of equations (the first set of equations is commented in the code).

For this example we have chosen the constants $M_{11} = 3$, $M_{12} = 2$, $M_{13} = 4$, $M_{1r} = 1$, $M_{21} = 2$, $M_{22} = 1$, $M_{23} = 3$, $M_{2r} = 1$ and $M_{31} = 4$, $M_{32} = 3$, $M_{33} = 1$, $M_{3r} = 2$. From the network we have $P_{11} = P_{22} = P_{33} = 1$, $P_{12} = P_{13} = P_{21} = P_{23} = P_{31} = P_{32} = 0$. The formula for the growth rate in this case is

$$\mu = \rho_1 v_1 + \rho_2 v_2 + \rho_3 v_3 +$$
$$+(\sigma_1 - 3\rho_1 - 2\rho_2 - 4\rho_3)w_1 + (\sigma_2 - 2\rho_1 - \rho_2 - 3\rho_3)w_2 + (\sigma_3 - 4\rho_1 - 3\rho_2 - \rho_3)w_3 + (\sigma_r - \rho_1 - \rho_2 - 2\rho_3)w_r.$$

While the Balanced Growth Equations for this example are

$$\begin{aligned} &F_1(x_1, x_2, x_3, \beta_1, \beta_2, \beta_3, \beta_r) = \\ &x_1\left[\left(\frac{a_1 f_1}{\mu} + \sigma_1 - b_1\right)\beta_1 + \left(\frac{a_2 f_2}{\mu} + \sigma_2 - b_2\right)\beta_2 + \left(\frac{a_3 f_3}{\mu} + \sigma_3 - b_3\right)\beta_3 + (\sigma_r - b_r)\mu\right] + \\ &-\left(\frac{P_{11} f_1}{\mu} + M_{11}\right)\beta_1 - \left(\frac{P_{12} f_2}{\mu} + M_{12}\right)\beta_2 - \left(\frac{P_{13} f_3}{\mu} + M_{13}\right)\beta_3 - M_{1r}\mu \end{aligned}$$

$$\begin{aligned} &F_2(x_1, x_2, x_3, \beta_1, \beta_2, \beta_3, \beta_r) = \\ &x_2\left[\left(\frac{a_1 f_1}{\mu} + \sigma_1 - b_1\right)\beta_1 + \left(\frac{a_2 f_2}{\mu} + \sigma_2 - b_2\right)\beta_2 + \left(\frac{a_3 f_3}{\mu} + \sigma_3 - b_3\right)\beta_3 + (\sigma_r - b_r)\mu\right] + \\ &-\left(\frac{P_{21} f_1}{\mu} + M_{21}\right)\beta_1 - \left(\frac{P_{22} f_2}{\mu} + M_{22}\right)\beta_2 - \left(\frac{P_{23} f_3}{\mu} + M_{23}\right)\beta_3 - M_{2r}\mu \end{aligned}$$

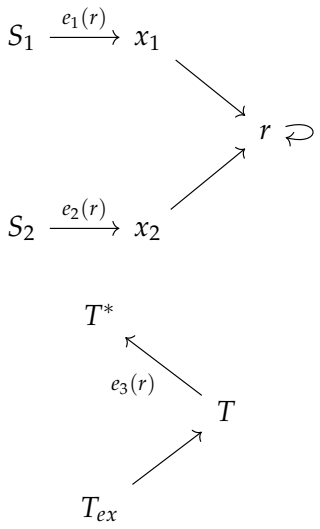
$$\begin{aligned} &F_3(x_1, x_2, x_3, \beta_1, \beta_2, \beta_3, \beta_r) = \\ &x_3\left[\left(\frac{a_1 f_1}{\mu} + \sigma_1 - b_1\right)\beta_1 + \left(\frac{a_2 f_2}{\mu} + \sigma_2 - b_2\right)\beta_2 + \left(\frac{a_3 f_3}{\mu} + \sigma_3 - b_3\right)\beta_3 + (\sigma_r - b_r)\mu\right] + \\ &-\left(\frac{P_{31} f_1}{\mu} + M_{31}\right)\beta_1 - \left(\frac{P_{32} f_2}{\mu} + M_{32}\right)\beta_2 - \left(\frac{P_{33} f_3}{\mu} + M_{33}\right)\beta_3 - M_{3r}\mu \end{aligned}$$

$$F_r(x_1, x_2, x_3, \beta_1, \beta_2, \beta_3, \beta_r) = \frac{\beta_1}{g_1(x_1, x_2, x_3)} + \frac{\beta_2}{g_2(x_1, x_2, x_3)} + \frac{\beta_3}{g_3(x_1, x_2, x_3)} + \frac{\mu}{g_r(x_1, x_2, x_3)} - 1$$

3.2.3 Toxin Over Membrane

In this section we attempt to investigate one example where the toxin diffuses over the cell membrane. The setting is similar to the last two examples we have seen. We have three metabolites (one is the toxin) and four enzymes, two enzymes import and metabolise one substrate each, one enzyme responsible of the depletion of the toxin and the ribosome which catalyses the production of each enzyme and of itself. The presence of the toxin changes the reaction rates of the import of both metabolites. This is expressed in the functions f_1 and f_2. The difference in this example is that there is not an enzyme associated with the import of the toxin. This deviates from the setting we specified where the number of reactions is the same as the number of enzymes. We will see that this difference will result in unforeseen difficulties. For this reason only the mathematical

setting of this example is described, no simulations are run for it. The network for this example is depicted by the following diagram



The matrix associated to it is the following

$$\left(\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0 & -M_{11} & -M_{12} & -M_{13} & -M_{14} & -M_{1r} \\ 0 & 1 & 0 & 0 & -M_{21} & -M_{22} & -M_{23} & -M_{24} & -M_{2r} \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_T \\ w_1 \\ w_2 \\ w_3 \\ w_T \\ w_r \end{pmatrix} = \mu \boldsymbol{x}$$

We can see that the P matrix is not a square matrix anymore. The differential equations system for this example is then the following

$$\begin{cases} \dot{x}_1 = v_1 - M_{11}w_1 - M_{12}w_2 - M_{13}w_3 - M_{14}w_T - M_{1r}w_r - \mu x_1 \\ \dot{x}_2 = v_2 - M_{21}w_1 - M_{22}w_2 - M_{23}w_3 - M_{24}w_T - M_{2r}w_r - \mu x_2 \\ \dot{T} = v_T - v_3 - \mu T \\ \dot{e}_1 = w_1 - \mu e_1 \\ \dot{e}_2 = w_2 - \mu e_2 \\ \dot{e}_3 = w_3 - \mu e_3 \\ \dot{r} = w_r - \mu r \end{cases} .$$

The $-v_3$ term in the third equation models the depletion of the toxin. The functions that specify the reactions rates are

$$f_1(S_1, x_1) = k_{cat} \frac{1}{K_T + T} \frac{S_1 - k_{1_f} x_1}{1 + S_1 + x_1} \qquad v_1 = e_1 f_1(x_1, x_2)$$

$$f_2(S_2, x_2, T) = k_{cat} \frac{1}{K_T + T} \frac{S_2 - k_{2_f} x_2}{1 + S_2 + x_2} \qquad v_2 = e_2 f_2(x_1, x_2, T)$$

$$f_3(T) = k_d \frac{T}{1 + T} \qquad v_3 = e_3 f_3(T)$$

$$g_i(x_1,x_2)=\frac{k_{i_g}x_1x_2}{1+x_1+x_2+x_1x_2} \qquad w_i=r\alpha_1 g_i(x_1,x_2) \quad \text{for} \quad i=1,2,3,r$$

We now wish to derive the Balanced Growth Equations. Since the toxin is not imported in the cell by any enzyme, we need to modify the derivations in Chapter 2. We start with the definition of the growth rate (2.3) which in general is

$$\mu=\sum_l \rho_l \sum_l N_{lj}v_j(\boldsymbol{c})$$

and is further specified by the formula

$$\mu=\sum_{k=1}^{m}\rho_k\sum_{j=1}^{n}P_{kj}v_j+\sum_{j=1}^{n+1}\left(\sigma_j-\sum_{k=1}^{m}\rho_k M_{kj}\right)$$

We simplify the expression

$$\sum_{k=1}^{m}\rho_n\sum_{j=1}^{n}N_{kj}v_j=\sum_{j=1}^{n}\left(\sum_{k=1}^{m}\rho_k N_{kj}\right)v_j=\sum_{j=1}^{n}a_jv_j$$

by introducing the term $a_j=\sum\limits_{k=1}^{m}\rho_k N_{kj}$ as in the general setting. We then have for the growth rate

$$\mu=\sum_{j=1}^{n}a_jv_j+\sum_{j=1}^{n}\Big(\sigma_j-b_j\Big)w_j-\Big(\sigma_{n+1}-b_{n+1}\Big)w_{n+1}$$

$$=a_Tv_T+\sum_{j=1}^{n-1}a_jv_j+\sum_{j=1}^{n-1}\Big(\sigma_j-b_j\Big)w_j-\Big(\sigma_{n+1}-b_{n+1}\Big)w_{n+1}$$

$$=a_Tv_T+r\cdot\left[\sum_{j=1}^{n-1}\left(\frac{a_jf_j}{\mu}+\sigma_j-b_j\right)\beta_j+\Big(\sigma_{n+1}-b_{n+1}\Big)\mu\right]$$

Which gives the formula for r as in the general setting, except that we have now the additional term $-a_Tv_T$

$$r=\frac{\mu-a_Tv_T}{\sum\limits_{j=1}^{n-1}\left(\frac{a_jf_j}{\mu}+\sigma_j-b_j\right)\beta_j+\left(\sigma_{n+1}-b_{n+1}\right)\mu}.$$

To simplify the notation, in the following we set

$$\gamma_j=\sigma_j-b_j$$

So we have the formula for the growth rate is

$$\mu=a_Tv_T+r\sum_{j=1}^{n-1}\left(\frac{a_jf_j}{\mu}+\gamma_j\right)\beta_j+r\gamma\mu.$$

We now derive separately the two cases, one for x_1 and x_2 and one for the toxin T. For the first two metabolites we have the formula:

$$\frac{\mu}{r}x_k = \sum_{j=1}^{n-1}\left(\frac{N_{kj}f_j}{\mu} - M_{kj}\right)\beta_j - M_{k,n+1}\mu \quad k = 1,2$$

while for the toxin ($k = 3$)

$$\mu T = v_T - v_3 = \sum_{j=1}^{4} N_{kj}v_j = v_T - e_3 f_3 \quad k = 3$$

$$\mu T = v_T - \frac{r\beta_3}{\mu}f_3 \quad k = 3$$

Again for the first two metabolites we have

$$x_k\mu = x_k a_T v_T + r x_k\left[\sum_{j=1}^{n-1}\left(\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_j\mu\right] \quad k = 1,2$$

$$x_k\mu = r\left(\sum_{j=1}^{n-1}\left(\frac{N_{kj}f_j}{\mu} - M_{kj}\right)\beta_j - M_{k,n+1}\mu\right) \quad k = 1,2$$

which implies

$$x_k a_T v_T + r x_k\left[\sum_{j=1}^{n-1}\left(\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_j\mu\right] =$$

$$= r\left(\sum_{j=1}^{n-1}\left(\frac{N_{kj}f_j}{\mu} - M_{kj}\right)\beta_j - M_{k,n+1}\mu\right)$$

while for the toxin

$$\mu T = v_T - r\frac{\beta_3}{\mu}f_3 \quad k = 3$$

$$\mu T = a_T v_T T + rT\left(\left(\sum_{j=1}^{n-1}\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_{n+1}\mu\right) \quad k = 3$$

which implies

$$v_T - r\frac{\beta_3}{\mu}f_3 = a_T v_T T + rT\left(\left(\sum_{j=1}^{n-1}\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_{n+1}\mu\right) \quad k = 3$$

Combining the equations above we finally have the formulas of the Balanced Growth Equations for both x_1, x_2 and T

$$x_k\left[\sum_{j=1}^{n-1}\left(\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_j\mu\right] - \sum_{j=1}^{n-1}\left(\frac{N_{kj}f_j}{\mu} - M_{kj}\right)\beta_j + M_{k,n+1}\mu = -\frac{xk}{r}a_T v_T \quad k = 1,2$$

$$\frac{v_T}{r}\left(1 - a_T T\right) = T\left(\left(\sum_{j=1}^{n-1}\frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_{n+1}\mu\right) + \frac{\beta_3}{\mu}f_3 \quad k = 3.$$

In this case we need to supply these with one additional equation for r as r appears explicitly in the right hand side of the linear programming problem

$$\mu - a_T v_T = r\left(\left(\sum_{j=1}^{n-1} \frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_{n+1}\mu\right.$$

$$\mu - r\left(\left(\sum_{j=1}^{n-1} \frac{a_j f_j}{\mu} + \gamma_j\right)\beta_j + \gamma_{n+1}\mu\right) = a_T v_T$$

We have then the linear programming problem

$$(L(x_1, x_2, T, r)\tilde{\mu}) \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \mu \end{pmatrix} = \begin{pmatrix} -\frac{x_1}{r} a_T v_T \\ -\frac{x_2}{r} a_T v_T \\ \frac{1-a_1 T}{r} v_T \\ a_T v_T \end{pmatrix}$$

We notice that the matrix L is now not only x_1, x_2 and T dependent as in the previous case but also r dependent. Moreover it is not a square matrix as in the other cases. Because of these difficulties we did not have enough time to investigate this example example numerically.

In the following Chapter we investigate numerically the examples above and report the results we have found.

Chapter 4

Results

In this section we discuss the results we have obtained using the MATLAB programs we have implemented for the examples we have described in the previous section. In the first three sections of this chapter we report the results we have obatined by investigating the first three examples. In the last section we will calculate the maximum growth rate for the first example in a different way, using the ODE dynamics to test the results we had. We will see that this last result will suggest that the optimisation algorithm that we have used seems to be not reliable. It is able to find the metabolite values for which the system attains maximum growth rate but it is not able to improve the precision of the solution. Moreover the graph of the H function calculated with our maximisation routine seems to have a high number of discontinuities. This is in contrast with our expectation that the H function should be continous (since the functions in the system are continous we expect that the maximum should be a continous respect to the metabolites concentrations). The test that we report in the last section will suggest that the dependancy of the maximum growth rate on the metabolites concentrations is indeed continous and hence the discontinous graph of the H function we have obtained should not be trusted. Due to lack of time, we could not consider this more robust technique in more detail for the other examples.

4.1 Example Without Toxin

As we have seen in the previous chapter for this example we have chosen the constants in the enzymes 'stoichiometry' matrix M as $M = (2, 4, 3; 3, 4, 2)$. Also there are eleven constants in this example. The constants that appear in the functions that describe the reactions: $\boldsymbol{k} = (k_{cat}, k_{f_1}, k_{f_2}, k_{g_1}, k_{g_2}, k_r)$ and the osmotic constants of the metabolites, enzymes and ribosomes, $\boldsymbol{\rho} = (\rho_1, \rho_2)$ and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_r)$.

In what follows we will always keep $k_{cat} = 0.25$ while the other constants will be a slight perturbation of $\boldsymbol{k} = (0.25, 0.5, 0.5, 0.5, 0.5, 0.5)$, $\boldsymbol{\rho} = (1, 1)$ and $\boldsymbol{\sigma} = (10, 10, 10)$. In particular the values that we use are:

$$\boldsymbol{k} = (0.2500, 0.4785, 0.5173, 0.5164, 0.4623, 0.4907)$$

$$\boldsymbol{\rho} = (0.9775, 1.0217)$$

$$\boldsymbol{\sigma} = (9.783, 10.396, 10.327).$$

Also we use both external substrate concentrations $S_1 = S_2 = 1$. When called with these parameters the optimisation routine gives the result vector

$$(x_1, x_2, \beta_1, \beta_2, \mu) = (0.2073, 0.1985, 0.0045, 0.0044, 0.0050).$$

As expected from the symmetry of the system we can see that the x_1 and x_2 concentrations are almost the same in the optimal steady state. This is the same for β_1 and β_2 (the constants are only a slight perturbation of the constants that make the system symmetric). We also notice that the β factors are smaller by about two order of magnitude. This result have been found with a grid precision of 10^{-5} (that is, we used a grid that gets refined until the stated precision is reached, then the result is further maximised using *fminsearch*).

The image of the optimum we have found when plugged in the Balanced Growth (2.7) and Optimality Equations (2.11) is $(0.0000, 0.0000, -0.0000, 0.0468, 0.0820)$. The first three entries of the vector are the image of the point found using the Balanced Growth Equations while the last two entries are the image of the point found using the Optimality Equations. We notice that all the entries of the image relative to the Balanced Growth Equations are zero up to the fourth decimal while the entries relative to the Optimality Equations are zero only up to the first decimal. This is probably because the Optimality Equations involve matrices and eigenvalues which can lead to multiplication of numerical errors while the Balanced Growth Equations do not.

Image 4.1 shows a plot of the function H calculated using the maximisation routine we have introduced in chapter 2 on a 121×121 grid of (x_1, x_2) values for the fixed constants above. For each point of the grid, hence for fixed metabolites values we have plotted the maximum μ value relative to this choice of metabolite concentrations. We notice that, althougth it has been calculated with high precision, the plot looks very noisy and discontinous. This is in contrast with our expectation that the H function is smooth (according to the functions we have chosen in the system, for small changes in the metabolite concentrations we expect small changes in the maximum growth rate that can be attained for these metabolites concentrations). In particular we expect that the H function has only one global maximum. The plot of the H function as calculated with the optimisation procedure seem to have several local maxima. We hence hypothesize that the maximisation procedure that we have used, althougth seems to be able to find the maximum, is not reliable and the discontinuity of the H function is a numberical artefact of the algorithm we have used. In the last section of this chapter, when we will calculate this function in a different way, we will have more evidence that seem to confirm this hypotesis.

Next we report the result that we have obtained by continuing the solution optimum we have found respect to substrate S_1 as a solution of the Balanced Growth and Optimality Equations using pseudo arc-length continuation. We use the combined set of Balanced Growth and Optimality Equations to calculate a one-parameter curve of optima parametrized by the substrate S_1. We have first adjusted the solution to a point very close to it which is a better zero of the combined set of equations using the function *fsolve* and we have then continued the solution on the curve defined by the combined set of equations of the Balanced Growth and Optimality Equations. Images 4.2 and 4.3 show the plot of the optimal β concentrations versus the substrate S_1. Because of convergence issues we were able to continue the solution only in a small interval, this issues were probably caused by the level of precision that the Optimality Equations provide. In the small interval considered the dependancy appears approximately linear.

Next we report a plot obtained by running the system of differential equations. Image 4.4 shows the evolution of each variable in the system when the system is run 20 times, each time with the same initial condition $[0.1, 0.1, 0.1, 0.1, 0.1]$ but with with random α_i.

We notice, as expected from the biological intuition but not necessarily from a system of non-linear differential equations, that for each random α's the system reaches a steady state. We can also see that for few sets of α's, the steady state of the ribosome is much bigger. These are choices of α's close to the values that give optimal growth rate.

Next we report the result of the μORAC control of this example. Image 4.5 shows the evolution of each variable in the system controlled using the continuation data.

We can see that, despite the small continuation interval that we have provided, the system reaches a stead state adjusting dynamically according to the internal sensor concentrations. We have included a dummy variable as the sixth variable to track the growth rate. The steady state that the

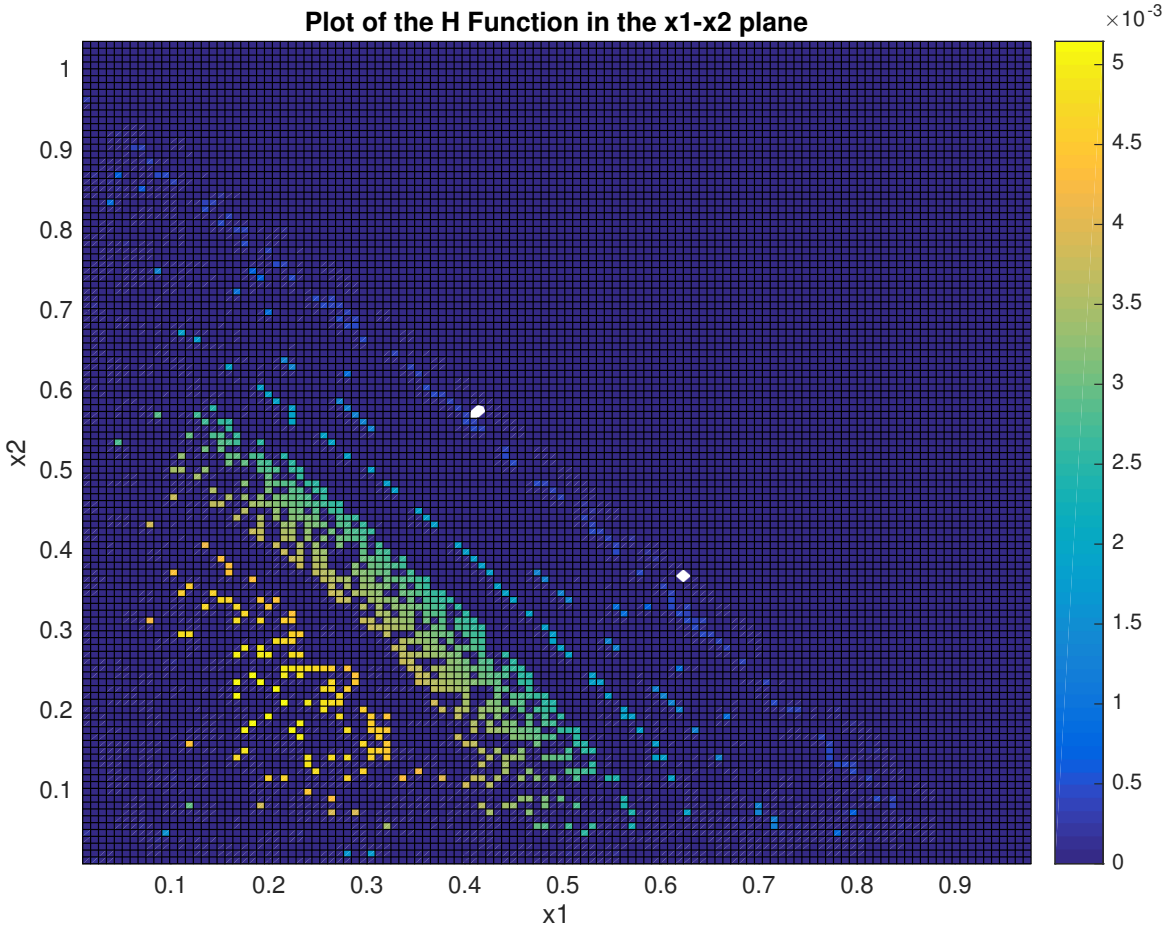


FIGURE 4.1: Example without toxin: image showing a surface plot of the result of growth rate maximizer routine seen from above. x and y axis are x_1 and x_2 variables respectively. The z axis is the value of the growth rate. The plot looks very noisy, discontinous and with lots of local maxima despite our expectation that the H function is continous. This graph seems to suggest that the maximisation routine we have used is not reliable.

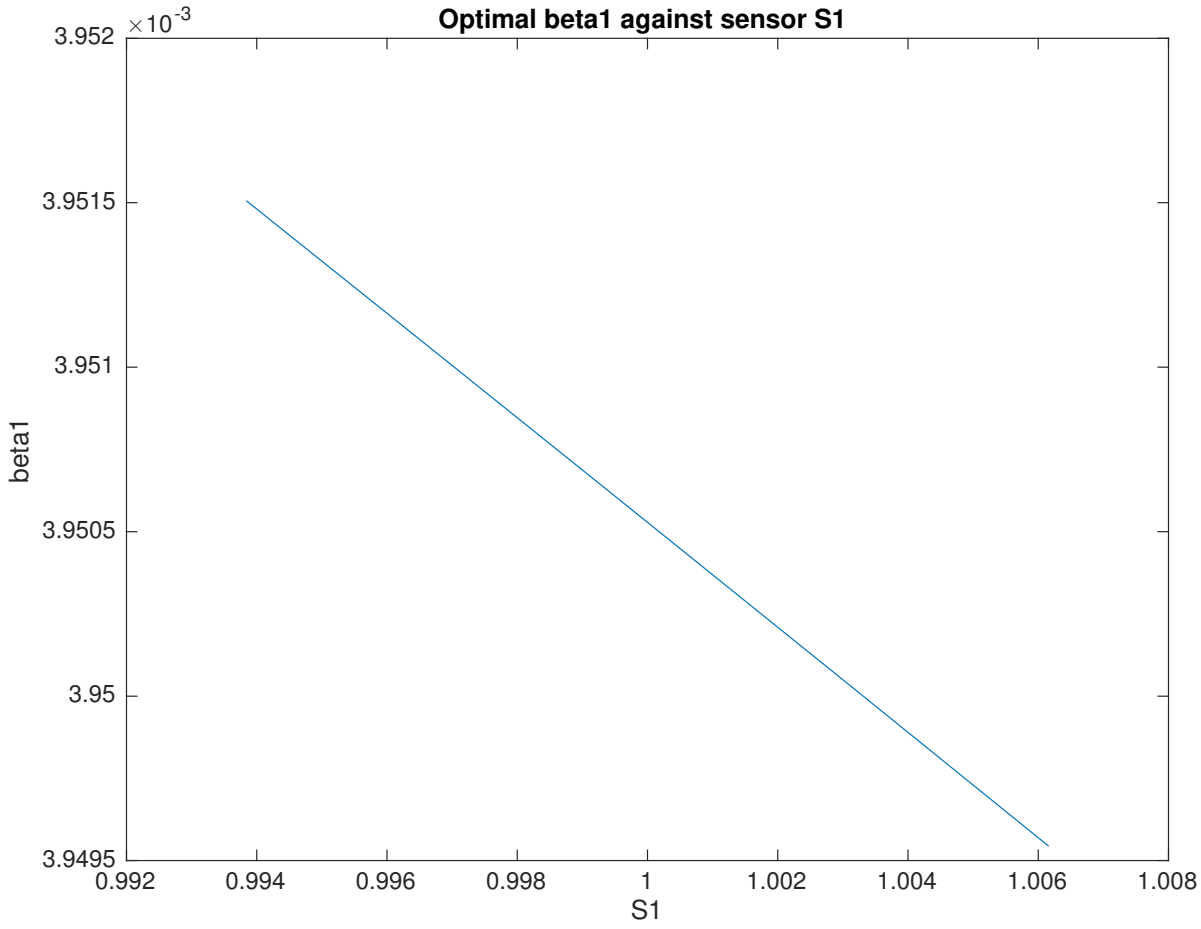


FIGURE 4.2: Example without toxin: image showing the plot of the optimal enzyme concentration β_1 versus the substrate concentration S_1. The plot has been produced using pseudo arc-length continuation

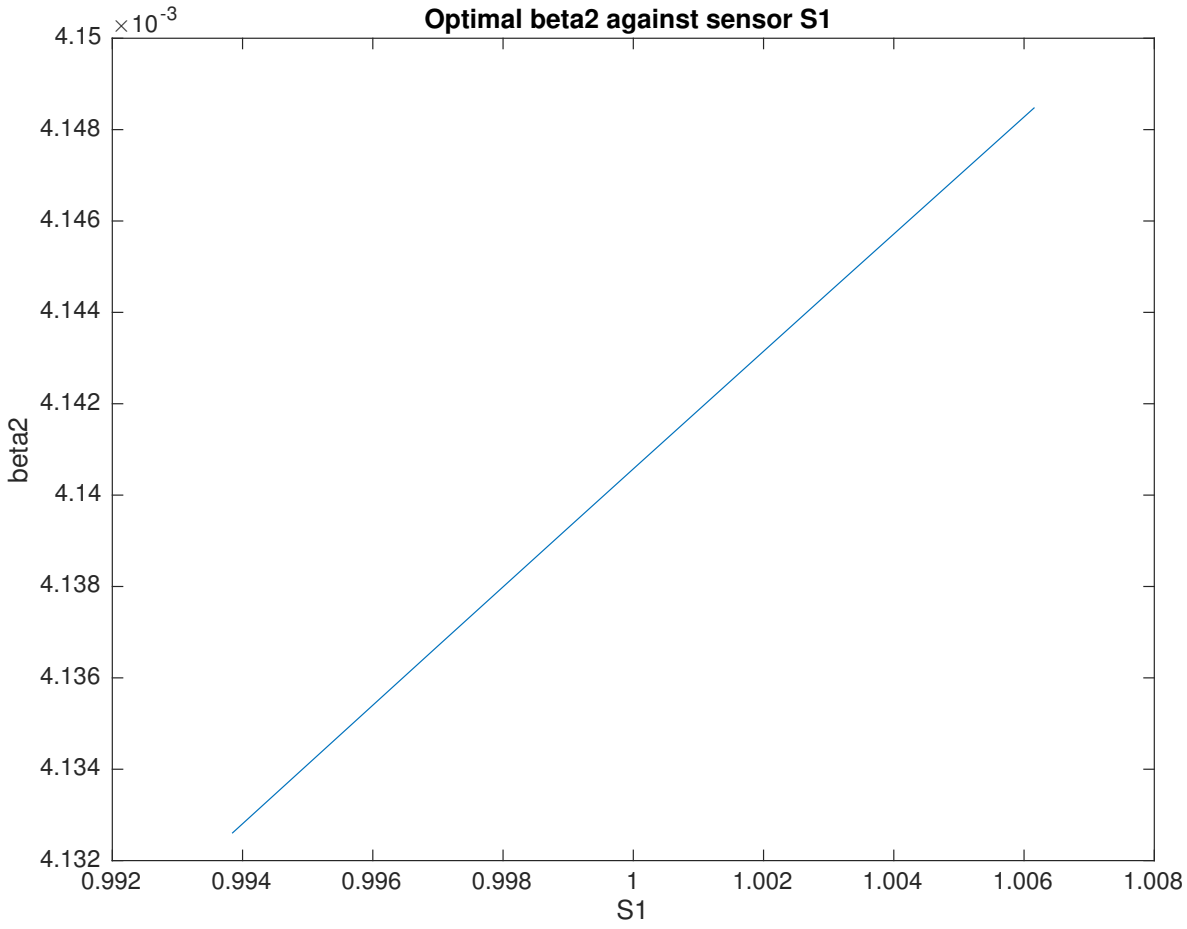


FIGURE 4.3: Example without toxin: image showing the plot of the optimal enzyme concentration β_2 versus the substrate concentration S_1. The plot has been produced using pseudo arc-length continuation

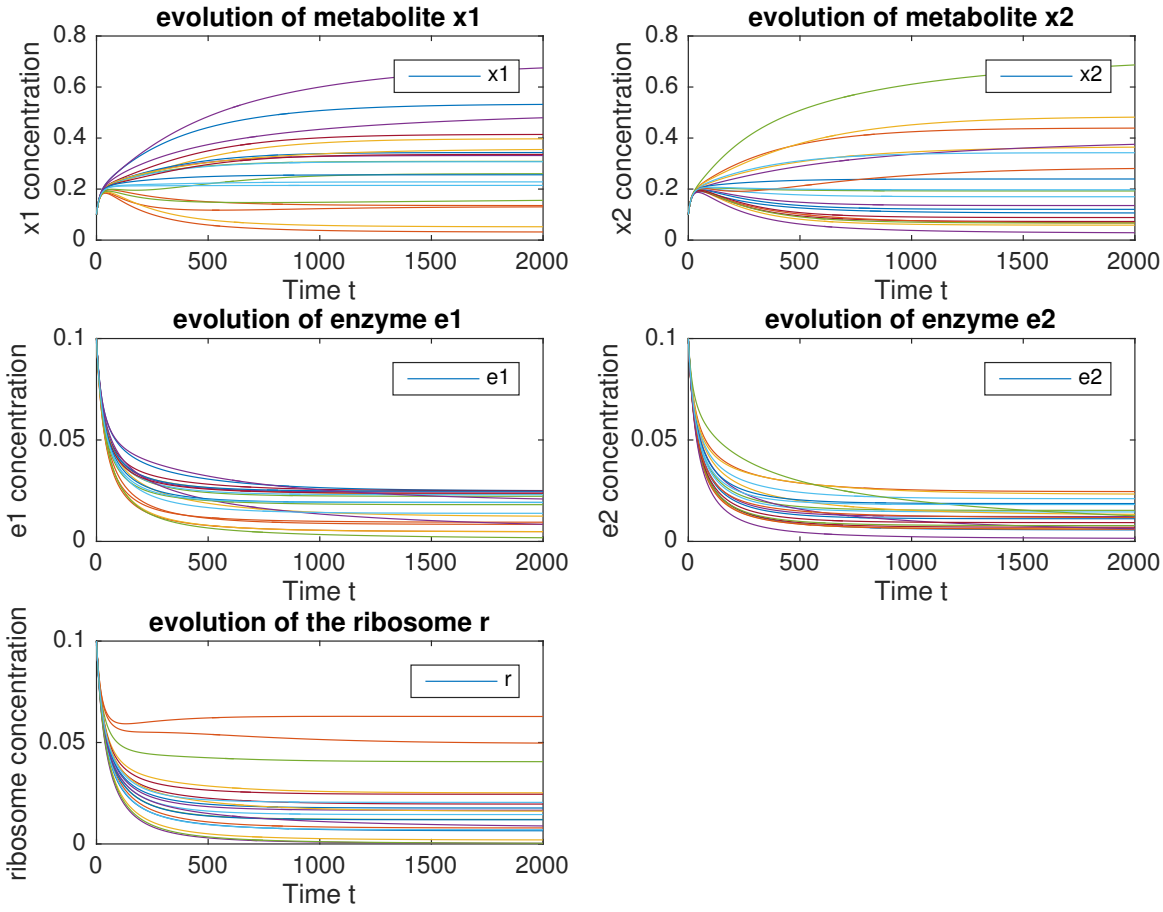


FIGURE 4.4: Example without toxin: image showing the evolution of the metabolites and enzymes. Each picture corresponds to a variable ordered as $(x_1, x_2, \beta_1, \beta_2, \mu)$. The picture has been produced by running the differential equations 20 times with random α's. We can notice that as expected, but differently from the oscillating or chaotic evolution nonlinear differential equations often present, for each random α_i the system reaches a steady state.

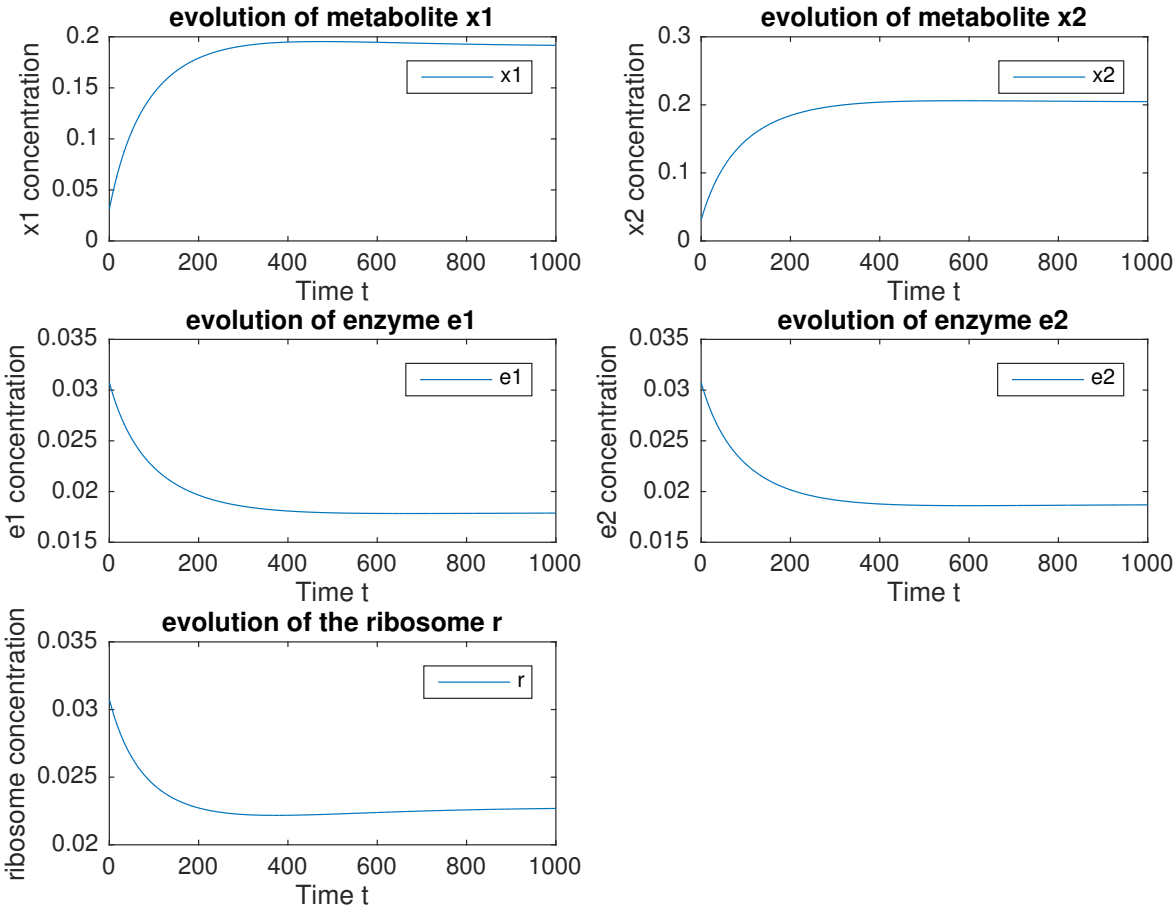


FIGURE 4.5: Example without toxin: image showing a the evolution of the metabolites and enzymes in the μORAC setting. Each picture corresponds to a variable ordered as $(x_1, x_2, \beta_1, \beta_2, \mu)$.

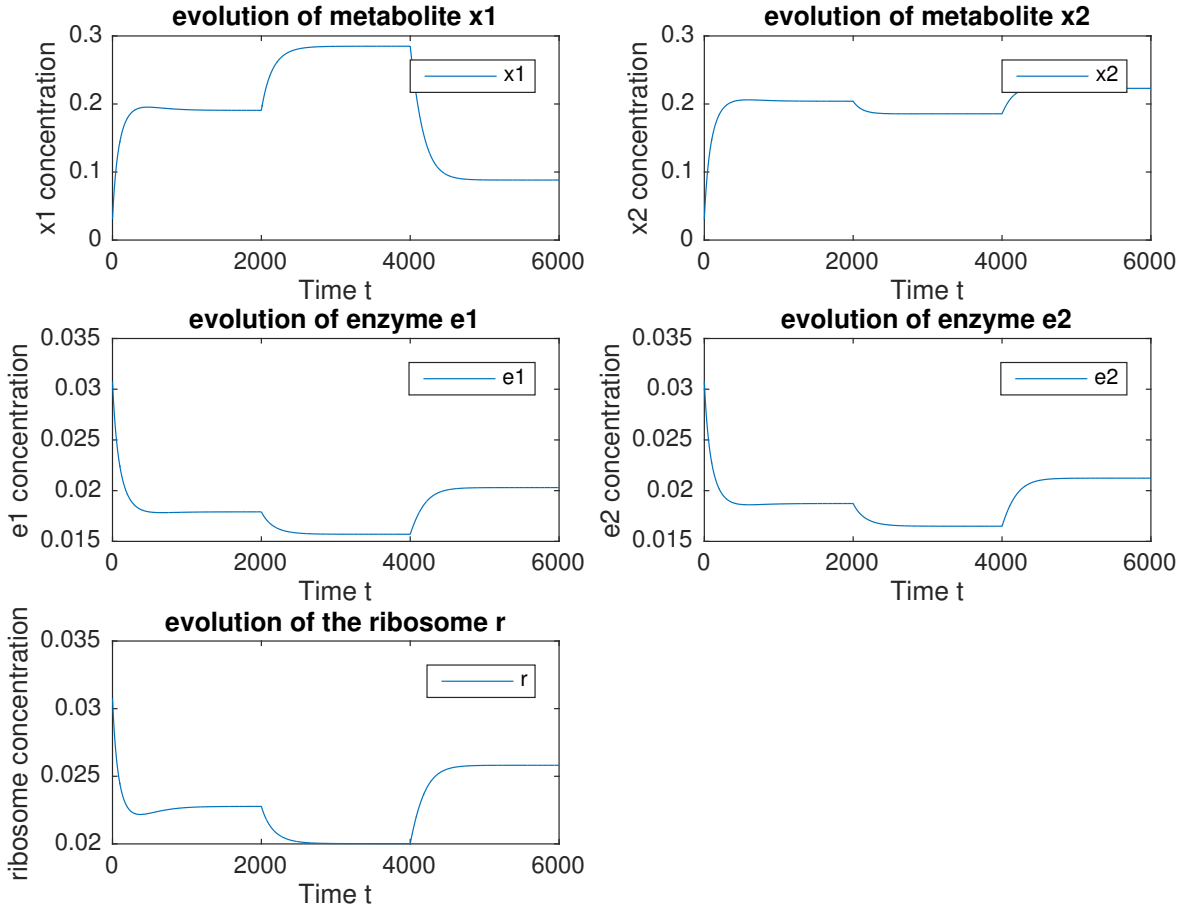


FIGURE 4.6: Example without toxin: image showing the evolution of the system in the μORAC setting for changing external concentrations of substrate S_1

system reaches is

$$\boldsymbol{q} = (x_1, x_2, e_1, e_2, r, \mu) = (0.1917, 0.2047, 0.0179, 0.0187, 0.0227, 0.0050).$$

We can see that the x_1, x_2 and μ values coincide with the ones we have found with the maximisation routine. This means that the system was able to steer itself to the maximum by continually updating the ribosome allocation for the production of the enzymes according to the sensor value. The system is able to do this using the continuation data we have supplied it with and that we assume the cell stores internally.

Image 4.6 shows what happens to the evolution of the system in the μORAC setting when we change external substrate concentration S_1. We have first run the system in the time range $[0, 2000]$ as in picture 4.5 where the S_1 concentration is kept at value $S_1 = 1$. We have then increased the external S_1 concentration to the value $S_1 = 2$ and run the system on the time range $[2000, 4000]$. After that we have decreased the S_1 value to $S_1 = 0.5$ and run the system again on the time range $[4000, 6000]$. We can see that the μORAC framework is able to explain adaptation of the cell to changing external conditions only using internal sensor metabolite concentrations for this example. In the last section of this chapter we will redo this using optimal data that we have calculated with a different method (we will call it the dynamic optimization). With the second method we will be able to supply the system with a dependancy function which covers a much bigger interval.

Image 4.7 is the same as image 4.6 where we have added asymptota to better appreciate the convergence to the steady state after each change in the exteral substrate S_1.

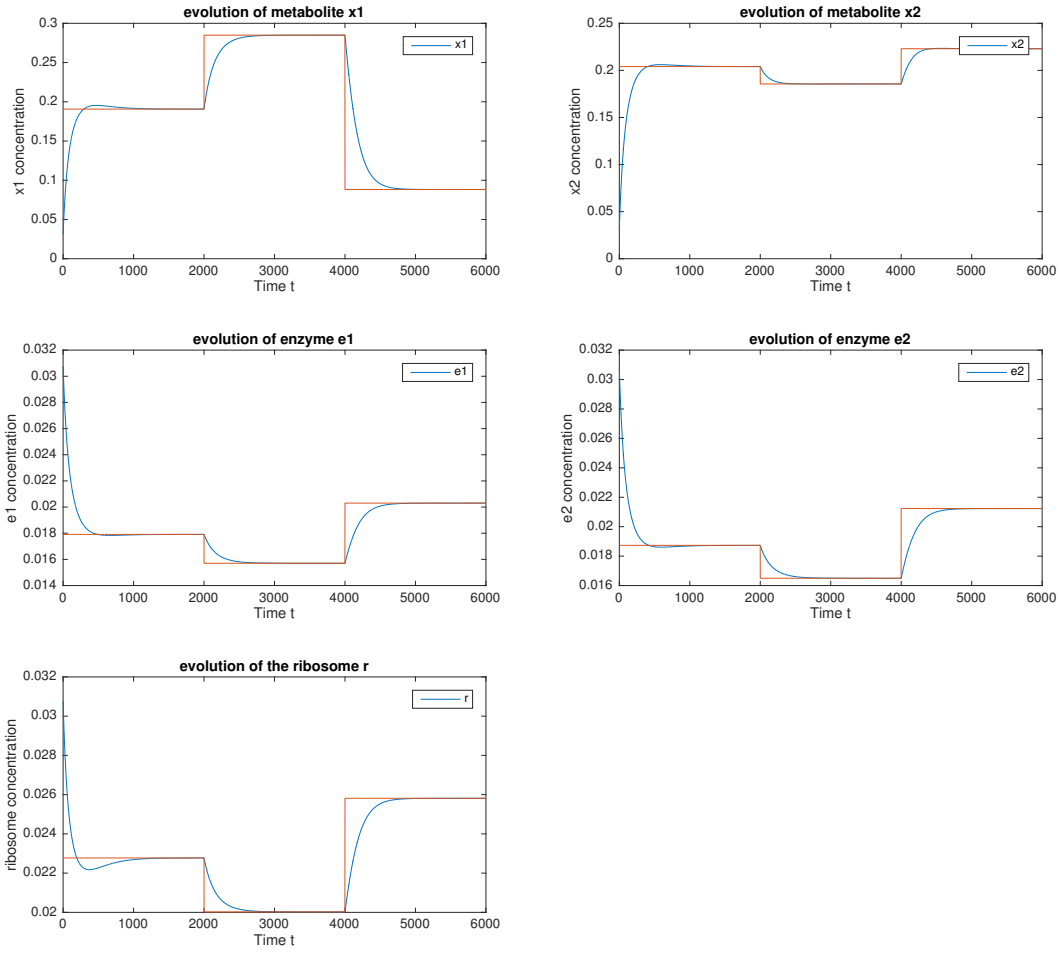


FIGURE 4.7: Example without toxin: image showing the evolution of the system in the μORAC setting for changing external concentrations of substrate S_1 with asymptota

4.2 Enzymatic Toxin Transport

For the first toxin example we have chosen the constants in the enzymes 'stoichiometry' matrix M as $M = (3,4,2,2;2,4,3,2;0,0,0,0)$. Also there are sixteen constants in this example: the constants that appear in the functions that describe the reactions, $\boldsymbol{k} = (k_{cat}, K_T, k_{f_1}, k_{f_2}, k_d, k_{g_1}, k_{g_2}, k_{g_3}, k_r)$ and the osmotic constants of the metabolites, enzymes and ribosomes: $\boldsymbol{\rho} = (\rho_1, \rho_2, \rho_3)$ and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3, \sigma_r)$. Again we will keep $k_{cat} = 0.25$ and use a perturbation of the other constants from the fixed values: $\boldsymbol{k} = (0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$, $\boldsymbol{\rho} = (1,1,1)$ and $\boldsymbol{\sigma} = (1,1,1,1)$. In particular the values that we use are:

$$\boldsymbol{k} = (0.2500, 0.4881, 0.5023, 0.4863, 0.4998, 0.4977, 0.5234, 0.5089, 0.5090)$$

$$\boldsymbol{\rho} = (0.9711, 1.0404, 1.0225)$$

$$\boldsymbol{\sigma} = (0.9992, 1.0311, 0.9748, 1.0167).$$

Also, we use both external substrate concentrations $S_1 = S_2 = 1$. When called with these parameters the optimisation routine gives the result vector

$$(x_1, x_2, T, \beta_1, \beta_2, \beta_3, \mu) = (0.7748, 0.7225, 0.1813, 0.0231, 0.0267, 0.0314, 0.0119).$$

We notice that in the optimal steady state the concentration of the toxin is much lower than the concentration of the other two metabolites (but not zero because the second substrate is necessary for growth so it has to be imported and consequently the toxin is also imported). Also the enzyme systhesis rate β_3 of the enzyme e_3 which removes the toxin is higher than the other two enzyme syntesis rates. The model was able to predict that in this case, in order to obtain maximal growth rate, the toxin is eliminated continuously from the cell by producing more toxin-removing enzyme. This result has been found with a grid precision of 0.00001 (which has then been refined with *fminsearch*).

The image of the optimum we have found, when plugged in the Balanced Growth (2.7) and Optimality Equations (2.8), in this case is $(0.0000, 0.0000, 0.0000, 0.0000, 0.1652, 0.0755, 0.0154)$. The first four entries of the vector are the image of the point found using the Balanced Growth Equations while the next three entries are the image of the point found using the Optimality Equations. We notice that also in this case all the entries of the image relative to the Balanced Growth Equations are zero up to the fourth decimal while the entries relative to the Optimality Equations are zero only up to the first decimal (in one case only up to the integer value). This is probably because in this case the matrices used in the Optimality Equations are bigger than in the previous case.

Next, Image 4.8 shows a plot of the maximization routine calculated on a $95 \times 95 \times 9$ grid of (x_1, x_2, T) values for the fixed constants above. For each point of the grid, hence for fixed metabolites values we have plotted the maximum μ value relative to this choice of metabolite concentrations.

As in the previous example, the plot of the H function calculated using the optimisation routine looks noisy and discontinous, in contrast with our expectation that the H function is smooth.

Next we report a plot obtained by running the system of differential equations. Image 4.9 shows the evolution of each variable in the system when the system is run 100 times, each time with the same initial condition $[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$ but with with random α_i.

Again we notice that, as in the previous example, for each set of random α_i the system reaches a steady state mainly in a monotone way. This is expected from the modeling equations and from the biological intuition but differs from the oscillating or chaotic evolution often displayed by

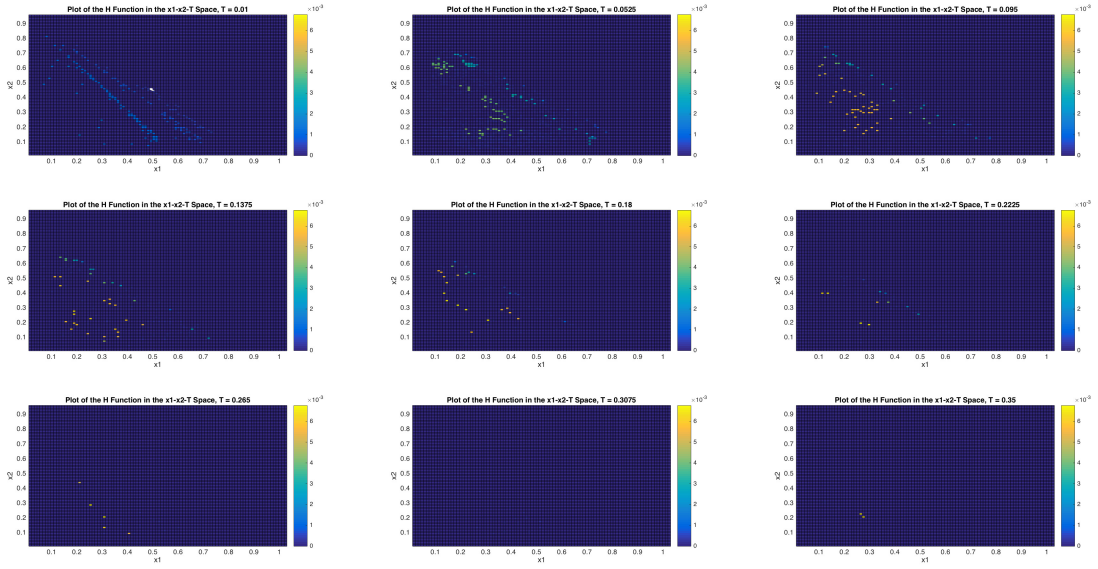


FIGURE 4.8: Enzymatic toxin transport case: image showing a surfaces plot of the optimisation routine function seen from above. For each image the x and y axis are x_1 and x_2 variables respectively. z axis is the growth rate. The sequence of images corresponds to increasing values of the toxin T.

non-linear differential equations systems. We can also see that for few sets of α_i the value of the ribosome in the steady state is bigger, these are choices of α_i close to the values that give optimal growth rate. As expected, these sets of values are more rare than the others.

4.3 Toxin Presence Leads To Substrate Switch

For this example we have chosen the constants in the enzymes 'stoichiometry' matrix M as $M = (3,2,4,1;2,1,3,1;4,3,1,2)$. Also in this example there are sixteen constants: the constants that appear in the functions that describe the reactions, $\boldsymbol{k} = (k_{cat}, k_{f_1}, k_{f_2}, k_{f_3}, k_{g_1}, k_{g_2}, k_{g_3}, k_r)$, and the osmotic constants of the metabolites, enzymes and ribosomes: $\boldsymbol{\rho} = (\rho_1, \rho_2, \rho_3)$ and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3, \sigma_r)$. Again we will keep $k_{cat} = 0.25$ and use a perturbation of the other constants from the fixed values: $\boldsymbol{k} = (0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$, $\boldsymbol{\rho} = (1,1,1)$ and $\boldsymbol{\sigma} = (1,1,1,1)$. In particular the values that we use are

$$\boldsymbol{k} = (0.2500, 0.4981, 0.5023, 0.4963, 0.4998, 0.4977, 0.5134, 0.5089, 0.5090)$$

$$\boldsymbol{\rho} = (1.0711, 1.0404, 1.0225)$$

$$\boldsymbol{\sigma} = (1.0992, 1.0311, 1.0748, 1.0167).$$

We then run the program for increasing values of T to see if we can observe adaptation. We would expect that, for increasing values of the toxin, the cell invests more resources to import the substrate which is not affected by the presence of the toxin. Also, we use external substrate concentrations levels at $S_1 = S_2 = S_3 = 1$.

Image 4.10 shows the results of the optimisation routine for 30 equally spaced increasing T values between 0 and 24 for the first set of g_i equations. According to the modeling equations, the toxin value 0 makes the reaction rate relative to the import of substrate S_1 5 times more effective than

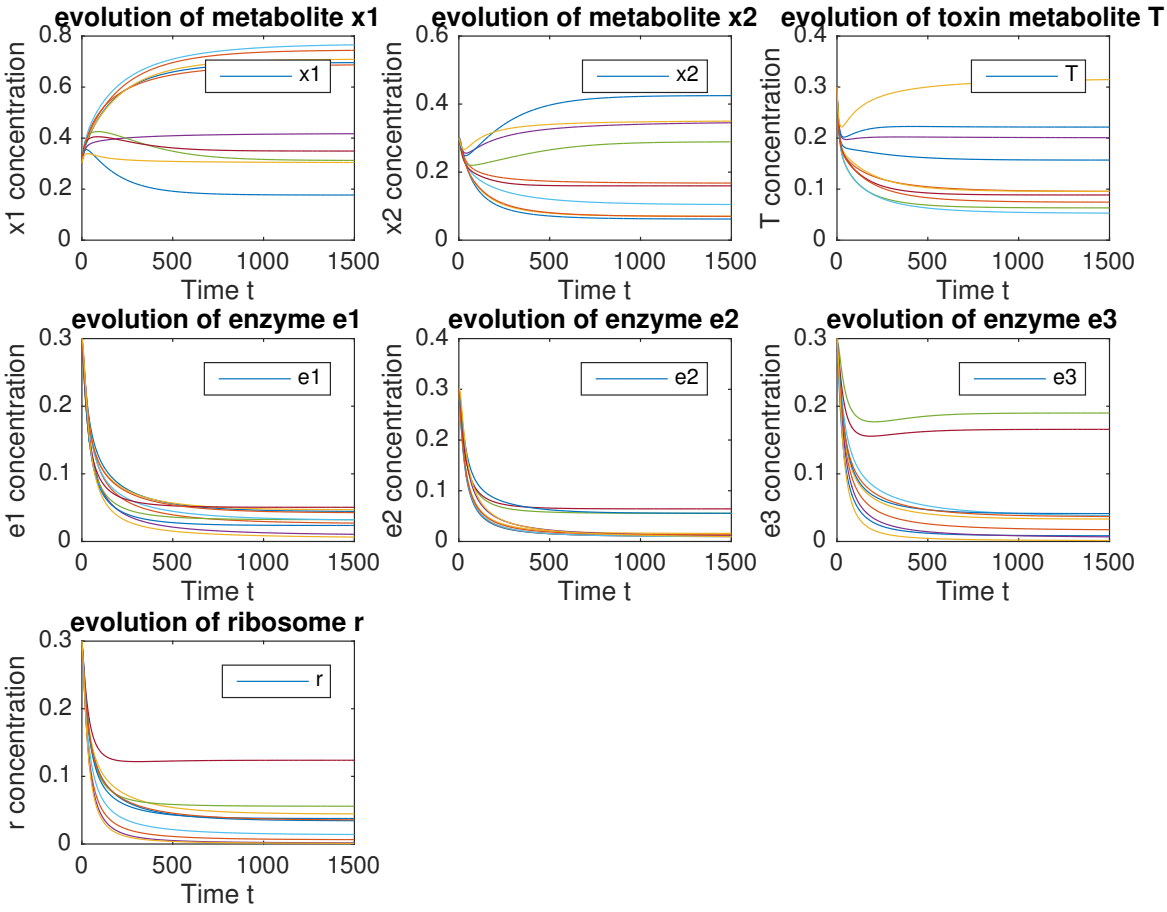


FIGURE 4.9: Enzymatic toxin transport case: image showing a the evolution of the metabolites and enzymes. Each picture corresponds to a variable ordered as $(x_1, x_2, T, \beta_1, \beta_2, \beta_3, \mu)$. The picture has been produced by running the diferential equations 10 times with random α's. We can see that, as expected and as in the previous example, the concentrations reach a steady state mainly in a monotone way, differently from the oscillating or chaotic evolution often displayed by non-linear differential equations systems

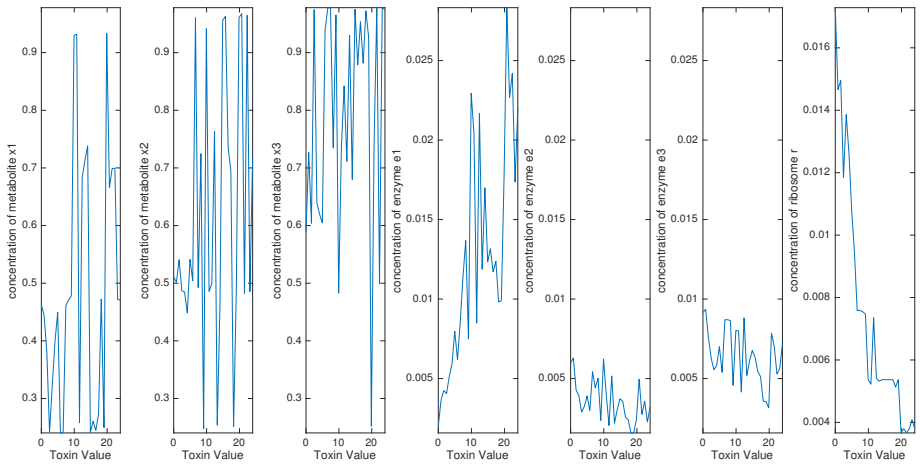


FIGURE 4.10: Substrate switch case: Image showing the results of the optimisation routine for 30 equally spaced increasing T values between 0 and 24 for the first set of g_i functions.

the others, a toxin level 4 makes this reaction rate equal to the others and a toxin level 24 makes this reaction rate $\frac{1}{5}$ less effective than the others. We have used different scales for the metabolite, enzyme and ribosome concentrations.

We notice that, for increasing toxin values, the concentrations of the metabolites x_1, x_2 and x_3 seem not to show any trend maybe because we have used not enough toxin values or probably because of the non reliability of the optimisation procedure. On the other hand, the concentration of enzyme e_1 increases for increasing toxin as expected. This is probably because, using the first set of g_i functions, the cell needs to import each of the three substrates in order grow. The growth rate however shows a decreasing trend, this is probably because as the cell invests more ribosome for the sythesis of enzyme e_1, it has less resources to invest in the production of the other two enzymes which in turn show a slightly decreasing trend.

Image 4.11 shows the results of the optimisation routine for 30 equally spaced increasing T values between 0 and 24 for the second set of g_i equations. We notice that as in the previous case the metabolite concentrations do not show any particular trend while the enzyme e_1 shows an increasing trend. Also in this case the growth rate shows a decreasing trend. This might suggest that the modeled cell adapts to different toxin levels but even if it can grow using only two substrates, it grows faster when all three are available. In both cases the cell shows adaptation by increasing the sythesis of enzyme e_1 for higher toxin concentrations.

Lastly, images 4.12, 4.13 and 4.14 show three plots of the function H on a $101 \times 101 \times 9$ grid respectively for toxin values 0, 4 and 24. In this case, the three plots seem to be even more noisy and discontinous and do not show any particular trend for increasing toxin value. This seem to give more evidence that the maximisation procedure we have used should not be trusted.

4.4 Checking The Results

In this section we report two tests we have conducted to check the reliability of our optimisation routine and to test our hypotesis that the maximisation routine we have used is not fully reliable. Here we show that in cases where the Linear Programming routine gives a positive growth rate (not very close to zero), the algorithm did in fact work correctly. This shows that we did implement the Balanced Growth Equations correctly, and that the errors must come from the underlying algorithm in *fminsearch*. Somehow this algorithm does not respect the convexity properties of the problem.

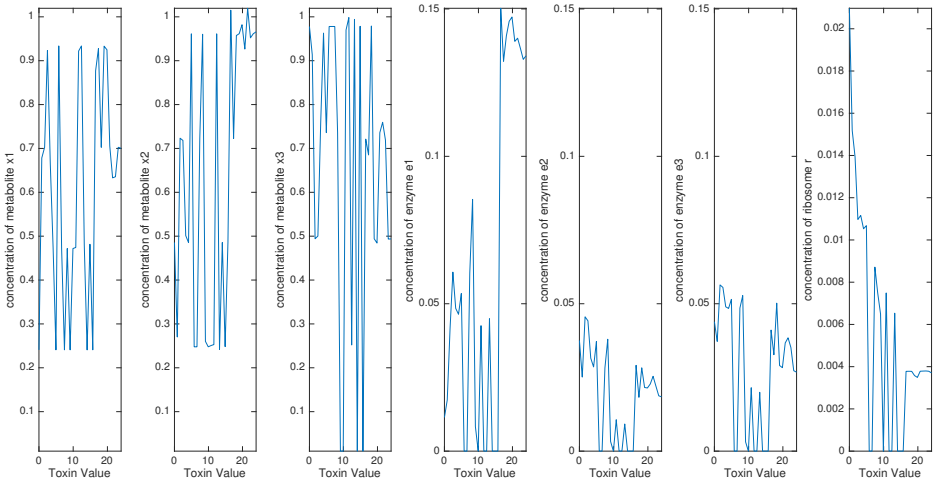


FIGURE 4.11: Substrate switch case: Image showing the results of the optimisation routine for 30 equally spaced increasing T values between 0 and 24 for the second set of g_i functions.

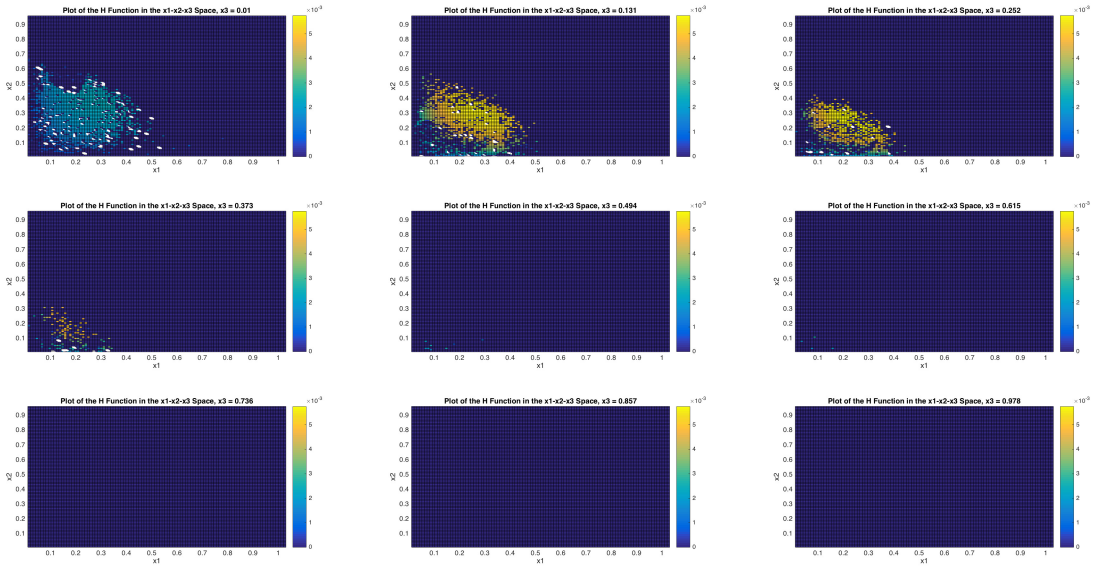


FIGURE 4.12: Substrate switch case: image showing the surface plot of the optimisation routine for toxin value 0

In the first test we have run the system with the initial conditions and ribosome allocation given by the optimum solution we have found. Image 4.15 shows the evolution of the system starting from the initial conditions and the α's given by the optimum. We can see that except for very slight oscillations due probably to numerical errors, the evolution of the system of differential equations confirms that the found optimum is indeed a steady state of the system (notice that the range of the oscillations is extremely small). This confirms that the result of our maximisation routine finds a solution of the Balanced Growth Equations.

Next, we report a test we have conducted which seems to suggest that the maximisation routine we have used is not fully reliable and that the H function is not noisy and discontinous as it appears calculated with the maximisation routine we have used. We have calculated the maximum growth rate μ for the first example in a different way. We have created a grid of (α_1, α_2) values and run the system of differential equations for each pair that give constant osmotic pressure. For

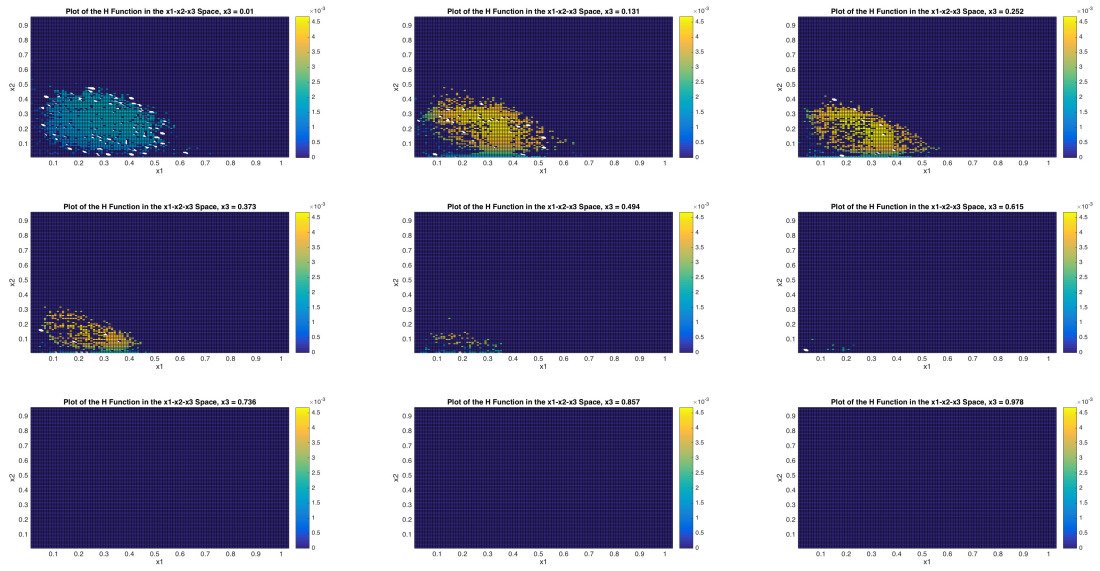


FIGURE 4.13: Substrate switch case: image showing the surface plot of the optimisation routine for toxin value 4

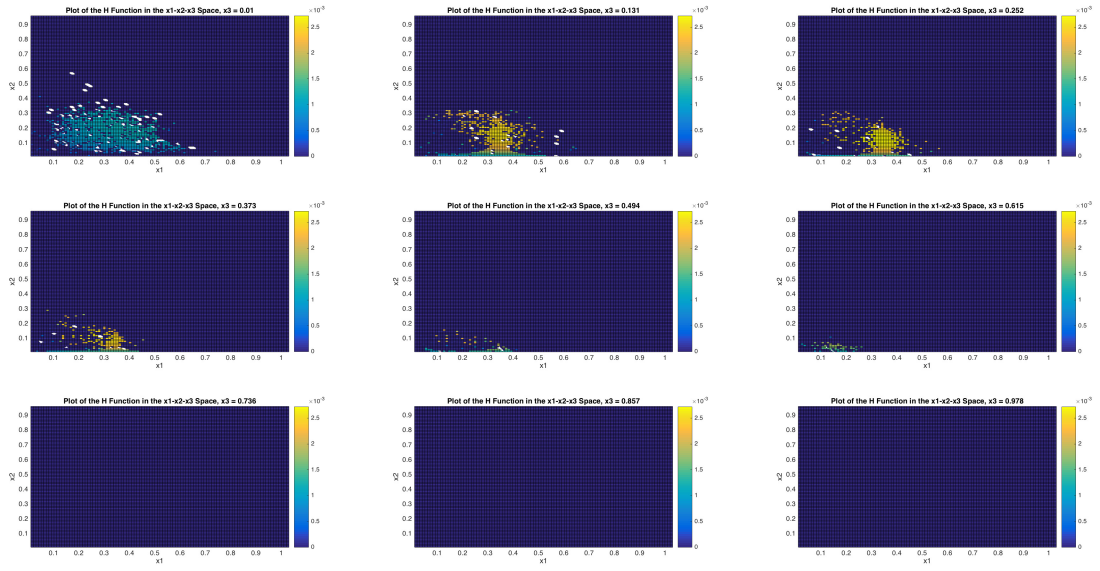


FIGURE 4.14: Substrate switch case: image showing the surface plot of the optimisation routine for toxin value 24

each pair we have have plotted the resulting growth rate against the x_1 and x_2 coordinates. Image 4.16 shows the result obtained using with a 50×50 grid of α values. When we calculate the growth rate in this way we find the maximum value is 0.0051 which coincides with the value we had found with the optimisation routine. This plot, which has been created using the dynamics of the system, is very different from the one we have calculated using the maximisation routine, in particular it is much smoother in accordance with our expectation that the growth rate μ changes continously with the metabolites concentrations $\boldsymbol{x}$. This test suggests that the H function is indeed continous and not noisy and it appeared so because of the algorithm that we have used to calculate it (the optimisation procedure). Moreover, if we compare the two plots, we can see that, in the plot created with the optimisation routine, the points which have value away from zero seem to follow the same shape of the plot calculated with the ODE dynamics. It seems that for a very

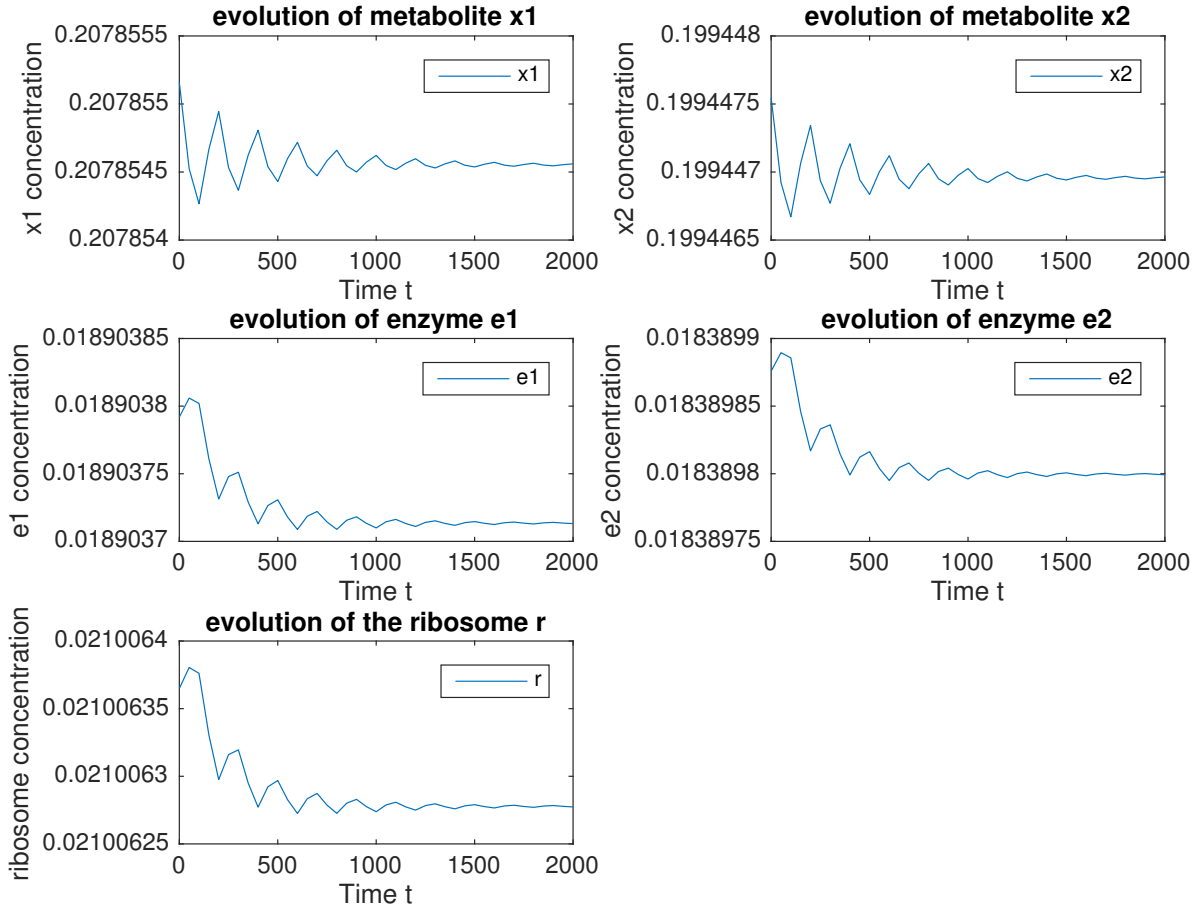


FIGURE 4.15: Example without toxin: image showing the evolution of the system starting from the initial conditions and ribosome allocation given by the optimum solution. The optimum is indeed a steady state of the system

big number of $\boldsymbol{x}$ combinations the optimisation procedure fails to calculate the correct value of μ. We suppose that the problem lies in the algorithm architecture. From this test we can conclude that the optimisation procedure we have attempted is not reliable and calculating the maximum growth rate μ using the ODE dynamics is a much more reliable method. However it should be mentioned that also this second way to calulate the maximum growth rate and the plot of the growth rate in the $\boldsymbol{x}$ space has its limitations. In particular it assumes that for every combination of α's, the ODE system has a unique steady state, which has not been proven for the general case.

4.5 Exploring the Dynamic Optimization

In the previous section we have found out that using the dynamics of the system, seems a much more reliable method for calculating the optimal values that correspond to the maximum growth rate (at least for the first example without toxin). In the light of this, in this section, we have further explored this method.

We have used a function that, for fixed set of $\boldsymbol{\alpha}$, runs the system of differential equations for fixed initial conditions and gives as output the values relative to the steady state that the system reaches. We have then used the inbuilt function *fminsearch* to maximize the growth rate given by this function.

The resulting algorithm proved out to be much more stable and its execution time is much shorter compared the static optimisation algorithm we have investigated in the previous sections. As we

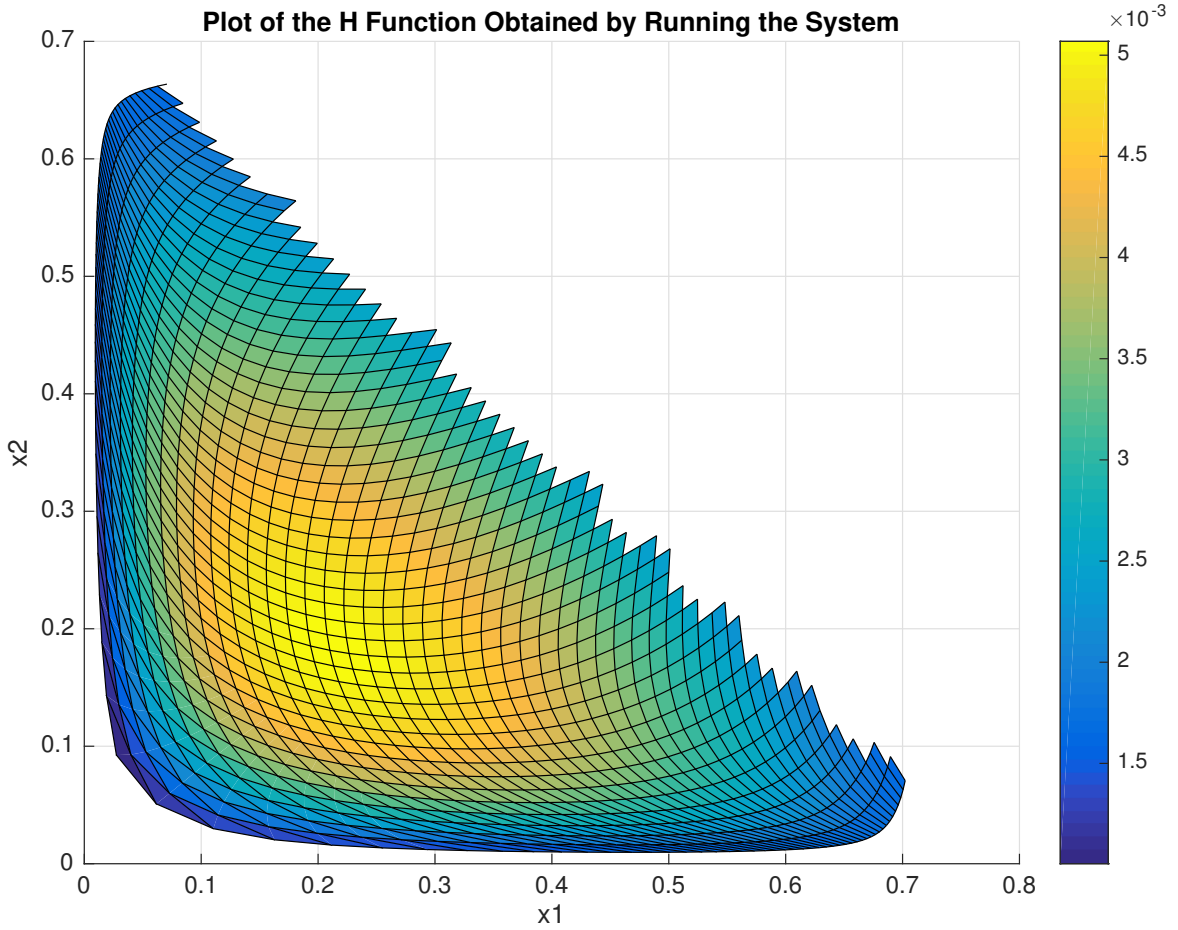


FIGURE 4.16: First example without toxin: plot of the growth rate μ in the (x_1, x_2) space calculated using the ODE dynamics

have seen from the previous plots (images 4.4, 4.9), when we have run the system for different combinations of $\boldsymbol{\alpha}$, the system of differential equations seems to always converge one steady state for each set of parameters. However we have found out that this algorithm seems to have also convergence issues which happen when the fluxes become too big or too small.

We have then used this algorithm to produce different graphs of the dependancy of the optimal substrate concentrations, the enzyme concentations, the ribosome concentration, the growth rate μ, the $\boldsymbol{\alpha}$ and the $\boldsymbol{\beta}$. Lastly we have used this data to implement the μORAC approach for the first example without toxin, as we did in section 4.1 but now using a bigger interval of optimal values.

In what follows we report and comment these plots.

4.5.1 Example Without Toxin

Image 4.17 shows the the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the substrate concentrations S_1 and S_2. We have considered values of S_1 and S_2 between 0.1 and 5. We notice that, as expected for increasing values of both substrates, the growth rate μ increases. Also we notice that when increasing the substrate concentration S_1, the enzyme allocation relative to the import of this substrate decreases, the same happens when increasing the substrate concentration S_2. Similarly when increasing the substrate concentration S_1, the enzyme e_1 which imports it decreases while the enzyme e_2 increases, the same happens when increasing substrate concentration S_2.

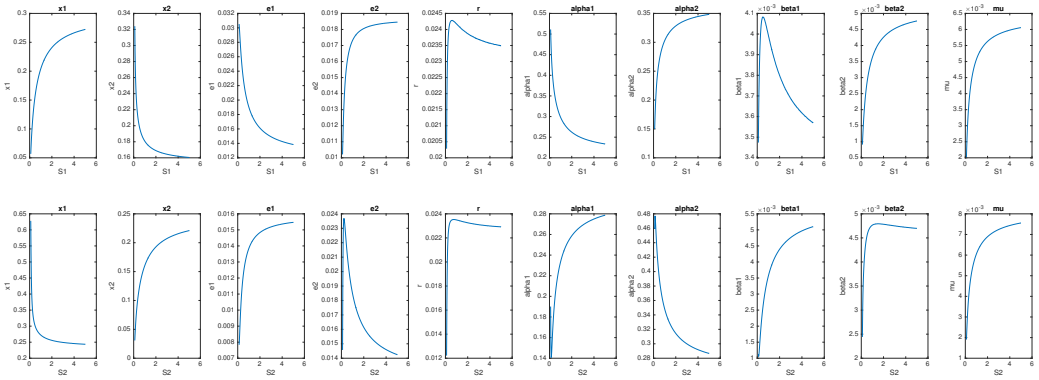


FIGURE 4.17: Example without toxin: image showing the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the substrate concentrations S_1 and S_2. We have considered values of S_1 and S_2 between 0.1 and 5.

Image 4.18 shows the the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the catalyzing constant k_{cat}. We have considered values of k_{cat} between 0.1 and 5. We can see that, as expected, when increasing k_{cat} the growth rate μ increases.

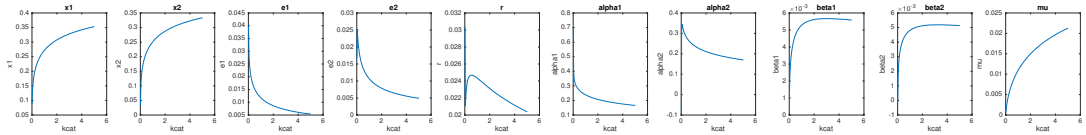


FIGURE 4.18: Example without toxin: image showing the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the catalyzing constant k_{cat}. We have considered values of k_{cat} between 0.1 and 5.

Image 4.19 shows the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the catalyzing constant $k1_f$. We have considered values of $k1_f$, between 0.1 and 5. We notice that for increasing values of $k1_f$ the growth rate decreases.

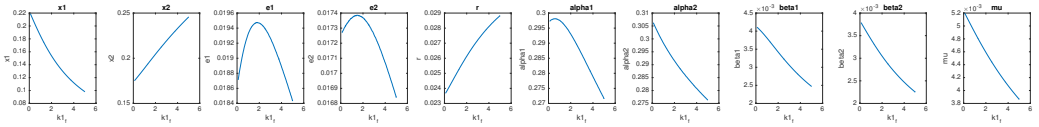


FIGURE 4.19: Example without toxin: image showing the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the catalyzing constant $k1_f$. We have considered values of $k1_f$ between 0.1 and 5.

Image 4.20 shows the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to the osmotic constant ρ_1. We have considered values of ρ_1 between 0.1 and 5. We notice that for increasing values of ρ_1 the growth rate decreases.

Lastly, Image 4.21 shows the result of running the system in the μORAC setting using the data of the dependancy on substrate S_1 that we have obtained with the dynamic optimization method. As in section 4.1, we have run the system first with $S_1 = 1$ on the time interval $[0, 2000]$, then

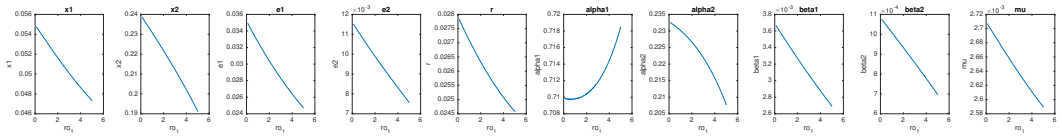


FIGURE 4.20: Example without toxin: image showing the dependancy of x_1, x_2, e_1, e_2, r, α_1, α_2, β_1, β_2 and μ respect to ρ_1, the osmotic constant relative to x_1. We have considered values of ρ_1 between 0.1 and 5.

we have rised the value to $S_1 = 2$ and run it on the time interval $[2000, 4000]$, then decreased the value to $S_1 = 0.5$ and run the system on the time interval $[4000, 6000]$. Each time we can see that the system steers itself to a new optimal state after the external substrate concentration has been changed.

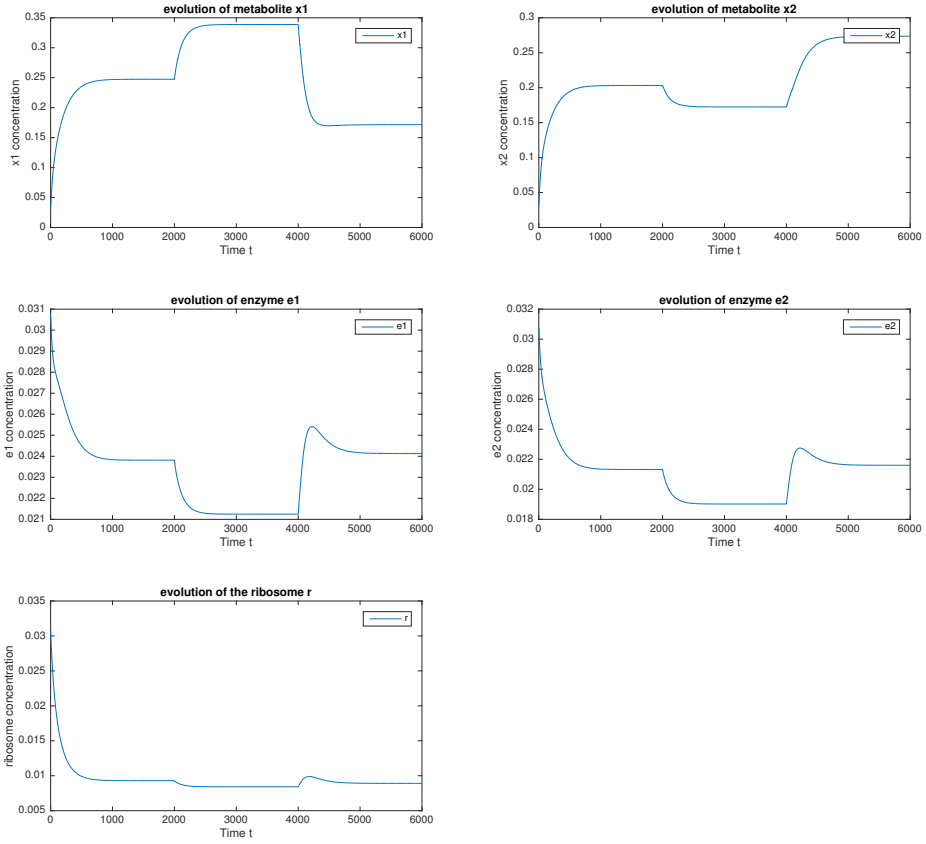


FIGURE 4.21: Example without toxin: image showing the result of running the system in the μORAC setting using the data of the dependancy on substrate S_1 that we have obtained with the new method.

4.5.2 Enzymatic Toxin Transport

Image 4.22 shows the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentration S_1. We have considered values of S_1 between 0.1 and 5. We notice that for nearly all values of S_1, the toxin value is lower than the other two metabolites x_1 and x_2.

Also the toxin removing enzyme e_3 is bigger than the other two, the same happens for the enzyme production allocation, α_3 is bigger than α_1 and α_2 for sufficiently big S_1 values. Lastly we notice that for increasing values of S_1 the growth rate μ also increases.

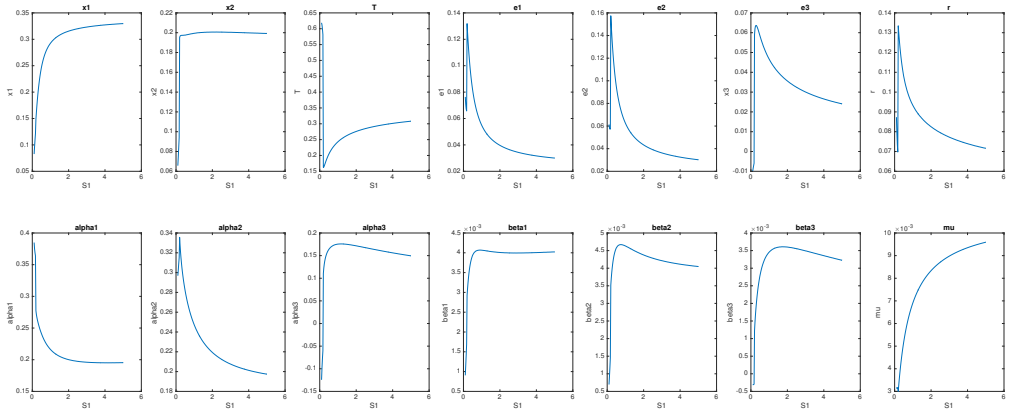


FIGURE 4.22: Enzymatic toxin transport case: image showing the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentration S_1. We have considered values of S_1 between 0.1 and 5.

Image 4.23 shows the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentration S_2. We have considered values of S_2 between 0.1 and 5. We notice that like in the previous case, for sufficiently big values of S_2 the toxin value is much lower than the other two metabolites x_1 and x_2. Also, the enzyme e_2 which imports the substrate S_2 becomes lower and the toxin removing enzyme e_3 stays high. Notice that, differently from the previous plot, the enzyme e_1 stays significantly higher, this is probably because the cell relies mostly on the import of S_1 to grow and does not need to use a lot enzyme e_2. Even using little enzyme to import S_2 and transform it into metabolite x_2, the cell is able to use the requires amount of metabolite x_2 necessary for growth when the substrate value S_2 becomes higher. We also notice that similarly to the previous plot, for increasing values of substrate S_2 the enzyme allocation changes and α_2 becomes small. Lastly we notice that as in the previous plot, for increasing values of S_2, the growth rate μ also increases.

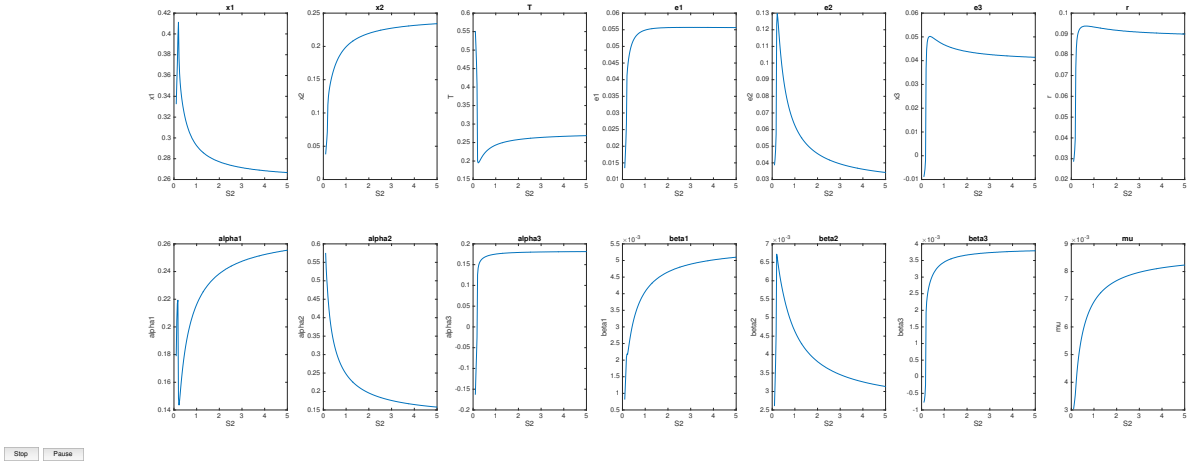


FIGURE 4.23: Enzymatic toxin transport case: image showing the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentration S_2. We have considered values of S_2 between 0.1 and 5.

Image 4.24 shows the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the catalyzing constant k_{cat}. We have considered values of k_{cat} between 0.05 and 0.65. For values of k_{cat} outside this interval the algorithm seems not to converge probably because the fluxes become too small and too big. We notice that, as in the previous plots, for sufficiently high values of k_{cat}

the toxin values stays lower than the other two metabolites x_1 and x_2. Also, we can see that the fraction α_3 of the ribosome responsible for the production of the toxin-removing enzyme e_3 stays higher than the other two. The enzyme e_3 is also significantly higher than the other two altougth, unexpectedly, we can see a decreasing trend also for this enzyme for increasing values of k_{cat}. Lastly, we notice that, as in the previous plots, the growth rate μ also increases for increasing values of k_{cat}.

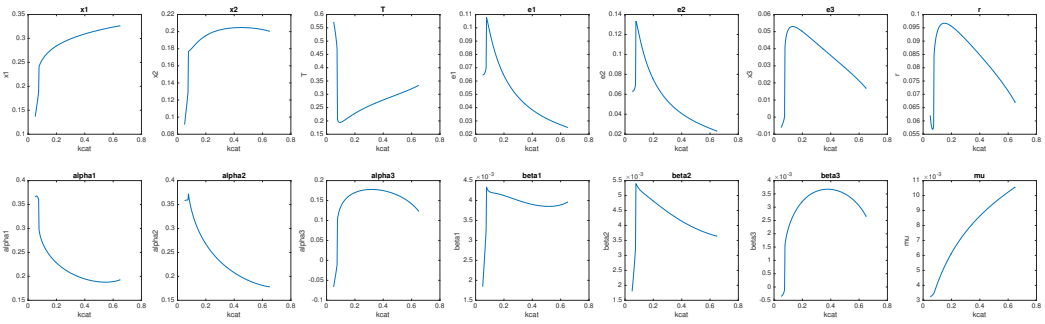


FIGURE 4.24: Enzymatic toxin transport case: image showing the dependancy of x_1, x_2, T, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the catalyzing constant k_{cat}. We have considered values of k_{cat} between 0.05 and 0.65. For values of k_{cat} outside this interval the algorithm seems not to converge.

4.5.3 Substrate Switch Case

Image 4.25 shows the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentrations S_1, S_2 and S_3 for toxin value $T = 4$ which makes the system symmetric. We have considered values of S_1, S_2 and S_3 between 0.1 and 5. We can see that, as expected, the three parts of the plot show symmetry. For increasing values of S_1, α_1 decreases, α_2 and α_3 increase, e_1 decreases, e_2 and e_3 increase. For increasing values of S_2, α_2 decreases, α_1 and α_3 increase, e_2 decreases, e_1 and e_3 increase. For increasing values of S_3, α_3 decreases, α_1 and α_2 increase, but e_2 increases, e_1 and e_3 decrease. As expected in all the three cases for each increasing substrate the growth rate μ increases.

Image 4.26 shows the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the toxin T. We have considered values of the toxin between 0 and 18. Remind that for values of the toxin between 0 and 4 the toxin presence makes the flux relative to the import of substrate S_1 bigger than the other two, while for toxin values higher than 4 the toxin makes the same flux smaller and smaller respect to the other two. For values of the toxin higher than approximately 18 the flux becomes too small and the algorithm seems not to converge. We notice that, as expected, for increasing values of the toxin the growth rate μ decreases. We also notice that, as expected, the enzyme which imports the first substrate S_1 increases, probably because the presence of the toxin makes the corresponding flux smaller. The same goes for the corresponding α_1 and β_1.

Image 4.27 shows the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the catalyzing constant k_{cat} for three different toxin values: 0, 4 and 24 when keeping the substrate concentrations fixed as usual at value $S_1 = S_2 = S_3 = 1$. We considered values of k_{cat} between 0.1 and 5. We notice that for a few values of k_{cat} the algorithm seems to find a wrong solution, however these seem to be really exceptional cases considering that we have used 200 discretization points. We also notice that for all three values of the toxin T, the growth rate μ increases for increasing values of k_{cat}.

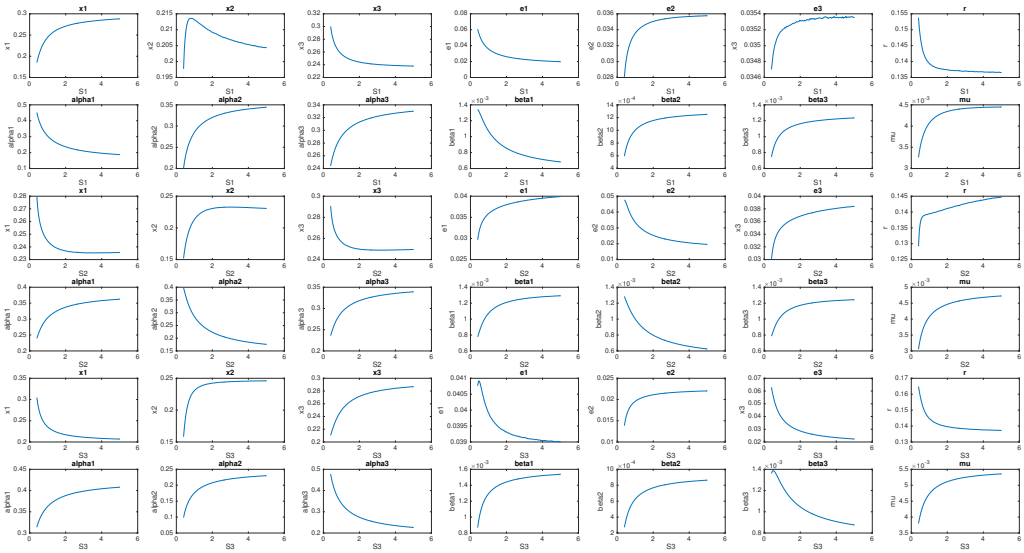


FIGURE 4.25: Substrate switch case: image showing the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the substrate concentrations S_1, S_2 and S_3 for toxin value $T = 4$ which makes the system symmetric. We have considered values of S_1, S_2 and S_3 between 0.1 and 5.

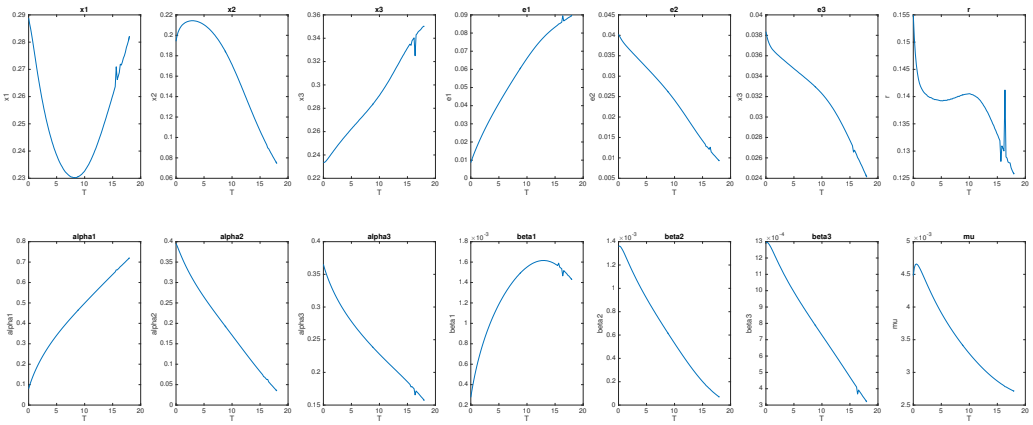


FIGURE 4.26: Substrate switch case: image showing the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the toxin T. We have considered values of the toxin between 0 and 18. Remind that for values of the toxin between 0 and 4 the toxin presence makes the flux relative to the import of substrate S_1 bigger than the other two, while for toxin values higher than 4 the toxin makes the same flux smaller and smaller respect to the other two. For values of the toxin higher than approximately 18 the flux becomes too small and the algorithm seems not to converge.

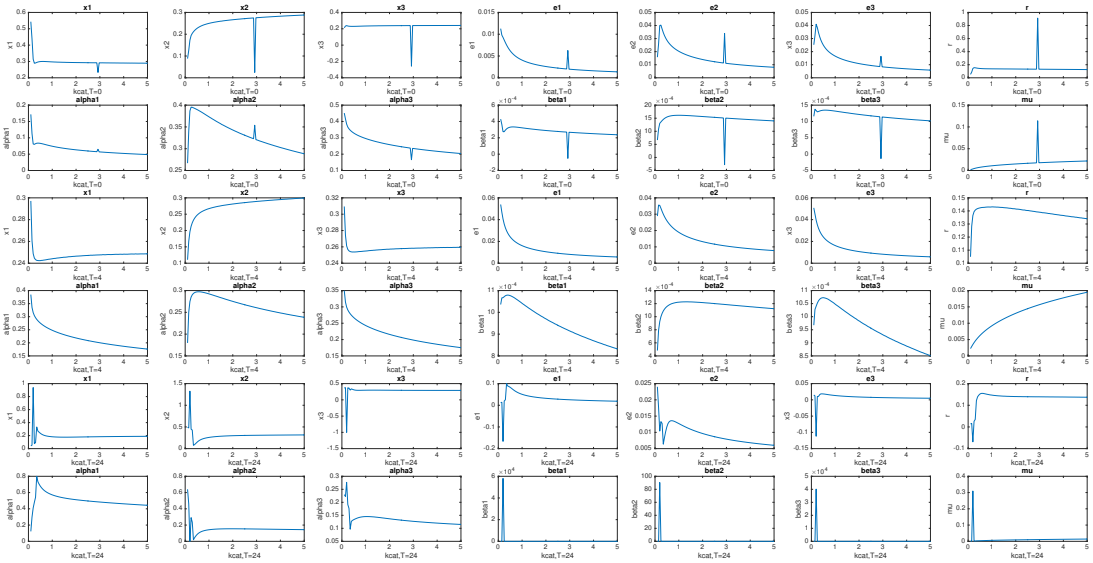


FIGURE 4.27: Substrate switch case: image showing the dependancy of x_1, x_2, x_3, e_1, e_2, e_3, r, α_1, α_2, α_3, β_1, β_2, β_3 and μ respect to the catalyzing constant k_{cat} for three different toxin values: 0, 4 and 24 when keeping the substrate concentrations fixed as usual at value $S_1 = S_2 = S_3 = 1$. We considered values of k_{cat} between 0.1 and 5.

Chapter 5

Code Description

In this section we briefly describe the code and what it does. We describe only the code for the non-toxin case as the other cases are similar. The code consists of 13 main functions and 3 auxiliary functions and one script. The auxiliary functions are `cont.m` and `multinewton.m` (courtesy of professor J.B. van Den Berg), which continue the solution of a set of algebraic equations and implements the multi-dimensional Newton method respectively and a small function `ran.m` that creates a random number centred at zero (a number between $-a$ and a).

When running the functions, the first one to be run is the script `Constants_set` which sets all the constants of the problem. The code for the substrate switch example is slightly different than the others as the constants are defined inside the main function `SOL.m`. In the script it is possible to choose if we want to use some fixed constants or some perturbed ones. It is important to call both commands `clear` and `clear global` if we want to run the functions again with different constants.

The main functions that solve the problem are: `optH.m`, `fixed_pt.m`, `Maximizer.m`, and `SOL.m`. These functions together form the maximization routine. In particular, `optH.m` solves the linear program for fixed $\tilde{\mu}$ and so maximises μ for fixed metabolites concentrations and fixed $\tilde{\mu}$. `fixed_pt.m` is a small functions that uses `optH.m` to find the fixed point of the function H and thus find the optimal growth rate for fixed metabolites concentrations.

Then we have the two functions, `Maximizer.m` and `SOL.m` which solve the problem. `Maximizer.m` is a routine that maximizes the growth rate and that uses `fixed_pt.m` and `optH.m` to optimize the solution over the metabolites concentrations. A solution is a set of metabolites and respective enzymes allocation. However we need to find the optimum only over the metabolites variables as the enzymes allocation is dependent on the metabolites concentrations. in order to do this we use first the grid routine and then we refine the solution using the inbuilt function *fminsearch* (this is a minimising function so we use it to minimise $-\mu$ and thus maximise μ).

The function *SOL.m* is then the function that find the optimal growth rate. It takes as input the desired maximum number of iterations of *fminsearch* and the desired precision of the grid method.

The function *contG.m* takes as input a solution point and calculates both the Balanced Growth and Optimality Equations. For the Optimality Equations we don't calculate the determinant of the matrices but instead give as output the values of the smallest eigenvalue which is an equivalent but computationally more stable way to calculate the result.

The function `visualH.m` creates and visualises the function which has as input the metabolites concentrations and as output the growth rate (this is the same function that we calculate on the grid, only this time on a bigger grid in order to plot it).

The other functions are: `ContG_S1` which is similar to `ContG.m` and is used to continue the solution respect to the S_1 substrate. `input_output.m` is the function that creates the continuation dataset which is used in the μORAC system. `run_set.m` is the function that we used to calculate the growth rate in the second background test. `Simple_Example_System.m` calculates the right hand side of the system of differential equations and `run_system.m` runs the system of differential equations for chosen α's. We have also the functions `Simple_Example_System_morac.m` and

`Run_System_morac.m` which respectively calculate the right hand side of the differential equations and run the system of differential equations in the μORAC setting.

Lastly, for the dynamic optimization section we have used two functions and a script: `cal_fixed_alphas.m` which takes as input a set of $\boldsymbol{\alpha}$ and gives as output the steady state the system reaches when run from a fixed initial condition, `DummyFunction1.m` which extracts the growth rate from the function in order to use *fminsearch* to maximise it and `run_script`, a script used to produce the dependancy plots presented in the relative section.

We summarise next the functions of which the program is composed and how the are called:

- `Constants_Set` is the script used to set all the constants of the problem.
- `[mu_opt,exitflag,b_s_opt] = optH(x1,x2,mu_tilde)` calculates the optimum for fixed $\tilde{\mu}$. It takes as input the fixed metabolites concentrations x_1 and x_2 and $\tilde{\mu}$ and gives as output the relative β's of the optimal solution for these fixed values, we remind that the β's say how the ribosome is allocated among the enzymes. It also gives back the exitflag of the linear programming function *linprog*
- `[s]=fixed_pt(x1,x2)` finds the fixed point where $\mu = \tilde{\mu}$. It then takes as input the metabolites concentrations x_1 and x_2 and gives as output the value of the optimal growth rate μ
- `[B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH,H] = Maximizer(Substr_con,x1_range,x2_range,max_iter,precision)` is the main maximisation routine, it takes as input the substrates concentrations, the intervals of x_1 and x_2 where the maximisation over the metabolites concentrations take place. The maximum number of iterations of *fminsearch* and the precision of the grid routine used to find the point to be supplied to *fminsearch*. It gives as output the β's of the optimal solution over x_1 and x_2, the exit flag of *fminsearch* and the exitflag of `optH` (which is the exitflag of *linprog*.
- `[x0,k_s,ro_s,sigma_s] = SOL(max_iter,precision,method,in_point)` is finally the main problem solving function that uses all the previous ones. `max_iter` is again the maximum number of iterations of *fminsearch*. The optimum is at first searched with the grid method and then further improved using *fminsearch*. After that the optimum which has been found is checked for optimality using the function `contG.m` which calculates the Balanced Growth Equations and the Optimality Equations. After that, in order to continue the solution, we try to find a point which is close to the found optimum but which is a better zero of `contG.m`, to do this we use the inbuilt MATLAB functions *gsolve* or *lsqnonlin*, if `method = 'gs'` then *gsolve* is used, otherwise, if `method = 'ls'`, *lsqnonlin* is used. Lastly, if the number of inputs is 4, the grid method is skipped and *fminsearch* is supplied directly with an initial point `in_point`. The function `SOL.m` also prints to screen the results and some of the intermediate values it calculates.
- `[z] = contG(x)` calculates the Balanced Growth and Optimality Equations of the input vector.
- `[H,k_s,ro_s,sigma_s,x1_s,x2_s] = visualH(n,H_in,X1_RANGE,X2_RANGE,visu)` is used to visualise the function that associates the optimal growth rate to each fixed metabolite pairs, that is, it's the function we want to maximise using the grid method and then using *fminsearch*. If the number of inputs is 2, the function creates a matrix of the function values, n is a vector of two integers that are the number of discretisation points for both the ranges of $x1$ and $x2$, in this case the range of visualization is the whole rectangle of the the metabolites values for which both fluxes are positive. If the number of inputs is 3, the function plots the matrix that we provide as input (the one that we created with another call to the function). If the number of inputs is 4, the functions creates again the matrix but for given ranges of the metabolites (used to magnify a specific region of variables). If the number of inputs is 5 it plots the matrix with given ranges (the ones that we specified in the previous call).

- `[dx]=Simple_Example_System(t,x,alpha_s)` is the function that calculates the right hand side of the differential equations system for a chosen set of α's.
- `[q,MU] = Run_System(alpha_s,xinit)` runs the system of differential equations, if the number of inputs is 0 the α's are randomly chosen and the initial point is chosen to be $[0.1, 0.1, 0.1, 0.1, 0.1]$. Otherwise both the α's and inital point can be supplied when the number of inputs is 2.
- `[dx]=Simple_Example_System_morac(t,x)` is the function that calculates the right hand side of the differential equations system in the μORAC setting.
- `[q,MU] = Run_System_morac(xinit)` runs the system of differential equations in the μORAC setting.
- `[x1,x2,mu] = run_set(N)` calculates the graph of the growth rate over the α's space for the second background test.
- `zopts = input_output(z)` takes as input the solution point and gives as output the matrix of continuation points.
- `[r] = contG_S1(x)` takes as input the solution vector which includes in this case the substrate variable S_1 and calculates the value of the Balanced Growth and Optimality equations for this point. This function is used for the continuation.

Chapter 6

Conclusions

In this thesis we have introduced a model for bacteria metabolism and investigated it numerically. Several examples of the model have been investigated. We have developed a routine which finds the optimal solution of metabolite and enzyme concentrations which correspond to the maximum growth rate. Our goal was to gain insight on whether it is possible to understand stress management in cells using a growth rate maximization approach.

We have found that the optimisation procedure is able to find the maximum growth rate and it coincides with the maximum growth rate that we have calculated using the ODE dynamics. However, evaluating the plot of the growth rate calculated with the optimisation routine and checking it against our expectation of smoothness and against the plot that we have calulated using the ODE dynamics, we can conclude that the numerical results we have found should not be trusted and the optimization procedure is not fully reliable. We have double checked the correctness of the results for fixed metabolites concentrations. We can conclude that using the ODE dynamics to calculate the maximum growth rate is a much more reliable method. In the last section of the results chapter we have investigated this second method that uses the dynamics of the system to calculate the optimal values that lead to the maximum growth rate. This second method proved out to be not only more reliable and stable but also computationally more effective.

However, this model and the procedure we have introduced was able to give insight on our initial research idea to unite a stress management setting with a growth rate optimisation one. We have modeled a first example without toxin and three examples which include the presence of a toxin. As expected, we have seen that the cell removes the toxin and invests more energy in the production of the toxin-removing enzyme in the first toxin case (enzymatic toxin transport).

We have introduced the μORAC framework and implemented it for the first example. We have seen how this approach is able to model how the cell steers itself to the optimal steady state only using internal sensor information and the continuation data that we assume is stored inside the cell. Because of lack of time this is the only example where we have implemented the μORAC framework.

This study has showed the versatility of this model and how it is able to predict cell adaptation to stress factors such as toxins and to changing environmental conditions. The results of this study seem to suggest that it is possible to gain insight on how the cell deals with stress factors using a fitness perspective. Both growth and stress management are part of bacterial life and the cell uses feedback mechanisms to choose how to allocate resources between the two according to the external conditions.

Further research might include the study of examples where the μORAC framework is applied to an example which models the presence of a toxin or the modeling of other discontinous stress factors such as damage caused by heat or radiation. Also, the discontinuity of the maximisation function should be further investigated. Further research could also consider examples with more variables and model metabolic networks which contain more than one EGM.

Appendix A

MATLAB codes

A.0.1 Example Without Toxin Codes

Constants_set.m

```
global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

rndmz = 'N';

e = 0.05;

Network = [1,0;
0,1];

Enz_con = [2,4,3;
3,4,2];

Substr_con = [1 1];


if rndmz == 'Y'

%disp('Vector of randomization:')

rad = [ran(e) , ran(e) , ran(e) , ran(e) , ran(e) ,...
 ran(e) , ran(e) , ran(e) , ran(e) , ran(e)];

%disp('Original constants:');

k_s = [0.25,0.5,0.5,0.5,0.5,0.5];

ro_s = [1,1];

sigma_s = 10*[1,1,1];

disp('Perturbed values:');

k_s = [0.25,0.5+rad(1)/2,0.5+rad(2)/2,0.5+rad(3)/2,0.5+rad(4)/2,0.5+rad(5)/2];

ro_s = [1+rad(6),1+rad(7)];

sigma_s = 10*[1+rad(8),1+rad(9),1+rad(10)];

else

%disp('Using fixed constants');

k_s = [0.2500  ,  0.4785  ,  0.5173 ,  0.5164  ,  0.4623  ,  0.4907];

```

```
ro_s = [0.9775  ,  1.0217];

sigma_s = 10*[0.9783  ,  1.0396  ,  1.0327];

end

%%%%%%%%%%%%%%%%%%%% CONSTANTS %%%%%%%%%%%%%%%%

M11 = Enz_con(1,1);
M12 = Enz_con(1,2);
M1r = Enz_con(1,3);
M21 = Enz_con(2,1);
M22 = Enz_con(2,2);
M2r = Enz_con(2,3);

P11 = Network(1,1);
P12 = Network(1,2);
P21 = Network(2,1);
P22 = Network(2,2);

kcat = k_s(1);
k1_f = k_s(2);
k2_f = k_s(3);
k1_g = k_s(4);
k2_g = k_s(5);
kr_g = k_s(6);

ro_1 = ro_s(1);
ro_2 = ro_s(2);

sigma_1 = sigma_s(1);
sigma_2 = sigma_s(2);
sigma_r = sigma_s(3);

S1_c = Substr_con(1);
S2_c = Substr_con(2);

a1=ro_1*P11+ro_2*P21;
a2=ro_1*P12+ro_2*P22;

b1=ro_1*M11+ro_2*M21;
b2=ro_1*M12+ro_2*M22;
br=ro_1*M1r+ro_2*M2r;
```

optH.m

```
 function [mu_opt,exitflag,b_s_opt,Aeq,exit]=optH(Substr_con,x1,x2,mu_tilde);

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

S1 = Substr_con(1);
S2 = Substr_con(2);

syms mu beta_1 beta_2

f1 = @(x1,S1) kcat*((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2) kcat*((S2-k2_f*x2)/(1+S2+x2));

g1 = @(x1,x2) k1_g*x1*x2/(1+x1+x2+x1*x2);
g2 = @(x1,x2) k2_g*x1*x2/(1+x1+x2+x1*x2);
gr = @(x1,x2) kr_g*x1*x2/(1+x1+x2+x1*x2);

%F1_mod = @(beta_1,beta_2,mu)   x1*(  (a1*f1(x1,S1)/mu_tilde +...
% sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu_tilde+sigma_2-b2)*beta_2 + (sigma_r-br)*mu  )...
```

```
%- (P11*f1(x1,S1)/mu_tilde-M11)*beta_1 - (P12*f2(x2,S2)/mu_tilde-M12)*beta_2 + M1r*mu;
%F2_mod = @(beta_1,beta_2,mu)   x2*(  (a1*f1(x1,S1)/mu_tilde +...
%sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu_tilde+sigma_2-b2)*beta_2 + (sigma_r-br)*mu  )...
%- (P21*f1(x1,S1)/mu_tilde-M21)*beta_1 - (P22*f2(x2,S2)/mu_tilde-M22)*beta_2 + M2r*mu;

%Fr = @(beta_1,beta_2,mu) beta_1/g1(x1,x2) + beta_2/g2(x1,x2) + mu/gr(x1,x2);

f = [0,0,-1];
B = -eye(3);
b = [0;0;0];
beq = [0;0;1];

Aeq = zeros(3);

Aeq(1,1) = x1*(a1*f1(x1,S1)/mu_tilde + sigma_1-b1) - (P11*f1(x1,S1)/mu_tilde-M11);
Aeq(1,2) = x1*(a2*f2(x2,S2)/mu_tilde + sigma_2-b2) - (P12*f2(x2,S2)/mu_tilde-M12);
Aeq(1,3) = x1*(sigma_r-br) + M1r;

Aeq(2,1) = x2*(a1*f1(x1,S1)/mu_tilde + sigma_1-b1) - (P21*f1(x1,S1)/mu_tilde-M21);
Aeq(2,2) = x2*(a2*f2(x2,S2)/mu_tilde + sigma_2-b2) - (P22*f2(x2,S2)/mu_tilde-M22);
Aeq(2,3) = x2*(sigma_r-br) + M2r;

Aeq(3,1)=1/g1(x1,x2);
Aeq(3,2)=1/g2(x1,x2);
Aeq(3,3)=1/gr(x1,x2);

Aeq;

options = optimset('Display','none','maxiter',100);

LB = [];
UB = [];

[b_s_opt,fval,exitflag,output] = linprog(f,B,b,Aeq,beq,LB,UB,[],options);

if exitflag == 1 || exitflag == 0
mu_opt=b_s_opt(3);
exit = 1;
else
mu_opt=0;
b_s_opt = [0;0;0];
exit = 0;
end

end
```

fixed_pt.m

```
function [s]=fixed_pt(Substr_con,x1,x2);

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

fixedpt = @(mu_tilde) optH(Substr_con,x1,x2,mu_tilde) - mu_tilde;
options = optimset('TolX',1e-7,'TolFun',1e-7,'maxiter',250);
s = -fzero(fixedpt,1e-9,options);
```

Maximizer.m

```
function [B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH,H] =...
Maximizer(Substr_con,x1_range,x2_range,max_iter,precision)

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
```

```
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

n=5;

count = 0;

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);

x1_s = linspace(a_x1,b_x1,n);
x2_s = linspace(a_x2,b_x2,n);

H = zeros(n,n);

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2));

f = waitbar(0,'please wait..');

for i=1:n
for j=1:n
if ro_1*x1_s(i)+ro_2*x2_s(j) > 1
H(i,j)=0;
count=count+1;
f = waitbar(count/25,f);
else
H(i,j) = opt_M([x1_s(i),x2_s(j)]);
count = count + 1;
f = waitbar(count/25,f);
end
end
end

close(f)

count = 0;

[M,I] = min(H(:));
[I_row, I_col] = ind2sub(size(H),I);

disp('Point found from the grid:')

y = [x1_s(I_row),x2_s(I_col)]

disp('With minimum:')

optm = opt_M(y)

disp('And precision:')

pres = max(abs(a_x1-b_x1),abs(a_x2-b_x2))/4

while  pres > precision

if I_row == 1
a_x1 = x1_s(I_row);
b_x1 = x1_s(I_row+1);
else
if I_row == n
a_x1 = x1_s(I_row-1);
b_x1 = x1_s(I_row);
else
a_x1 = x1_s(I_row-1);
b_x1 = x1_s(I_row+1);
end
end
```

```
73
74
75  if I_col == 1
76  a_x2 = x2_s(I_col);
77  b_x2 = x2_s(I_col+1);
78  else
79  if I_col == n
80  a_x2 = x2_s(I_col-1);
81  b_x2 = x2_s(I_col);
82  else
83  a_x2 = x2_s(I_col-1);
84  b_x2 = x2_s(I_col+1);
85  end
86  end
87
88  x1_s = linspace(a_x1,b_x1,n);
89  x2_s = linspace(a_x2,b_x2,n);
90
91  H = zeros(n,n);
92
93  f = waitbar(0,'please wait..');
94
95  for i=1:n
96  for j=1:n
97  if ro_1*x1_s(i)+ro_2*x2_s(j) > 1
98  H(i,j)=0;
99  count=count+1;
100 f = waitbar(count/25,f);
101 else
102 H(i,j) = opt_M([x1_s(i),x2_s(j)]);
103 count = count + 1;
104 f = waitbar(count/25,f);
105 end
106 end
107 end
108
109 close(f)
110
111 [M,I] = min(H(:));
112 [I_row, I_col] = ind2sub(size(H),I);
113
114
115 disp('Point found from the grid')
116
117 y = [x1_s(I_row),x2_s(I_col)]
118
119 disp('With minimum:')
120
121 optm = opt_M(y)
122
123 disp('And precision:')
124
125 pres = max(abs(a_x1-b_x1),abs(a_x2-b_x2))/4
126
127 count = 0;
128
129 end
130
131 disp('fminsearch starts')
132
133 options = optimset('Display','iter','PlotFcns',@optimplotfval,...
134 'TolFun',0.001,'maxiter',max_iter);
135
136 [x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)
137
138 [mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),-fval);
139
140 B_S_OPT = b_s_opt;
```

```
X_OPT = x;
MU_OPT = mu_opt;

end
```

SOL.m

```
function [x0,k_s,ro_s,sigma_s,z] = SOL(max_iter,precision,method,in_point)

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

Substr_con = [1 1];

%%%%%%%%%%%%%%%%%%%%%%%%% STARTS OPTIMIZATION %%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin == 4

disp('fminsearch starts with user-given starting point:')

y = in_point

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2));

options = optimset('Display','iter','PlotFcns',...
@optimplotfval,'TolFun',0.001,'maxiter',max_iter);

[x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)

[mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),-fval);

B_S_OPT = b_s_opt;
X_OPT = x;
MU_OPT = mu_opt;

if MU_OPT == 0
disp('Optimization didnt succeed with given starting point')
else
disp('Optimization succeeded')
end

disp('Solution vector [x1 x2 beta1 beta2 mu] :')

x0 = [X_OPT , B_S_OPT.']

disp('Zeros of balanced growth and optimality equations:')

contG(x0)

fun = @contG;

if method == 'fs'

disp('Point found by fsolve:')

z = fsolve(fun,x0)

disp('Image of the point found:')

contG(z)

else

disp('Point found by lsqnonlin:')

```

```
lb = zeros(size(x0)); %lower bound of zero
ub = Inf*ones(size(x0));
z = lsqnonlin(fun,x0,lb,ub)

end

else

disp('Please wait! searching for a maximum over the x_s with ranges...
 for x1 and x2,[0 , S1/k_1] and [0 , S2/k_2]:')

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]

t_start = tic;

[B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH,H] =...
 Maximizer(Substr_con,x1_range,x2_range,max_iter,precision)

if B_S_OPT(3) == 0
disp('Optimization didnt succeed with given starting point')
else
disp('optimization succeeded')
end

disp('Minutes it took to find a solution:')

time = round(toc(t_start)/60,3)

disp('Solution vector [x1 x2 beta1 beta2 mu] :')

x0 = [X_OPT , B_S_OPT.']

disp('Zeros of steady state and optimality equations:')

contG(x0)

fun = @contG;

if method == 'fs'

disp('Point found by fsolve:')

z = fsolve(fun,x0)

disp('Image of the point found:')

contG(z)

else

disp('Point found by lsqnonlin:')

lb = zeros(size(x0)); %lower bound of zero
ub = Inf*ones(size(x0));
z = lsqnonlin(fun,x0,lb,ub)

end

end

end
```

contG.m

```
function [z] = contG(x);
```

```
%used to get a point which is on the curve to start the continuation

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

S_1 = S1_c;
S_2 = S2_c;

x_1 = x(1);
x_2 = x(2);
B1 = x(3);
B2 = x(4);
MU = x(5);

syms x1 x2 S1 S2 mu beta_1 beta_2

f1 = @(x1,S1) kcat*((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2) kcat*((S2-k2_f*x2)/(1+S2+x2));

g1 = @(x1,x2) k1_g*x1*x2/(1+x1+x2+x1*x2);
g2 = @(x1,x2) k2_g*x1*x2/(1+x1+x2+x1*x2);
gr = @(x1,x2) kr_g*x1*x2/(1+x1+x2+x1*x2);

F1 = @(beta_1,beta_2,mu)   x1*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
 (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 + (sigma_r-br)*mu  ) -...
  (P11*f1(x1,S1)/mu - M11)*beta_1 - (P12*f2(x2,S2)/mu - M12)*beta_2 + M1r*mu;
F2 = @(beta_1,beta_2,mu)   x2*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
 (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 + (sigma_r-br)*mu  ) -...
  (P21*f1(x1,S1)/mu - M21)*beta_1 - (P22*f2(x2,S2)/mu - M22)*beta_2 + M2r*mu;

Fr = @(beta_1,beta_2,mu) beta_1/g1(x1,x2) + beta_2/g2(x1,x2) + mu/gr(x1,x2) - 1;

dF1dx1 = diff(F1,x1);
dF1dx2 = diff(F1,x2);
dF1dbeta_1 = diff(F1,beta_1);
dF1dbeta_2 = diff(F1,beta_2);

dF2dx1 = diff(F2,x1);
dF2dx2 = diff(F2,x2);
dF2dbeta_1 = diff(F2,beta_1);
dF2dbeta_2 = diff(F2,beta_2);

dFrdx1 = diff(Fr,x1);
dFrdx2 = diff(Fr,x2);
dFrdbeta_1 = diff(Fr,beta_1);
dFrdbeta_2 = diff(Fr,beta_2);

dF1dx1 = matlabFunction(dF1dx1); %@(S1,S2,beta_1,beta_2,mu,x1,x2)
dF1dx2 = matlabFunction(dF1dx2); %@(S2,beta_2,mu,x1,x2)
dF1dbeta_1 = matlabFunction(dF1dbeta_1); %@(S1,mu,x1)
dF1dbeta_2 = matlabFunction(dF1dbeta_2); %@(S2,mu,x1,x2)

dF2dx1 = matlabFunction(dF2dx1); %@(S1,beta_1,mu,x1,x2)
dF2dx2 = matlabFunction(dF2dx2); %@(S1,S2,beta_1,beta_2,mu,x1,x2)
dF2dbeta_1 = matlabFunction(dF2dbeta_1); %@(S1,mu,x1,x2)
dF2dbeta_2 = matlabFunction(dF2dbeta_2); %@(S2,mu,x2)

dFrdx1 = matlabFunction(dFrdx1); %@(beta_1,beta_2,mu,x1,x2)
dFrdx2 = matlabFunction(dFrdx2); %@(beta_1,beta_2,mu,x1,x2)
dFrdbeta_1 = matlabFunction(dFrdbeta_1); %@(x1,x2)
dFrdbeta_2 = matlabFunction(dFrdbeta_2); %@(x1,x2)

A1 = zeros(3);
A2 = zeros(3);

A1(1,1) = dF1dx1(S_1,S_2,B1,B2,MU,x_1,x_2);
A1(1,2) = dF1dbeta_1(S_1,MU,x_1);
```

```
A1(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);

A1(2,1) = dF2dx1(S_1,B1,MU,x_1,x_2);
A1(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
A1(2,3) = dF2dbeta_2(S_2,MU,x_2);

A1(3,1) = dFrdx1(B1,B2,MU,x_1,x_2);
A1(3,2) = dFrdbeta_1(x_1,x_2);
A1(3,3) = dFrdbeta_2(x_1,x_2);

A2(1,1) = dF1dx2(S_2,B2,MU,x_1,x_2);
A2(1,2) = dF1dbeta_1(S_1,MU,x_1);
A2(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);

A2(2,1) = dF2dx2(S_1,S_2,B1,B2,MU,x_1,x_2);
A2(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
A2(2,3) = dF2dbeta_2(S_2,MU,x_2);

A2(3,1) = dFrdx2(B1,B2,MU,x_1,x_2);
A2(3,2) = dFrdbeta_1(x_1,x_2);
A2(3,3) = dFrdbeta_2(x_1,x_2);

r1 = min(abs(eigs(A1)));
r2 = min(abs(eigs(A2)));

FF1 = @(x1,x2,beta_1,beta_2,mu,S1,S2)   x1*(  (a1*f1(x1,S1)/mu +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P11*f1(x1,S1)/mu - M11)*beta_1 -...
   (P12*f2(x2,S2)/mu - M12)*beta_2 + M1r*mu;
FF2 = @(x1,x2,beta_1,beta_2,mu,S1,S2)   x2*(  (a1*f1(x1,S1)/mu +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P21*f1(x1,S1)/mu - M21)*beta_1 -...
   (P22*f2(x2,S2)/mu - M22)*beta_2 + M2r*mu;

FFr = @(x1,x2,beta_1,beta_2,mu) beta_1/g1(x1,x2) +...
 beta_2/g2(x1,x2) + mu/gr(x1,x2) - 1;

y1 =  FF1(x_1,x_2,B1,B2,MU,S_1,S_2);

y2 =  FF2(x_1,x_2,B1,B2,MU,S_1,S_2);

y3 =  FFr(x_1,x_2,B1,B2,MU);

y4 = r1;
y5 = r2;

z = [y1;y2;y3;y4;y5];
```

visualH.m

```
function [H,k_s,ro_s,sigma_s,x1_s,x2_s] = visualH(n,H_in,X1_RANGE,X2_RANGE,visu)

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

Substr_con = [1,1];

count = 0

if nargin == 1

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]

a_x1 = x1_range(1);
```

```
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));

H = zeros(n(1),n(2));

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2));

f = waitbar(0,'please wait..');

tot = n(1)*n(2)

t_start = tic;

for i=1:n(1)
for j=1:n(2)
if ro_1*x1_s(i)+ro_2*x2_s(j) > 1
H(i,j)=0;
count=count+1
f = waitbar(count/tot,f);
else
H(i,j) = opt_M([x1_s(i),x2_s(j)]);
count = count + 1
f = waitbar(count/tot,f);
end
end
end

time = round(toc(t_start)/60,3)

close(f);
end

if nargin == 2

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));

figure
surf(x1_s,x2_s,-H_in)
%title('Plot of the H Function in the x1-x2 plane');
xlabel('x1');
ylabel('x2');
height = [0 -min(H_in(:))];
axis([x1_s(1) x1_s(end) x2_s(1) x2_s(end) height])
az = 0;
el = 90;
view(az, el);
caxis([0 -min(H_in(:))])
colorbar

end

if nargin == 4

x1_range = X1_RANGE;
x2_range = X2_RANGE;
```

```
85
86  a_x1 = x1_range(1);
87  b_x1 = x1_range(2);
88  a_x2 = x2_range(1);
89  b_x2 = x2_range(2);
90
91  x1_s = linspace(a_x1,b_x1,n(1));
92  x2_s = linspace(a_x2,b_x2,n(2));
93
94  H = zeros(n(1),n(2));
95
96  opt_M = @(X) fixed_pt(Substr_con,X(1),X(2));
97
98  f = waitbar(0,'please wait..');
99
100 tot = n(1)*n(2)
101
102 t_start = tic;
103
104 for i=1:n(1)
105 for j=1:n(2)
106 %t_start2 = tic;
107 H(i,j) = opt_M([x1_s(i),x2_s(j)]);
108 count = count + 1
109 f = waitbar(count/tot,f);
110 %time_iteration = round(toc(t_start2)/60,3)
111 end
112 end
113
114 time = round(toc(t_start)/60,3)
115
116 close(f);
117 end
118
119 if nargin == 5
120
121 x1_range = X1_RANGE;
122 x2_range = X2_RANGE;
123
124 a_x1 = x1_range(1);
125 b_x1 = x1_range(2);
126 a_x2 = x2_range(1);
127 b_x2 = x2_range(2);
128
129 x1_s = linspace(a_x1,b_x1,n(1));
130 x2_s = linspace(a_x2,b_x2,n(2));
131
132 figure
133 surf(x1_s,x2_s,-H_in)
134 height = [0 -min(H_in(:))];
135 axis([x1_s(1) x1_s(end) x2_s(1) x2_s(end) height])
136 az = 0;
137 el = 90;
138 view(az, el);
139 caxis([0 -min(H_in(:))])
140 colorbar
141
142 end
143
144 end
```

Simple_Example_System.m

```
1  function [dx]=Simple_Example_System(t,x,alpha_s)
2
3  global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
```

```
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

alpha_1 = alpha_s(1);
alpha_2 = alpha_s(2);
alpha_r = alpha_s(3);

S1 = S1_c;
S2 = S2_c;

f1 = kcat*((S1-k1_f*x(1))/(1+S1+x(1)));
f2 = kcat*((S2-k2_f*x(2))/(1+S2+x(2)));

g1 = k1_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
g2 = k2_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
gr = kr_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));

V1 = x(3)*f1;
V2 = x(4)*f2;

W1 = x(5)*alpha_1*g1;
W2 = x(5)*alpha_2*g2;
Wr = x(5)*alpha_r*gr;

MU = ro_1*(P11*V1+P12*V2)+ ...
ro_2*(P21*V1+P22*V2) + ...
(sigma_1-(ro_1*M11+ro_2*M21))*W1 + ...
(sigma_2-(ro_1*M12+ro_2*M22))*W2 + ...
(sigma_r-(ro_1*M1r+ro_2*M2r))*Wr;

dx1 = V1 - M11*W1 - M12*W2 - M1r*Wr - MU*x(1);
dx2 = V2 - M21*W1 - M22*W2 - M2r*Wr - MU*x(2);
de1 = W1 - MU*x(3);
de2 = W2 - MU*x(4);
dr  = Wr - MU*x(5);

%[ro_s sigma_s] * x;

dx=[dx1;dx2;de1;de2;dr];
```

Run_System.m

```
function [q]  = Run_System(alpha_s,xinit);

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

Substr_con = [1 1];

if nargin == 0
alpha1 = rand;
alpha2 = rand;
alpha3 = rand;

alpha_1 = alpha1/(alpha1+alpha2+alpha3);
alpha_2 = alpha2/(alpha1+alpha2+alpha3);
alpha_r = alpha3/(alpha1+alpha2+alpha3);

alpha_s = [alpha_1,alpha_2,alpha_r];

xinit = [0.1;0.1;0.1;0.1;0.1];                    % Initial condition

else
alpha_1 = alpha_s(1);
alpha_2 = alpha_s(2);
```

```
alpha_r = alpha_s(3);
end

S1 = Substr_con(1);
S2 = Substr_con(2);

t=[0 2000];                                                    % Time window

% somma = [ro_s sigma_s] * xinit;
% xinit = xinit/somma;
%x = xinit;          % Initial condition
[t,y]=ode45(@(t,x) Simple_Example_System...
(t,x,alpha_s),t,xinit);     % Integrate in time

s = size(y);
q = y(s(1),:);
x = q;                   % steady state found

subplot(3,2,1);

plot(t,y(:,1))
title('evolution of metabolite x1');
xlabel('Time t');
ylabel('x1 concentration');
legend('x1')

hold on

subplot(3,2,2);

plot(t,y(:,2))
title('evolution of metabolite x2');
xlabel('Time t');
ylabel('x2 concentration');
legend('x2')

hold on

subplot(3,2,3);

plot(t,y(:,3))
title('evolution of enzyme e1');
xlabel('Time t');
ylabel('e1 concentration');
legend('e1')

hold on

subplot(3,2,4);

plot(t,y(:,4))
title('evolution of enzyme e2');
xlabel('Time t');
ylabel('e2 concentration');
legend('e2')

hold on

subplot(3,2,5);

plot(t,y(:,5))
title('evolution of the ribosome r');
xlabel('Time t');
ylabel('ribosome concentration');
legend('r')

hold on
```

Simple_Example_System_morac.m

```
function [dx]=Simple_Example_System_morac(t,x)

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s sigma_s Enz_con

Substr_con = [1 1];

%load the optima, found with the continuation

load('optima.mat');
optima = zopts;

%S1 = S1_c;
%S1 = 1;
%S1 = 2;
S1 = 0.5;
S2 = S2_c;

f1 = kcat*((S1-k1_f*x(1))/(1+S1+x(1)));
f2 = kcat*((S2-k2_f*x(2))/(1+S2+x(2)));

g1 = k1_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
g2 = k2_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
gr = kr_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));

V1 = x(3)*f1;
V2 = x(4)*f2;

sensor=1;
betas = find_optimal_beta(x,sensor,optima);
beta_1 = betas(1);
beta_2 = betas(2);
beta_r = betas(3);

W1 = x(5)*beta_1;
W2 = x(5)*beta_2;
Wr = x(5)*beta_r;

MU = ro_1*(P11*V1+P12*V2)+ ...
ro_2*(P21*V1+P22*V2) + ...
(sigma_1-(ro_1*M11+ro_2*M21))*W1 + ...
(sigma_2-(ro_1*M12+ro_2*M22))*W2 + ...
(sigma_r-(ro_1*M1r+ro_2*M2r))*Wr;


dx1 = V1 - M11*W1 - M12*W2 - M1r*Wr - MU*x(1);
dx2 = V2 - M21*W1 - M22*W2 - M2r*Wr - MU*x(2);
de1 = W1 - MU*x(3);
de2 = W2 - MU*x(4);
dr  = Wr - MU*x(5);
dMu = 100*(MU-x(6));

[t [ro_s sigma_s] * x(1:5)];

dx=[dx1;dx2;de1;de2;dr;dMu];

%%%%%%%%%%
%%%%%%%%%%

function  beta = find_optimal_beta(x,sensor,optima);

xs = x(sensor);
```

```
xopt = optima(sensor,:);
betas = optima(3:5,:);
for i=1:3
betas(i,:);
beta(i) = interp1(xopt,betas(i,:),xs);
end
if xs < min(xopt(sensor,:))
beta = betas(:,1); % just one extreme choice of betas
elseif xs > max(xopt(sensor,:))
beta = betas(:,end); % the other extreme choice of betas
end
beta = beta(:);
```

Run_System_morac.m

```
function [q,t,y]  = Run_System_morac(xinit);

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

Substr_con = [1 1];

S1 = Substr_con(1);
S2 = Substr_con(2);

%t=[0 2000];% Time window
%t=[2000 4000];
t=[4000 6000];

somma = dot([ro_s sigma_s],xinit);
xinit = xinit/somma;
ini = [xinit 1];          % Initial condition
[t,y]=ode45(@(t,x) Simple_Example_System_morac(t,x),t,ini);     % Integrate in time

s = size(y);
q = y(s(1),:);
x = q;                    % steady state found

subplot(3,2,1);

plot(t,y(:,1))
title('evolution of metabolite x1');
xlabel('Time t');
ylabel('x1 concentration');
legend('x1')

hold on

subplot(3,2,2);

plot(t,y(:,2))
title('evolution of metabolite x2');
xlabel('Time t');
ylabel('x2 concentration');
legend('x2')

hold on

subplot(3,2,3);

plot(t,y(:,3))
title('evolution of enzyme e1');
xlabel('Time t');
ylabel('e1 concentration');
legend('e1')
```

```
52
53 hold on
54
55 subplot(3,2,4);
56
57 plot(t,y(:,4))
58 title('evolution of enzyme e2');
59 xlabel('Time t');
60 ylabel('e2 concentration');
61 legend('e2')
62
63 hold on
64
65 subplot(3,2,5);
66
67 plot(t,y(:,5))
68 title('evolution of the ribosome r');
69 xlabel('Time t');
70 ylabel('ribosome concentration');
71 legend('r')
72
73 hold on
```

run_set.m

```
1  function [x1,x2,mu] = run_set(N);
2
3  global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
4  kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
5   S1_c S2_c k_s ro_s sigma_s Enz_con
6
7  alpha1 = linspace(0,1,N);
8  alpha2 = linspace(0,1,N);
9
10 IC = [0.3;0.3;0.3;0.3;0.3;0.3];
11 som = [ro_s sigma_s] * IC(1:5);
12 IC = IC/som;
13 [ro_s sigma_s] * IC(1:5);
14
15 options = odeset('AbsTol',1e-8,'RelTol',1e-8);
16
17 x1 = zeros(N);
18 x2 = zeros(N);
19 mu = zeros(N);
20
21 for i=1:N
22 i
23 for j =1:N
24 j
25 al1 = alpha1(i);
26 al2 = alpha2(j);
27 alr = 1-al1 - al2;
28 if alr > 0
29 rhs = @(t,y) Simple_Example_System_dummy(t,y,[al1 al2 alr]);
30 [t,y] = ode45(rhs,[0,1000],IC,options);
31
32 x1(i,j) = y(end,1);
33 x2(i,j) = y(end,2);
34
35 e1(i,j) = y(end,3);
36 e2(i,j) = y(end,4);
37 r(i,j) = y(end,5);
38 mu(i,j) = y(end,6);
39
40 end
41 end
```

```
end

v = find(x1==0);
x1(v) = nan;
x2(v) = nan;
mu(v) = nan;

figure
surf(x1,x2,mu);
title('Plot of the H Function Obtained by Running the System');
xlabel('alpha1');
ylabel('alpha2');
view(0,90);
```

ContG_S1.m

```
function [r] = contG_S1(x);
%right-hand side for the continuation

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

%S2 = 1;
S_2 = 1;

x_1 = x(1);
x_2 = x(2);
B1 = x(3);
B2 = x(4);
MU = x(5);
S_1 = x(6);

syms x1 x2 S1 S2 mu beta_1 beta_2

f1 = @(x1,S1) kcat*((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2) kcat*((S2-k2_f*x2)/(1+S2+x2));

g1 = @(x1,x2) k1_g*x1*x2/(1+x1+x2+x1*x2);
g2 = @(x1,x2) k2_g*x1*x2/(1+x1+x2+x1*x2);
gr = @(x1,x2) kr_g*x1*x2/(1+x1+x2+x1*x2);

F1 = @(beta_1,beta_2,mu)   x1*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
  (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 + (sigma_r-br)*mu  ) -...
   (P11*f1(x1,S1)/mu - M11)*beta_1 - (P12*f2(x2,S2)/mu - M12)*beta_2 + M1r*mu;
F2 = @(beta_1,beta_2,mu)   x2*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
  (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 + (sigma_r-br)*mu  ) -...
   (P21*f1(x1,S1)/mu - M21)*beta_1 - (P22*f2(x2,S2)/mu - M22)*beta_2 + M2r*mu;

Fr = @(beta_1,beta_2,mu) beta_1/g1(x1,x2) + beta_2/g2(x1,x2) + mu/gr(x1,x2) - 1;

dF1dx1 = diff(F1,x1);
dF1dx2 = diff(F1,x2);
dF1dbeta_1 = diff(F1,beta_1);
dF1dbeta_2 = diff(F1,beta_2);

dF2dx1 = diff(F2,x1);
dF2dx2 = diff(F2,x2);
dF2dbeta_1 = diff(F2,beta_1);
dF2dbeta_2 = diff(F2,beta_2);

dFrdx1 = diff(Fr,x1);
dFrdx2 = diff(Fr,x2);
dFrdbeta_1 = diff(Fr,beta_1);
dFrdbeta_2 = diff(Fr,beta_2);

```

```
dF1dx1 = matlabFunction(dF1dx1); %@(S1,S2,beta_1,beta_2,mu,x1,x2)
dF1dx2 = matlabFunction(dF1dx2); %@(S2,beta_2,mu,x1,x2)
dF1dbeta_1 = matlabFunction(dF1dbeta_1); %@(S1,mu,x1)
dF1dbeta_2 = matlabFunction(dF1dbeta_2); %@(S2,mu,x1,x2)

dF2dx1 = matlabFunction(dF2dx1); %@(S1,beta_1,mu,x1,x2)
dF2dx2 = matlabFunction(dF2dx2); %@(S1,S2,beta_1,beta_2,mu,x1,x2)
dF2dbeta_1 = matlabFunction(dF2dbeta_1); %@(S1,mu,x1,x2)
dF2dbeta_2 = matlabFunction(dF2dbeta_2); %@(S2,mu,x2)

dFrdx1 = matlabFunction(dFrdx1); %@(beta_1,beta_2,mu,x1,x2)
dFrdx2 = matlabFunction(dFrdx2); %@(beta_1,beta_2,mu,x1,x2)
dFrdbeta_1 = matlabFunction(dFrdbeta_1); %@(x1,x2)
dFrdbeta_2 = matlabFunction(dFrdbeta_2); %@(x1,x2)

A1 = zeros(3);
A2 = zeros(3);

A1(1,1) = dF1dx1(S_1,S_2,B1,B2,MU,x_1,x_2);
A1(1,2) = dF1dbeta_1(S_1,MU,x_1);
A1(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);

A1(2,1) = dF2dx1(S_1,B1,MU,x_1,x_2);
A1(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
A1(2,3) = dF2dbeta_2(S_2,MU,x_2);

A1(3,1) = dFrdx1(B1,B2,MU,x_1,x_2);
A1(3,2) = dFrdbeta_1(x_1,x_2);
A1(3,3) = dFrdbeta_2(x_1,x_2);

A2(1,1) = dF1dx2(S_2,B2,MU,x_1,x_2);
A2(1,2) = dF1dbeta_1(S_1,MU,x_1);
A2(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);

A2(2,1) = dF2dx2(S_1,S_2,B1,B2,MU,x_1,x_2);
A2(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
A2(2,3) = dF2dbeta_2(S_2,MU,x_2);

A2(3,1) = dFrdx2(B1,B2,MU,x_1,x_2);
A2(3,2) = dFrdbeta_1(x_1,x_2);
A2(3,3) = dFrdbeta_2(x_1,x_2);

r1 = min(abs(eigs(A1)));
r2 = min(abs(eigs(A2)));

FF1 = @(x1,x2,beta_1,beta_2,mu,S1,S2)   x1*(  (a1*f1(x1,S1)/mu +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P11*f1(x1,S1)/mu - M11)*beta_1 -...
   (P12*f2(x2,S2)/mu - M12)*beta_2 + M1r*mu;
FF2 = @(x1,x2,beta_1,beta_2,mu,S1,S2)   x2*(  (a1*f1(x1,S1)/mu +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu + sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P21*f1(x1,S1)/mu - M21)*beta_1 -...
   (P22*f2(x2,S2)/mu - M22)*beta_2 + M2r*mu;

FFr = @(x1,x2,beta_1,beta_2,mu) beta_1/g1(x1,x2) +...
 beta_2/g2(x1,x2) + mu/gr(x1,x2) - 1;

y1 =  FF1(x_1,x_2,B1,B2,MU,S_1,S_2);

y2 =  FF2(x_1,x_2,B1,B2,MU,S_1,S_2);

y3 =  FFr(x_1,x_2,B1,B2,MU);

y4 = r1;
y5 = r2;

r = [y1;y2;y3;y4;y5];
```

input_output.m

```
function zopts = input_output(z);

z = z(:);
fun = @contG_S1
fun(z)
Fun = @(t,y) fun(y)
dfun = @(z) numjac(Fun, 0, z, Fun(0,z), 1e-9)
dfun(z)
zopt1 = cont(fun, dfun, z, 20, 0.001, eps, 1, 1e-7) % one way
zopt2 = cont(fun, dfun, z, 20, -0.001, eps, 1, 1e-7) % the other way

y = [zopt1 zopt2]
[ys,v] = sort(y(1,:)) %sort the vector, we need the indices not ys
zopts = y(:,v);
```

ran.m

```
function [r]=ran(a);
%genrates random number between -a and a

r = -a + 2*a*rand;
```

cont.m

```
function curve=cont(f,df,x0,nsteps,step0,stepmin,stepmax,tol,silent)
%CONT finds the solution curve to a system of equations
%
%  X=CONT(F,DF,X0,NSTEPS,STEP0,STEPMIN,STEPMAX,TOL,SILENT)
%  X=CONT(F,DF,X0,NSTEPS,STEP0)
%  computes the solution curve X
%  for N-1 equations F=0 in N variables,
%  starting from an initial solution X0,
%  using pseudo-arclength continuation.
%
%  Here X0 should be a column vector of length N and
%  the function F should send a vector of length N
%  to a column vector of length N-1,
%  i.e. F : R^N -> R^(N-1).
%  DF should be the (N-1) x N derivative matrix of F.
%  All vectors are assumed to be column vectors.
%
%  The output X is a matrix with per column a solution of F=0.
%
%  F should be a function_handle. It can be either a handle to
%  an anonymous or to a function file (but not an inline function).
%  If DF=[] then the derivative will be computed numerically.
%
%  In total a maximum of NSTEPS steps are taken,
%  starting with stepsize STEP0 and terminating
%  if the stepsize becomes smaller than STEPMIN.
%  The maximum stepsize is STEPMAX.
%  The initial guess X0 should be no further than STEPMAX from a real zero.
%  The sign of STEP0 determines the direction of the first step
%
%  Default values for STEPMIN and STEPMAX are 10^-8 and 1.
%  Stepsize are measured in terms of L^2 norms, so for large problems
%  the user may want to scale the max/min step size appropriately.
%
%  TOL determines the tolerance in determining zeros (default 10^-14).
%  By default a plot is produced of the first two
%  components of the solution curve,
```

```
%  except if the dimension equals 3, when a curve in 3D is plotted.
%
%  If SILENT=1 then no plot is produced and no warning messages are displayed.
%

% default values
if ~exist('stepmax') || isempty(stepmax)
stepmax=1;
end
if ~exist('stepmin') || isempty(stepmin)
stepmin=1e-8;
end
if ~exist('tol') || isempty(tol)
tol=1e-14;
end
if ~exist('silent')
silent=0;
end

if isempty(df)
% numerical derivative
df=@(x) numjac(@(t,y) f(y),0,x,f(x),eps);
end

% initialization
dim=length(x0);
y0=reshape(x0,dim,1);
curve=[];
niter=100;
step=sign(step0)*max(min(abs(step0),stepmax),stepmin);
k=0
%relative amount predictor is allowed to be off
correctiemax=1/10;

% compute first point
gr=df(y0);
nullgr=null(gr)';

[ynew,dummy,iters]=multinewton(@(x) [f(x);nullgr*(x-y0)],...
@(x) [df(x);nullgr],y0,tol,niter,1);

% check first point
if iters > niter || norm(ynew-y0)>stepmax
if silent~=1
fprintf(['could not find a zero near the given starting point.\n'])
end
return
end

% start loop
while k<=nsteps & abs(step)>=stepmin
%update
y=ynew;
k=k+1
curve=[curve,y];
gr=df(y);
nullgrold=nullgr;
nullgr=null(gr)';

% check that the gradient did not flip
if nullgrold*nullgr'<0
nullgr=-nullgr;
end
result=0;

% iterate until successful
while abs(step)>=stepmin & result~=1
% predictor
```

```
z=y+step*nullgr';
% corrector
[ynew,dummy,iters]=multinewton(@(x) [f(x);nullgr*(x-z)],...
@(x) [df(x);nullgr],z,tol,niter,1);
% check newtonoutput is OK
if norm(z-ynew)<correctiemax*abs(step) & iters<niter
result=1;
% iteration succesfull: adapt stepsize
% increase stepsize if iters<5
% decrease stepsize if iters>5
step=sign(step)*max(stepmin,min(stepmax,abs(step)*2^((4-iters)/3)));
else
% corrector too large (or nonconvergence):
% decrease stepsize and try again
step=step*2^(-1/3);
end
end
end

% check untimely end
if k<nsteps & silent~=1
fprintf(['Continuation terminated after %i steps ', ...
'due to lack of convergence.\n'],k-1);
end

% plot
if silent~=1 & k>1
if dim==3
plot3(curve(1,:)',curve(2,:)',curve(3,:)','.-')
else
plot(curve(1,:)',curve(2,:)','.-')
end
end

return
```

multinewton.m

```
function [zero,res,niter]=multinewton(f,df,x0,tol,nmax,silent,varargin)
% MULTINEWTON Find function zeros using Newtons method in more dimensions.
%
%   ZERO=MULTINEWTON(FUN,DFUN,X0,TOL,NMAX,SILENT)
%   tries to find the zero ZERO of the continuous and
%   differentiable function FUN nearest to X0 using Newtons method.
%   FUN is a function from R^n to R^n (a vector valued function)
%   Its derivative is DFUN, which a function from
%   R^n to R^(n^2), i.e., a matrix valued function.
%   Both accept a column vector input (and FUN also returns a column vector).
%   FUN and DFUN can also be inline objects or anonymous functions.
%
%   If the search fails an error message is displayed.
%   If SILENT=1 then no error messages are displayed (SILENT=0 by default).
%
%   The tolerance TOL and the maximum number NMAX of iterations
%   have default values 1e-14 and 100.
%   Convergence test is: norm(x(n)-x(n-1)) < TOL.
%
%   [ZERO,RES,NITER]= MULTINEWTON(FUN,...)
%   returns the value of the residual RES (as a vector)
%   in ZERO and the number of itererations NITER.

% initialization
if ~exist('nmax')
nmax=100;
end
if ~exist('tol')
```

```
tol=1e-14;
end
if ~exist('silent')
silent=0;
end

x = x0;
niter = 0;
diff = tol+1;
while norm(diff) >= tol & niter <= nmax
fx = feval(f,x,varargin{:});
dfx = feval(df,x,varargin{:});
niter = niter + 1;
diff = - dfx\fx;
x = x + diff;
end
if niter > nmax & silent==0
fprintf(['newton stopped without converging to the desired tolerance\n',...
'because the maximum number of iterations was reached\n']);
end
zero = x;
res = feval(f,x,varargin{:});

return
```

cal_fixed_alphas.m

```
function [x1,x2,mu,al1,al2,e1,e2,r] = cal_fixed_alphas(alpha1,alpha2);

global a1 a2 b1 b2 br M11 M12 M1r M21 M22 M2r P11 P12 P21 P22 ro_1 ro_2 ...
kcat k1_f k2_f k1_g k2_g kr_g sigma_1 sigma_2 sigma_r S1_c S2_c k_s ro_s sigma_s Enz_con

IC = [0.3;0.3;0.3;0.3;0.3;0.3];
som = [ro_s sigma_s] * IC(1:5);
IC = IC/som;
[ro_s sigma_s] * IC(1:5);

options = odeset('AbsTol',1e-9,'RelTol',1e-9);

alpha_r = 1 - alpha1 - alpha2;

if alpha_r < 0
x1=0;
x2=0;
mu=0;
e1=0;
e2=0;
r=0;
else
rhs = @(t,y) Simple_Example_System_dummy(t,y,[alpha1 alpha2 alpha_r]);
[~,y] = ode15s(rhs,[0,1000],IC,options);

x1 = y(end,1);
x2 = y(end,2);
e1 = y(end,3);
e2 = y(end,4);
r  = y(end,5);
mu = -y(end,6);
end

al1 = alpha1;
al2 = alpha2;

end
```

DummyFunction1.m

```
function out3 = DummyFunction1(alpha1,alpha2)
[~,~,out3,~,~,~,~,~] = cal_fixed_alphas(alpha1,alpha2);
end
```

run_script.m

```

global k1_g k2_g

DummyFunction2 = @(X) DummyFunction1(X(1),X(2));

options = optimset('Display','iter','PlotFcns',@optimplotfval,'TolFun',0.000001,'maxiter',200);

y = [0.3,0.3];

%[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

tic
N = 200;

sub1 = linspace(0.1,5,N);

zopts = zeros(11,N);

for i = 1:N

k1_f = sub1(i);

[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

[x1,x2,mu,al1,al2,e1,e2,r,] = cal_fixed_alphas(x(1),x(2))

zopts(1,i)= x1;
zopts(2,i)= x2;
zopts(3,i)= e1;
zopts(4,i)= e2;
zopts(5,i)= r;

zopts(6,i)= al1;
zopts(7,i)= al2;
zopts(8,i)= al1*k1_g*x1*x2/(1+x1+x2+x1*x2);
zopts(9,i)= al2*k2_g*x1*x2/(1+x1+x2+x1*x2);
zopts(10,i)= -mu;

zopts(11,i)= sub1(i);

end

time = toc
```

A.0.2 Enzymatic Toxin Transport Codes

Constants_set.m

```
global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s k_s sigma_s Enz_con

rndmz = 'N';
```

```
7
8  e = 0.05;
9
10 Substr_con = [1 1];
11
12 Network = [1,0,0;
13 0,1,0;
14 0,1,-1];
15
16 Enz_con = [3,4,2,2;
17 2,4,3,2;
18 0,0,0,0];
19
20 if rndmz == 'Y'
21
22 %disp('Vector of randomization:')
23
24 rad = [ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ...
25 ran(e) , ran(e) , ran(e) , ran(e), ran(e) , ran(e) , ran(e) , ran(e)]; %15 values
26
27 %disp('Original constants:')
28
29 k_s = [0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]; % 9 values
30
31 ro_s = [1,1,1];
32
33 sigma_s = 10*[1,1,1,1];
34
35 %disp('Perturbed values:')
36
37 k_s = [0.25,0.5+rad(1)/2,0.5+rad(2)/2,0.5+rad(3)/2,0.5+rad(4)/2,...
38 0.5+rad(5)/2,0.5+rad(6),0.5+rad(7),0.5+rad(8)];
39
40 ro_s = [1+rad(9),1+rad(10),1+rad(11)];
41
42 sigma_s = 10*[1+rad(12),1+rad(13),1+rad(14),1+rad(15)];
43
44 else
45 if rndmz == 'N'
46
47 %disp('Using fixed constants')
48
49 k_s = [0.2500    0.4881    0.5023    0.4863    0.4998...
50     0.4977    0.5234    0.5089    0.5090];
51
52 ro_s = [0.9711    1.0404    1.0225];
53
54 sigma_s = [0.9992    1.0311    0.9748    1.0167];
55
56 else
57 if rndmz == 'S'
58
59 disp('Using original (symmetric) constants:')
60
61 k_s = [0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]; % 9 values
62
63 ro_s = [1,1,1];
64
65 sigma_s = 10*[1,1,1,1];
66 end
67 end
68 end
69
70 %%%%%%%%%%%%%%%%%% CONSTANTS %%%%%%%%%%%%%%%%%%
71
72 M11 = Enz_con(1,1);
73 M12 = Enz_con(1,2);
74 M13 = Enz_con(1,3);
```

```
M1r = Enz_con(1,4);
M21 = Enz_con(2,1);
M22 = Enz_con(2,2);
M23 = Enz_con(2,3);
M2r = Enz_con(2,4);
M31 = Enz_con(3,1);
M32 = Enz_con(3,2);
M33 = Enz_con(3,3);
M3r = Enz_con(3,4);

P11 = Network(1,1);
P12 = Network(1,2);
P13 = Network(1,3);
P21 = Network(2,1);
P22 = Network(2,2);
P23 = Network(2,3);
P31 = Network(3,1);
P32 = Network(3,2);
P33 = Network(3,3);

kcat = k_s(1);
k1_f = k_s(2);
k2_f = k_s(3);
k3_fd= k_s(4);
k1_g = k_s(5);
k2_g = k_s(6);
k3_g = k_s(7);
kr_g = k_s(8);
k_T  = k_s(9);

ro_1 = ro_s(1);
ro_2 = ro_s(2);
ro_3 = ro_s(3);

sigma_1 = sigma_s(1);
sigma_2 = sigma_s(2);
sigma_3 = sigma_s(3);
sigma_r = sigma_s(4);

S1_c = Substr_con(1);
S2_c = Substr_con(2);

a1 = ro_1*P11 + ro_2*P21 + ro_3*P31;
a2 = ro_1*P12 + ro_2*P22 + ro_3*P32;
a3 = ro_1*P13 + ro_2*P23 + ro_3*P33;

b1=ro_1*M11+ro_2*M21;
b2=ro_1*M12+ro_2*M22;
b3=ro_1*M13+ro_2*M23;
br=ro_1*M1r+ro_2*M2r;
```

optH.m

```
function [mu_opt,exitflag,b_s_opt,Aeq,exit]=optH(Substr_con,x1,x2,T,mu_tilde);

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T sigma_1 sigma_2 sigma_3 sigma_r

S1 = Substr_con(1);
S2 = Substr_con(2);

f1 = @(x1,S1,T) kcat*(1/(k_T+T))*((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2,T) kcat*((S2-k2_f*x2*T)/(1+S2+x2+T+T*x2));
f3 = @(T) k3_fd*(T/(1+T));

```

```
g1 = @(x1,x2) k1_g*x1*x2/(1+x1+x2+x1*x2);
g2 = @(x1,x2) k2_g*x1*x2/(1+x1+x2+x1*x2);
g3 = @(x1,x2) k3_g*x1*x2/(1+x1+x2+x1*x2);
gr = @(x1,x2) kr_g*x1*x2/(1+x1+x2+x1*x2);

%F1_mod = @(beta_1,beta_2,mu)   x1*(  (a1*f1(x1,S1)/mu_tilde +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu_tilde+sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P11*f1(x1,S1)/mu_tilde-M11)*beta_1 -...
   (P12*f2(x2,S2)/mu_tilde-M12)*beta_2 + M1r*mu;
%F2_mod = @(beta_1,beta_2,mu)   x2*(  (a1*f1(x1,S1)/mu_tilde +...
 sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu_tilde+sigma_2-b2)*beta_2 +...
  (sigma_r-br)*mu  ) - (P21*f1(x1,S1)/mu_tilde-M21)*beta_1 -...
   (P22*f2(x2,S2)/mu_tilde-M22)*beta_2 + M2r*mu;

%Fr = @(beta_1,beta_2,mu) beta_1/g1(x1,x2) + beta_2/g2(x1,x2) + mu/gr(x1,x2);

f = [0,0,0,-1];
B = -eye(4);
b = [0;0;0;0];
beq = [0;0;0;1];

Aeq = zeros(4);
%mu_tilde;

Aeq(1,1) = x1*(a1*f1(x1,S1,T)/mu_tilde + sigma_1-b1) - (P11*f1(x1,S1,T)/mu_tilde-M11);
Aeq(1,2) = x1*(a2*f2(x2,S2,T)/mu_tilde+sigma_2-b2) - (P12*f2(x2,S2,T)/mu_tilde-M12);
Aeq(1,3) = x1*(a3*f3(T)/mu_tilde+sigma_3-b3) - (P13*f3(T)/mu_tilde-M13);
Aeq(1,4) = x1*(sigma_r-br)+ M1r;

Aeq(2,1) = x2*(a1*f1(x1,S1,T)/mu_tilde + sigma_1-b1) - (P21*f1(x1,S1,T)/mu_tilde-M21);
Aeq(2,2) = x2*(a2*f2(x2,S2,T)/mu_tilde+sigma_2-b2) - (P22*f2(x2,S2,T)/mu_tilde-M22);
Aeq(2,3) = x2*(a3*f3(T)/mu_tilde+sigma_3-b3) - (P23*f3(T)/mu_tilde-M23);
Aeq(2,4) = x2*(sigma_r-br) + M2r;

Aeq(3,1) = T*(a1*f1(x1,S1,T)/mu_tilde + sigma_1-b1) - (P31*f1(x1,S1,T)/mu_tilde-M31);
Aeq(3,2) = T*(a2*f2(x2,S2,T)/mu_tilde+sigma_2-b2) - (P32*f2(x2,S2,T)/mu_tilde-M32);
Aeq(3,3) = T*(a3*f3(T)/mu_tilde+sigma_3-b3) - (P33*f3(T)/mu_tilde-M33);
Aeq(3,4) = T*(sigma_r-br) + M3r;

Aeq(4,1)=1/g1(x1,x2);
Aeq(4,2)=1/g2(x1,x2);
Aeq(4,3)=1/g3(x1,x2);
Aeq(4,4)=1/gr(x1,x2);

options = optimset('Display','none','maxiter',200);

LB = [];
UB = [];

[b_s_opt,fval,exitflag,output] = linprog(f,B,b,Aeq,beq,LB,UB,[],options);

if exitflag == 1 || exitflag == 0
mu_opt=b_s_opt(4);
exit = 1;
else
mu_opt=0;
b_s_opt = [0;0;0;0];
exit = 0;
end

[mu_opt mu_tilde];
```

fixed_pt.m

```
 function [s]=fixed_pt(Substr_con,x1,x2,T);

```

```
fixedpt = @(mu_tilde) optH(Substr_con,x1,x2,T,mu_tilde) - mu_tilde;
options = optimset('TolX',1e-7,'TolFun',1e-7,'maxiter',200);
s = -fzero(fixedpt,0.0000001,options);
```

Maximizer.m

```
function [B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH] = ...
Maximizer(Substr_con,x1_range,x2_range,T_range,max_iter,precision,n)

n=5;

count = 0;

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);
a_T  = T_range(1);
b_T  = T_range(2);

x1_s = linspace(a_x1,b_x1,n);
x2_s = linspace(a_x2,b_x2,n);
T_s  = linspace(a_T,b_T,n);

H = zeros(n,n,n);

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3));

f = waitbar(0,'please wait..');

for i=1:n
for j=1:n
for k=1:n
x1 = x1_s(i);
x2 = x2_s(j);
T = T_s(k);
y = [x1,x2,T];
H(i,j,k) = opt_M(y);
count = count + 1;
f = waitbar(count/125,f);
end
end
end

close(f)

count = 0;

[M,I] = min(H(:));
[I_row, I_col, I_page] = ind2sub(size(H),I);

disp('Point found from the grid:')

y = [x1_s(I_row),x2_s(I_col),T_s(I_page)]

disp('Coordinates of the point:')

coor = [I_row, I_col, I_page]

disp('With minimum:')

optm = opt_M(y)

disp('And precision:')

pres = max([abs(a_x1-b_x1),abs(a_x2-b_x2),abs(a_T-b_T)])/4
```

```
61
62 while  pres > precision
63
64 if I_row == 1
65 a_x1 = x1_s(I_row);
66 b_x1 = x1_s(I_row+1);
67 else
68 if I_row == n
69 a_x1 = x1_s(I_row-1);
70 b_x1 = x1_s(I_row);
71 else
72 a_x1 = x1_s(I_row-1);
73 b_x1 = x1_s(I_row+1);
74 end
75 end
76
77
78 if I_col == 1
79 a_x2 = x2_s(I_col);
80 b_x2 = x2_s(I_col+1);
81 else
82 if I_col == n
83 a_x2 = x2_s(I_col-1);
84 b_x2 = x2_s(I_col);
85 else
86 a_x2 = x2_s(I_col-1);
87 b_x2 = x2_s(I_col+1);
88 end
89 end
90
91
92 if I_page == 1
93 a_T = T_s(I_page);
94 b_T = T_s(I_page+1);
95 else
96 if I_page == n
97 a_T = T_s(I_page-1);
98 b_T = T_s(I_page);
99 else
100 a_T = T_s(I_page-1);
101 b_T = T_s(I_page+1);
102 end
103 end
104
105 x1_s = linspace(a_x1,b_x1,n);
106 x2_s = linspace(a_x2,b_x2,n);
107 T_s  = linspace(a_T,b_T,n);
108
109 H = zeros(n,n);
110
111 f = waitbar(0,'please wait..');
112
113 for i=1:n
114 for j=1:n
115 for k=1:n
116 x1 = x1_s(i);
117 x2 = x2_s(j);
118 T = T_s(k);
119 y = [x1,x2,T];
120 H(i,j,k) = opt_M(y);
121 count = count + 1;
122 f = waitbar(count/125,f);
123 end
124 end
125 end
126
127 close(f)
128
```

```
[M,I] = min(H(:));
[I_row, I_col, I_page] = ind2sub(size(H),I);
count = 0;

disp('Point found from the grid')

y = [x1_s(I_row),x2_s(I_col),T_s(I_page)]

disp('With minimum:')

optm = opt_M(y)

disp('And precision:')

pres = max([abs(a_x1-b_x1),abs(a_x2-b_x2),abs(a_T-b_T)])/4

end

disp('fminsearch starts')

options = optimset('Display','iter','PlotFcns',...
@optimplotfval,'TolFun',0.001,'maxiter',max_iter);

[x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)

[mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),x(3),-fval);

B_S_OPT = b_s_opt;
X_OPT = x;
MU_OPT = mu_opt;

end
```

SOL.m

```
function [x0,k_s,ro_s,sigma_s,z] = SOL(max_iter,precision,method,in_point)

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s k_s sigma_s Enz_con

Substr_con = [1 1];

Constants_Set

%%%%%%%%%%%%%%%%%%%%%%%%%%% STARTS OPTIMIZATION %%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin == 7

disp('fminsearch starts with user-given starting point:')

y = in_point

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3));

options = optimset('Display','iter','PlotFcns',...
@optimplotfval,'TolFun',0.001,'maxiter',max_iter);

[x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)

[mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),x(3),-fval);

B_S_OPT = b_s_opt;
X_OPT = x;
MU_OPT = mu_opt;
```

```
32
33 if MU_OPT == 0
34 disp('Optimization didnt succeed with given starting point')
35 else
36 disp('Optimization succeeded')
37 end
38
39 disp('Solution vector [x1 x2 T beta1 beta2 beta3 mu] :')
40
41 x0 = [X_OPT , B_S_OPT.']
42
43 disp('Zeros of balanced growth and optimality equations:')
44
45 contG(x0)
46
47 fun = @contG;
48
49 if method == 'fs'
50
51 disp('Point found by fsolve:')
52
53 z = fsolve(fun,x0)
54
55 disp('Image of the point found:')
56
57 contG(z)
58
59 else
60
61 disp('Point found by lsqnonlin:')
62
63 lb = zeros(size(x0)); %lower bound of zero
64 ub = Inf*ones(size(x0));
65 z = lsqnonlin(fun,x0,lb,ub)
66
67 end
68
69 else
70
71 disp('Please wait! searching for a maximum over the x_s with ranges...
72  for x1 and x2 and T, [0 , S1/k_1] , [0 , S2/k_2] and [0 , 5]:')
73
74 x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
75 x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
76 T_range = [0.0001,1.5]
77
78 t_start = tic;
79
80 [B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH] =...
81  Maximizer(Substr_con,x1_range,x2_range,T_range,max_iter,precision)
82
83 if B_S_OPT(3) == 0
84 disp('Optimization didnt succeed with given starting point')
85 else
86 disp('optimization succeeded')
87 end
88
89 disp('Minutes it took to find a solution:')
90
91 time = round(toc(t_start)/60,3)
92
93 disp('Solution vector [x1 x2 T beta1 beta2 beta3 mu] :')
94
95 x0 = [X_OPT , B_S_OPT.']
96
97 disp('Zeros of steady state and optimality equations:')
98
99 contG(x0)
```

```
100
101
102 fun = @contG;
103
104 if method == 'fs'
105
106 disp('Point found by fsolve:')
107
108 z = fsolve(fun,x0)
109
110 disp('Image of the point found:')
111
112 contG(z)
113
114 else
115
116 disp('Point found by lsqnonlin:')
117
118 lb = zeros(size(x0)); %lower bound of zero
119 ub = Inf*ones(size(x0));
120 z = lsqnonlin(fun,x0,lb,ub)
121
122 end
123
124 end
125
126 end
```

contG.m

```
1  function [z] = contG(x);
2  %used to get a point which is on the curve to start the continuation
3
4  global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
5   P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
6  kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c
7
8
9  S_1 = S1_c;
10 S_2 = S2_c;
11
12 x_1 = x(1);
13 x_2 = x(2);
14 T_in = x(3);
15 B1 = x(4);
16 B2 = x(5);
17 B3 = x(6);
18 MU = x(7);
19
20 syms x1 x2 T S1 S2 mu beta_1 beta_2 beta_3
21
22 f1 = @(x1,S1,T) kcat*(1/(k_T+T))*((S1-k1_f*x1)/(1+S1+x1));
23 %f1 = @(x1,S1,T) kcat*((S1-k1_f*x1)/(1+S1+x1));
24
25 f2 = @(x2,S2,T) kcat*((S2-k2_f*x2*T)/(1+S2+x2+T+T*x2));
26 f3 = @(T) k3_fd*(T/(1+T));
27
28 g1 = @(x1,x2) k1_g*x1*x2/(1+x1+x2+x1*x2);
29 g2 = @(x1,x2) k2_g*x1*x2/(1+x1+x2+x1*x2);
30 g3 = @(x1,x2) k3_g*x1*x2/(1+x1+x2+x1*x2);
31 gr = @(x1,x2) kr_g*x1*x2/(1+x1+x2+x1*x2);
32
33 F1 = @(beta_1,beta_2,beta_3,mu)   x1*(  (a1*f1(x1,S1,T)/mu + sigma_1-b1)*beta_1 +...
34 (a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 + (a3*f3(T)/mu+sigma_3-b3)*beta_3 +...
35 (sigma_r-br)*mu  ) - (P11*f1(x1,S1,T)/mu-M11)*beta_1 -...
36  (P12*f2(x2,S2,T)/mu-M12)*beta_2 -...
```

```
(P13*f3(T)/mu-M13)*beta_3 + M1r*mu;
F2 = @(beta_1,beta_2,beta_3,mu)   x2*(  (a1*f1(x1,S1,T)/mu + sigma_1-b1)*beta_1 +...
(a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 + (a3*f3(T)/mu+sigma_3-b3)*beta_3 +...
(sigma_r-br)*mu  ) - (P21*f1(x1,S1,T)/mu-M21)*beta_1 -...
 (P22*f2(x2,S2,T)/mu-M22)*beta_2 -...
(P23*f3(T)/mu-M23)*beta_3 + M2r*mu;
F3 = @(beta_1,beta_2,beta_3,mu)    T*(  (a1*f1(x1,S1,T)/mu + sigma_1-b1)*beta_1 +...
(a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 + (a3*f3(T)/mu+sigma_3-b3)*beta_3 +...
(sigma_r-br)*mu  ) - (P31*f1(x1,S1,T)/mu-M31)*beta_1 -...
 (P32*f2(x2,S2,T)/mu-M32)*beta_2 -...
(P33*f3(T)/mu-M33)*beta_3 + M3r*mu;

Fr = @(beta_1,beta_2,beta_3,mu) beta_1/g1(x1,x2) +...
 beta_2/g2(x1,x2) + beta_3/g3(x1,x2) + mu/gr(x1,x2) - 1;

dF1dx1 = diff(F1,x1);
dF1dx2 = diff(F1,x2);
dF1dT  = diff(F1,T);
dF1dbeta_1 = diff(F1,beta_1);
dF1dbeta_2 = diff(F1,beta_2);
dF1dbeta_3 = diff(F1,beta_3);

dF2dx1 = diff(F2,x1);
dF2dx2 = diff(F2,x2);
dF2dT  = diff(F2,T);
dF2dbeta_1 = diff(F2,beta_1);
dF2dbeta_2 = diff(F2,beta_2);
dF2dbeta_3 = diff(F2,beta_3);

dF3dx1 = diff(F3,x1);
dF3dx2 = diff(F3,x2);
dF3dT  = diff(F3,T);
dF3dbeta_1 = diff(F3,beta_1);
dF3dbeta_2 = diff(F3,beta_2);
dF3dbeta_3 = diff(F3,beta_3);

dFrdx1 = diff(Fr,x1);
dFrdx2 = diff(Fr,x2);
dFrdT  = diff(Fr,T);
dFrdbeta_1 = diff(Fr,beta_1);
dFrdbeta_2 = diff(Fr,beta_2);
dFrdbeta_3 = diff(Fr,beta_3);

dF1dx1 = matlabFunction(dF1dx1); %@(S1,S2,T,beta_1,beta_2,beta_3,mu,x1,x2)
dF1dx2 = matlabFunction(dF1dx2); %@(S2,T,beta_2,mu,x1,x2)
dF1dT  = matlabFunction(dF1dT); %@(S1,S2,T,beta_1,beta_2,beta_3,mu,x1,x2)
dF1dbeta_1 = matlabFunction(dF1dbeta_1); %@(S1,T,mu,x1)
dF1dbeta_2 = matlabFunction(dF1dbeta_2); %@(S2,T,mu,x1,x2)
dF1dbeta_3 = matlabFunction(dF1dbeta_3); %@(T,mu,x1)

dF2dx1 = matlabFunction(dF2dx1); %@(S1,T,beta_1,mu,x1,x2)
dF2dx2 = matlabFunction(dF2dx2); %@(S1,S2,T,beta_1,beta_2,beta_3,mu,x1,x2)
dF2dT  = matlabFunction(dF2dT);  %@(S1,S2,T,beta_1,beta_2,beta_3,mu,x1,x2)
dF2dbeta_1 = matlabFunction(dF2dbeta_1); %@(S1,T,mu,x1,x2)
dF2dbeta_2 = matlabFunction(dF2dbeta_2); %@(S2,T,mu,x2)
dF2dbeta_3 = matlabFunction(dF2dbeta_3); %@(T,mu,x2)

dF3dx1 = matlabFunction(dF3dx1); %@(S1,T,beta_1,mu,x1)
dF3dx2 = matlabFunction(dF3dx2); %@(S2,T,beta_2,mu,x2)
dF3dT  = matlabFunction(dF3dT);  %@(S1,S2,T,beta_1,beta_2,beta_3,mu,x1,x2)
dF3dbeta_1 = matlabFunction(dF3dbeta_1); %@(S1,T,mu,x1)
dF3dbeta_2 = matlabFunction(dF3dbeta_2); %@(S2,T,mu,x2)
dF3dbeta_3 = matlabFunction(dF3dbeta_3); %@(T,mu)

dFrdx1 = matlabFunction(dFrdx1); %@(beta_1,beta_2,beta_3,mu,x1,x2)
dFrdx2 = matlabFunction(dFrdx2); %@(beta_1,beta_2,beta_3,mu,x1,x2)
dFrdT  = matlabFunction(dFrdT); %@() identically 0
dFrdbeta_1 = matlabFunction(dFrdbeta_1); %@(x1,x2)
```

```
105 dFrdbeta_2 = matlabFunction(dFrdbeta_2); %@(x1,x2)
106 dFrdbeta_3 = matlabFunction(dFrdbeta_3); %@(x1,x2)
107
108 A1 = zeros(4);
109 A2 = zeros(4);
110 A3 = zeros(4);
111
112 A1(1,1) = dF1dx1(S_1,S_2,T_in,B1,B2,B3,MU,x_1,x_2);
113 A1(1,2) = dF1dbeta_1(S_1,T_in,MU,x_1);
114 A1(1,3) = dF1dbeta_2(S_2,T_in,MU,x_1,x_2);
115 A1(1,4) = dF1dbeta_3(T_in,MU,x_1);
116
117 A1(2,1) = dF2dx1(S_1,T_in,B1,MU,x_1,x_2);
118 A1(2,2) = dF2dbeta_1(S_1,T_in,MU,x_1,x_2);
119 A1(2,3) = dF2dbeta_2(S_2,T_in,MU,x_2);
120 A1(2,4) = dF2dbeta_3(T_in,MU,x_2);
121
122 A1(3,1) = dF3dx1(S_1,T_in,B1,MU,x_1);
123 A1(3,2) = dF3dbeta_1(S_1,T_in,MU,x_1);
124 A1(3,3) = dF3dbeta_2(S_2,T_in,MU,x_2);
125 A1(3,4) = dF3dbeta_3(T_in,MU);
126
127 A1(4,1) = dFrdx1(B1,B2,B3,MU,x_1,x_2);
128 A1(4,2) = dFrdbeta_1(x_1,x_2);
129 A1(4,3) = dFrdbeta_2(x_1,x_2);
130 A1(4,4) = dFrdbeta_3(x_1,x_2);
131
132 A2(1,1) = dF1dx2(S_2,T_in,B2,MU,x_1,x_2);
133 A2(1,2) = dF1dbeta_1(S_1,T_in,MU,x_1);
134 A2(1,3) = dF1dbeta_2(S_2,T_in,MU,x_1,x_2);
135 A2(1,4) = dF1dbeta_3(T_in,MU,x_1);
136
137 A2(2,1) = dF2dx2(S_1,S_2,T_in,B1,B2,B3,MU,x_1,x_2);
138 A2(2,2) = dF2dbeta_1(S_1,T_in,MU,x_1,x_2);
139 A2(2,3) = dF2dbeta_2(S_2,T_in,MU,x_2);
140 A2(2,4) = dF2dbeta_3(T_in,MU,x_2);
141
142 A2(3,1) = dF3dx2(S_2,T_in,B2,MU,x_2);
143 A2(3,2) = dF3dbeta_1(S_1,T_in,MU,x_1);
144 A2(3,3) = dF3dbeta_2(S_2,T_in,MU,x_2);
145 A2(3,4) = dF3dbeta_3(T_in,MU);
146
147 A2(4,1) = dFrdx2(B1,B2,B3,MU,x_1,x_2);
148 A2(4,2) = dFrdbeta_1(x_1,x_2);
149 A2(4,3) = dFrdbeta_2(x_1,x_2);
150 A2(4,4) = dFrdbeta_3(x_1,x_2);
151
152 A3(1,1) = dF1dT(S_1,S_2,T_in,B1,B2,B3,MU,x_1,x_2);
153 A3(1,2) = dF1dbeta_1(S_1,T_in,MU,x_1);
154 A3(1,3) = dF1dbeta_2(S_2,T_in,MU,x_1,x_2);
155 A3(1,4) = dF1dbeta_3(T_in,MU,x_1);
156
157 A3(2,1) = dF2dT(S_1,S_2,T_in,B1,B2,B3,MU,x_1,x_2);
158 A3(2,2) = dF2dbeta_1(S_1,T_in,MU,x_1,x_2);
159 A3(2,3) = dF2dbeta_2(S_2,T_in,MU,x_2);
160 A3(2,4) = dF2dbeta_3(T_in,MU,x_2);
161
162 A3(3,1) = dF3dT(S_1,S_2,T_in,B1,B2,B3,MU,x_1,x_2);
163 A3(3,2) = dF3dbeta_1(S_1,T_in,MU,x_1);
164 A3(3,3) = dF3dbeta_2(S_2,T_in,MU,x_2);
165 A3(3,4) = dF3dbeta_3(T_in,MU);
166
167 A3(4,1) = 0;
168 A3(4,2) = dFrdbeta_1(x_1,x_2);
169 A3(4,3) = dFrdbeta_2(x_1,x_2);
170 A3(4,4) = dFrdbeta_3(x_1,x_2);
171
172 r1 = min(abs(eigs(A1)));
```

```
r2 = min(abs(eigs(A2)));
r3 = min(abs(eigs(A3)));

FF1 = @(x1,x2,T,beta_1,beta_2,beta_3,mu,S1,S2)   x1*(  (a1*f1(x1,S1,T)/mu +...
 sigma_1-b1)*beta_1 +...
(a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 +...
 (a3*f3(T)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
(P11*f1(x1,S1,T)/mu-M11)*beta_1 - (P12*f2(x2,S2,T)/mu-M12)*beta_2 -...
 (P13*f3(T)/mu-M13)*beta_3 + M1r*mu;
FF2 = @(x1,x2,T,beta_1,beta_2,beta_3,mu,S1,S2)   x2*(  (a1*f1(x1,S1,T)/mu +...
 sigma_1-b1)*beta_1 +...
(a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 +...
 (a3*f3(T)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
(P21*f1(x1,S1,T)/mu-M21)*beta_1 - (P22*f2(x2,S2,T)/mu-M22)*beta_2 -...
 (P23*f3(T)/mu-M23)*beta_3 + M2r*mu;
FF3 = @(x1,x2,T,beta_1,beta_2,beta_3,mu,S1,S2)    T*(  (a1*f1(x1,S1,T)/mu +...
 sigma_1-b1)*beta_1 +...
(a2*f2(x2,S2,T)/mu+sigma_2-b2)*beta_2 +...
 (a3*f3(T)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
(P31*f1(x1,S1,T)/mu-M31)*beta_1 - (P32*f2(x2,S2,T)/mu-M32)*beta_2 -...
 (P33*f3(T)/mu-M33)*beta_3 + M3r*mu;

FFr = @(x1,x2,beta_1,beta_2,beta_3,mu) beta_1/g1(x1,x2) +...
 beta_2/g2(x1,x2) + beta_3/g3(x1,x2) + mu/gr(x1,x2) - 1;


y1 =  FF1(x_1,x_2,T_in,B1,B2,B3,MU,S_1,S_2);

y2 =  FF2(x_1,x_2,T_in,B1,B2,B3,MU,S_1,S_2);

y3 =  FF3(x_1,x_2,T_in,B1,B2,B3,MU,S_1,S_2);

y4 =  FFr(x_1,x_2,B1,B2,B3,MU);

y5 = r1;
y6 = r2;
y7 = r3;

z = [y1;y2;y3;y4;y5;y6;y7];
```

visualH.m

```
function [H,k_s,ro_s,sigma_s,x1_s,x2_s,T_s] =...
 visualH(n,H_in,X1_RANGE,X2_RANGE,T_RANGE,visu)

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r...
 S1_c S2_c k_s ro_s sigma_s Enz_con

Substr_con = [1 1];

count = 0

if nargin == 1

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
T_range = [0.01,0.35]

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);

```

```
a_T  = T_range(1);
b_T  = T_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));
T_s  = linspace(a_T,b_T,n(3));

H = zeros(n(1),n(2),n(3));

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3));

f = waitbar(0,'please wait..');

tot = n(1)*n(2)*n(3)

t_start = tic;

for i=1:n(1)
for j=1:n(2)
for k=1:n(3)
if ro_1*x1_s(i)+ro_2*x2_s(j)+ro_3*T_s(k) > 1
H(i,j,k)=0;
count=count+1
f = waitbar(count/tot,f);
else
H(i,j,k) = opt_M([x1_s(i),x2_s(j),T_s(k)]);
count = count + 1
f = waitbar(count/tot,f);
end
end
end
end

time = round(toc(t_start)/60,3)
close(f);

end

if nargin == 2

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
T_range = [0.01,0.35]

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);

a_T  = T_range(1);
b_T  = T_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));
T_s  = linspace(a_T,b_T,n(3));

figure

for i = 1:n(3)
subplot(floor(power(n(3),1/2)),floor(power(n(3),1/2)),i)

surf(x1_s,x2_s,-H_in(:,:,i))
height = [0,-min(H_in(:))];
axs = [x1_s(1) x1_s(end) x2_s(1) x2_s(end) height];
axis(axs);
az = 0;
el = 90;
view(az, el);
```

```
caxis([0 -min(H_in(:))])
colorbar
xlabel('x1');
ylabel('x2');
title(['Plot of the H Function in the x1-x2-T Space, T = ' num2str(T_s(i))]);
end

end

if nargin == 5

x1_range = X1_RANGE;
x2_range = X2_RANGE;
T_range = T_RANGE;

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);
a_T  = T_range(1);
b_T  = T_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));
T_s  = linspace(a_T,b_T,n(3));

H = zeros(n(1),n(1),n(1));

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3));

f = waitbar(0,'please wait..');

tot = n(1)*n(2)*n(3)

t_start = tic;

for i=1:n(1)
for j=1:n(2)
for k=1:n(3)
H(i,j,k) = opt_M([x1_s(i),x2_s(j),T_s(k)]);
count = count + 1;
f = waitbar(count/tot,f);
end
end
end

time = round(toc(t_start)/60,3)
close(f);

end

if nargin == 6

x1_range = X1_RANGE;
x2_range = X2_RANGE;
T_range = T_RANGE;

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);
a_T  = T_range(1);
b_T  = T_range(2);

x1_s = linspace(a_x1,b_x1,n(1));
x2_s = linspace(a_x2,b_x2,n(2));
T_s  = linspace(a_T,b_T,n(3));

```

```
figure
for i = 1:n(3)
subplot(floor(power(n(3),1/2))+1,floor(power(n(3),1/2))+1,i)
surf(x1_s,x2_s,-H_in(:,:,i))
height = [0,-min(H_in(:))];
axs = [x1_s(1) x1_s(end) x2_s(1) x2_s(end) height];
axis(axs);
az = 0;
el = 90;
view(az, el);
caxis([0 -min(H_in(:))])
colorbar
end
end

end
```

Toxin_System.m

```
function [dx]=Toxin_System(t,x,alpha_s)

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s sigma_s Enz_con

Substr_con = [1 1];

alpha_1 = alpha_s(1);
alpha_2 = alpha_s(2);
alpha_3 = alpha_s(3);
alpha_r = alpha_s(4);

S1 = Substr_con(1);
S2 = Substr_con(2);

f1 = kcat*(1/(k_T+x(3)))*((S1-k1_f*x(1))/(1+S1+x(1)));
f2 = kcat*((S2-k2_f*x(2)*x(3))/(1+S2+x(2)+x(3)+x(3)*x(2)));
f3 = k3_fd*(x(3)/(1+x(3)));

g1 = k1_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
g2 = k2_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
g3 = k3_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));
gr = kr_g*x(1)*x(2)/(1+x(1)+x(2)+x(1)*x(2));

V1 = x(4)*f1;
V2 = x(5)*f2;
V3 = x(6)*f3;

W1 = x(7)*alpha_1*g1;
W2 = x(7)*alpha_2*g2;
W3 = x(7)*alpha_3*g3;
Wr = x(7)*alpha_r*gr;

MU = ro_1*(P11*V1+P12*V2+P13*V3)+ro_2*(P21*V1+P22*V2+P23*V3)+...
ro_3*(P31*V1+P32*V2+P33*V3) + ...
(sigma_1-(ro_1*M11+ro_2*M21+ro_3*M31))*W1 +...
 (sigma_2-(ro_1*M12+ro_2*M22+ro_3*M32))*W2 + ...
(sigma_3-(ro_1*M13+ro_2*M23+ro_3*M33))*W3 +...
 (sigma_r-(ro_1*M1r+ro_2*M2r+ro_3*M3r))*Wr;


dx1 = V1 - M11*W1 - M12*W2 - M13*W3 - M1r*Wr - MU*x(1);
dx2 = V2 - M21*W1 - M22*W2 - M23*W3 - M2r*Wr - MU*x(2);
dT  = V2 - V3 - MU*x(3);
de1 = W1 - MU*x(4);
```

```
de2 = W2 - MU*x(5);
de3 = W3 - MU*x(6);
dr  = Wr - MU*x(7);

[ro_s sigma_s] * x;

dx=[dx1;dx2;dT;de1;de2;de3;dr];
```

Run_System.m

```
function [q] = Run_System(alpha_s,xinit);

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s sigma_s Enz_con

Substr_con = [1 1];

if nargin == 0
alpha1 = rand;
alpha2 = rand;
alpha3 = rand;
alpha4 = rand;

alpha_1 = alpha1/(alpha1+alpha2+alpha3+alpha4);
alpha_2 = alpha2/(alpha1+alpha2+alpha3+alpha4);
alpha_3 = alpha3/(alpha1+alpha2+alpha3+alpha4);
alpha_4 = alpha4/(alpha1+alpha2+alpha3+alpha4);

alpha_s = [alpha_1,alpha_2,alpha_3,alpha_4];

xinit = [0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1];                    % Initial condition

else
alpha_1 = alpha_s(1);
alpha_2 = alpha_s(2);
alpha_3 = alpha_s(3);
alpha_4 = alpha_s(4);
end

S1 = Substr_con(1);
S2 = Substr_con(2);

t=[0 1200];                                                   % Time window

% somma = [ro_s sigma_s] * xinit;
% xinit = xinit/somma;
x = xinit;                                                    % Initial condition
[t,y]=ode45(@(t,x) Toxin_System_Dummy(t,x,alpha_s),t,xinit);     % Integrate in time

s = size(y);
q = y(s(1),:);
x = q;                  % steady state found

subplot(3,3,1);

plot(t,y(:,1))
%title('metabs evolution');
%xlabel('Time t');
%ylabel('metabolites');
legend('x1')

hold on

subplot(3,3,2);
```

```
57
58 plot(t,y(:,2))
59 %title('metabs evolution');
60 %xlabel('Time t');
61 %ylabel('metabolites');
62 legend('x2')
63
64 hold on
65
66 subplot(3,3,3);
67
68 plot(t,y(:,3))
69 %title('enzymes evolution');
70 %xlabel('Time t');
71 %ylabel('enzymes concentrations');
72 legend('T')
73
74 hold on
75
76 subplot(3,3,4);
77
78 plot(t,y(:,4))
79 %title('enzymes evolution');
80 %xlabel('Time t');
81 %ylabel('enzymes concentrations');
82 legend('e1')
83
84 hold on
85
86 subplot(3,3,5);
87
88 plot(t,y(:,5))
89 %title('enzymes evolution');
90 %xlabel('Time t');
91 %ylabel('enzymes concentrations');
92 legend('e2')
93
94 hold on
95
96 subplot(3,3,6);
97
98 plot(t,y(:,6))
99 %title('enzymes evolution');
100 %xlabel('Time t');
101 %ylabel('enzymes concentrations');
102 legend('e3')
103
104 hold on
105
106 subplot(3,3,7);
107
108 plot(t,y(:,7))
109 %title('enzymes evolution');
110 %xlabel('Time t');
111 %ylabel('enzymes concentrations');
112 legend('r')
113
114 hold on
```

cal_fixed_alphas.m

```
1
2 function [x1,x2,T,mu,al1,al2,al3,e1,e2,e3,r] = cal_fixed_alphas(alpha1,alpha2,alpha3);
3
4 global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
5  P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
```

```
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_fd k1_g k2_g k3_g kr_g k_T...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c ro_s k_s sigma_s Enz_con

IC = [0.3;0.3;0.3;0.3;0.3;0.3;0.3;0.3];
som = [ro_s sigma_s] * IC(1:7);
IC = IC/som;
[ro_s sigma_s] * IC(1:7);

options = odeset('AbsTol',1e-9,'RelTol',1e-9);

alpha_r = 1 - alpha1 - alpha2 - alpha3;

if alpha_r < 0
x1=0;
x2=0;
T=0;
mu=0;
e1=0;
e2=0;
e3=0;
r=0;
else
rhs = @(t,y) Toxin_System_Dummy(t,y,[alpha1 alpha2 alpha3 alpha_r]);
[~,y] = ode15s(rhs,[0,1000],IC,options);

x1 = y(end,1);
x2 = y(end,2);
T = y(end,3);
e1 = y(end,4);
e2 = y(end,5);
e3 = y(end,6);
r  = y(end,7);
mu = -y(end,8);
end

al1 = alpha1;
al2 = alpha2;
al3 = alpha3;

end
```

DummyFunction1.m

```

function out4 = DummyFunction1(alpha1,alpha2,alpha3)
[~,~,~,out4,~,~,~,~,~,~,~] = cal_fixed_alphas(alpha1,alpha2,alpha3);
end
```

run_script.m

```


global k1_g k2_g k3_g S1_c kcat S2_c ro_1

DummyFunction2 = @(X) DummyFunction1(X(1),X(2),X(3));

options = optimset('Display','iter','PlotFcns',@optimplotfval,'TolFun',0.00001,'maxiter',100);

y = [0.3,0.3,0.3];

%[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

N = 200;

```

```
sub1 = linspace(0.1,2.5,N);

zopts = zeros(15,N);

tic

f = waitbar(0,'please wait..');


for i = 1:N

ro_1 = sub1(i);

f = waitbar(i/N,f);

[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

[x1,x2,T,mu,al1,al2,al3,e1,e2,e3,r] = cal_fixed_alphas(x(1),x(2),x(3))

zopts(1,i) = x1;
zopts(2,i) = x2;
zopts(3,i) = T;
zopts(4,i) = e1;
zopts(5,i) = e2;
zopts(6,i) = e3;
zopts(7,i) = r;

zopts(8,i) = al1;
zopts(9,i) = al2;
zopts(10,i) = al3;
zopts(11,i) = al1*k1_g*x1*x2/(1+x1+x2+x1*x2);
zopts(12,i) = al2*k2_g*x1*x2/(1+x1+x2+x1*x2);
zopts(13,i) = al3*k3_g*x1*x2/(1+x1+x2+x1*x2);
zopts(14,i) = -mu;

zopts(15,i) = sub1(i);

end

close(f);

toc
```

A.0.3 Toxin Presence Leads To Substrate Use Switch codes

optH.m

```
function [mu_opt,exitflag,b_s_opt,Aeq,exit]=optH(Substr_con,x1,x2,x3,mu_tilde,T);

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
kcat k1_f k2_f k3_f k1_g k2_g k3_g kr_g sigma_1 sigma_2 sigma_3 sigma_r

S1 = Substr_con(1);
S2 = Substr_con(2);
S3 = Substr_con(3);

f1 = @(x1,S1) 5 * kcat * (1/(1+ T)) * ((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2)       kcat *              ((S2-k2_f*x2)/(1+S2+x2));
f3 = @(x3,S3)       kcat *              ((S3-k3_f*x3)/(1+S3+x3));


g1 = @(x1,x2,x3) k1_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
g2 = @(x1,x2,x3) k2_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
g3 = @(x1,x2,x3) k3_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
```

```
gr = @(x1,x2,x3) kr_g*x1*x2/(1+x1+x2+x1*x2) + kr_g*x2*x3/(1+x2+x3+x2*x3);

%{
g1 = @(x1,x2,x3) k1_g*x1*x2/(1+x1+x2+x1*x2) + k1_g*x2*x3/(1+x2+x3+x2*x3);
g2 = @(x1,x2,x3) k2_g*x1*x2/(1+x1+x2+x1*x2) + k2_g*x2*x3/(1+x2+x3+x2*x3);
g3 = @(x1,x2,x3) k3_g*x1*x2/(1+x1+x2+x1*x2) + k3_g*x2*x3/(1+x2+x3+x2*x3);
gr = @(x1,x2,x3) kr_g*x1*x2/(1+x1+x2+x1*x2) + kr_g*x2*x3/(1+x2+x3+x2*x3);
%}
f = [0,0,0,-1];
B = -eye(4);
b = [0;0;0;0];
beq = [0;0;0;1];

Aeq = zeros(4);

Aeq(1,1) = x1*(a1*f1(x1,S1)/mu_tilde + sigma_1-b1) - (P11*f1(x1,S1)/mu_tilde-M11);
Aeq(1,2) = x1*(a2*f2(x2,S2)/mu_tilde + sigma_2-b2) - (P12*f2(x2,S2)/mu_tilde-M12);
Aeq(1,3) = x1*(a3*f3(x3,S3)/mu_tilde + sigma_3-b3) - (P13*f3(x3,S3)/mu_tilde-M13);
Aeq(1,4) = x1*(sigma_r-br)+ M1r;

Aeq(2,1) = x2*(a1*f1(x1,S1)/mu_tilde + sigma_1-b1) - (P21*f1(x1,S1)/mu_tilde-M21);
Aeq(2,2) = x2*(a2*f2(x2,S2)/mu_tilde + sigma_2-b2) - (P22*f2(x2,S2)/mu_tilde-M22);
Aeq(2,3) = x2*(a3*f3(x3,S3)/mu_tilde + sigma_3-b3) - (P23*f3(x3,S3)/mu_tilde-M23);
Aeq(2,4) = x2*(sigma_r-br) + M2r;

Aeq(3,1) = x3*(a1*f1(x1,S1)/mu_tilde + sigma_1-b1) - (P31*f1(x1,S1)/mu_tilde-M31);
Aeq(3,2) = x3*(a2*f2(x2,S2)/mu_tilde + sigma_2-b2) - (P32*f2(x2,S2)/mu_tilde-M32);
Aeq(3,3) = x3*(a3*f3(x3,S3)/mu_tilde + sigma_3-b3) - (P33*f2(x3,S3)/mu_tilde-M33);
Aeq(3,4) = x3*(sigma_r-br) + M3r;

Aeq(4,1)=1/g1(x1,x2,x3);
Aeq(4,2)=1/g2(x1,x2,x3);
Aeq(4,3)=1/g3(x1,x2,x3);
Aeq(4,4)=1/gr(x1,x2,x3);

options = optimset('Display','none','maxiter',100);

LB = [];
UB = [];

[b_s_opt,fval,exitflag,output] = linprog(f,B,b,Aeq,beq,LB,UB,[],options);

if exitflag == 1 || exitflag == 0
mu_opt=b_s_opt(4);
exit = 1;
else
mu_opt=0;
b_s_opt = [0;0;0;0];
exit = 0;
end

[mu_opt mu_tilde];
```

fixed_pt.m

```
 function [s]=fixed_pt(Substr_con,x1,x2,x3,T);

 fixedpt = @(mu_tilde) optH(Substr_con,x1,x2,x3,mu_tilde,T) - mu_tilde;
 options = optimset('TolX',1e-7,'TolFun',1e-7,'maxiter',250);
 s = -fzero(fixedpt,1e-9,options);
```

Maximizer.m

```
function [B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH] =...
```

```
 Maximizer(Substr_con,x1_range,x2_range,x3_range,max_iter,precision,T)

n=5;

count = 0;

a_x1 = x1_range(1);
b_x1 = x1_range(2);
a_x2 = x2_range(1);
b_x2 = x2_range(2);
a_x3 = x3_range(1);
b_x3 = x3_range(2);

x1_s = linspace(a_x1,b_x1,n);
x2_s = linspace(a_x2,b_x2,n);
x3_s  = linspace(a_x3,b_x3,n);

H = zeros(n,n,n);

opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3),T);

f = waitbar(0,'please wait..');

for i=1:n
for j=1:n
for k=1:n
x1 = x1_s(i);
x2 = x2_s(j);
x3 = x3_s(k);
y = [x1,x2,x3];
H(i,j,k) = opt_M(y);
count = count + 1;
f = waitbar(count/125,f);
end
end
end

close(f)

count = 0;

[M,I] = min(H(:));
[I_row, I_col, I_page] = ind2sub(size(H),I);

disp('Point found from the grid:')

y = [x1_s(I_row),x2_s(I_col),x3_s(I_page)]

disp('Coordinates of the point:')

coor = [I_row, I_col, I_page]

disp('With minimum:')

optm = opt_M(y)

disp('And precision:')

pres = max([abs(a_x1-b_x1),abs(a_x2-b_x2),abs(a_x3-b_x3)])/4

while  pres > precision

if I_row == 1
a_x1 = x1_s(I_row);
b_x1 = x1_s(I_row+1);
else
if I_row == n
a_x1 = x1_s(I_row-1);
```

```
b_x1 = x1_s(I_row);
else
a_x1 = x1_s(I_row-1);
b_x1 = x1_s(I_row+1);
end
end


if I_col == 1
a_x2 = x2_s(I_col);
b_x2 = x2_s(I_col+1);
else
if I_col == n
a_x2 = x2_s(I_col-1);
b_x2 = x2_s(I_col);
else
a_x2 = x2_s(I_col-1);
b_x2 = x2_s(I_col+1);
end
end


if I_page == 1
a_x3 = x3_s(I_page);
b_x3 = x3_s(I_page+1);
else
if I_page == n
a_x3 = x3_s(I_page-1);
b_x3 = x3_s(I_page);
else
a_x3 = x3_s(I_page-1);
b_x3 = x3_s(I_page+1);
end
end

x1_s = linspace(a_x1,b_x1,n);
x2_s = linspace(a_x2,b_x2,n);
x3_s  = linspace(a_x3,b_x3,n);

H = zeros(n,n,n);

f = waitbar(0,'please wait..');

for i=1:n
for j=1:n
for k=1:n
x1 = x1_s(i);
x2 = x2_s(j);
x3 = x3_s(k);
y = [x1,x2,x3];
H(i,j,k) = opt_M(y);
count = count + 1;
f = waitbar(count/125,f);
end
end
end

close(f)

[M,I] = min(H(:));
[I_row, I_col, I_page] = ind2sub(size(H),I);
count = 0;

disp('Point found from the grid')

y = [x1_s(I_row),x2_s(I_col),x3_s(I_page)]

disp('With minimum:')
```

```
138
139 optm = opt_M(y)
140
141 disp('And precision:')
142
143 pres = max([abs(a_x1-b_x1),abs(a_x2-b_x2),abs(a_x3-b_x3)])/4
144
145 end
146
147
148 disp('fminsearch starts')
149
150 options = optimset('Display','iter','PlotFcns',...
151 @optimplotfval,'TolFun',0.000001,'maxiter',max_iter);
152
153 [x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)
154
155 [mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),x(3),-fval,T);
156
157 B_S_OPT = b_s_opt;
158 X_OPT = x;
159 MU_OPT = mu_opt;
160
161 end
```

SOL.m

```
1  function [x0,k_s,ro_s,sigma_s] =...
2   SOL(rndmz,e,T,Substr_con,max_iter,precision,method,in_point)
3
4  global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
5   P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
6  ro_1 ro_2 ro_3 kcat k1_f k2_f k3_f k1_g k2_g k3_g kr_g...
7   sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c S3_c
8
9  %%%%%%%%%%%% PROBLEM DESCRIPTION %%%%%%%%%%%%%%%%
10
11 disp('Problem description:')
12
13 Network = [1,0,0;
14 0,1,0;
15 0,0,1];
16
17 Enz_con = [3,2,4,1;
18 2,1,3,1;
19 4,3,1,2];
20
21 System = [1,0,0,1,1,1,1;
22 0,1,0,1,1,1,1;
23 0,0,1,1,1,1,1;
24 0,0,0,1,0,0,0;
25 0,0,0,0,1,0,0;
26 0,0,0,0,0,1,0;
27 0,0,0,0,0,0,1];
28
29 if rndmz == 'Y'
30
31 if nargin == 1
32 e=0.05;
33 end
34
35 disp('Vector of randomization:')
36
37 rad = [ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ...
38 ran(e) , ran(e) , ran(e) , ran(e), ran(e) , ran(e) , ran(e)] %14 values
39
```

```
disp('Original constants:')

k_s = [0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5] % 8 values

ro_s = [1,1,1]

sigma_s = [1,1,1,1]

disp('Perturbed values:')

k_s = [0.25 , 0.5+rad(1)/2 , 0.5+rad(2)/2 , 0.5+rad(3)/2 ,...
 0.5+rad(4)/2 , 0.5+rad(5)/2 , 0.5+rad(6)/2 , 0.5+rad(7)/2]

ro_s = [1+rad(8),1+rad(9),1+rad(10)]

sigma_s = [1+rad(11),1+rad(12),1+rad(13),1+rad(14)]

else

disp('Using fixed constants')

k_s = [0.2500    0.4981    0.5023    0.4963    0.4998    0.4977    0.5134    0.5089]
%[0.2500    0.4881    0.5023    0.4863    0.4998    0.4977    0.5234    0.5089]

ro_s = [1.0711    1.0404    1.0225] %[0.9711    1.0404    1.0225]

sigma_s = [1.0992    1.0311    1.0748    1.0167]

end

%%%%%%%%%%%%%% CONSTANTS %%%%%%%%%%%%%%

M11 = Enz_con(1,1);
M12 = Enz_con(1,2);
M13 = Enz_con(1,3);
M1r = Enz_con(1,4);
M21 = Enz_con(2,1);
M22 = Enz_con(2,2);
M23 = Enz_con(2,3);
M2r = Enz_con(2,4);
M31 = Enz_con(3,1);
M32 = Enz_con(3,2);
M33 = Enz_con(3,3);
M3r = Enz_con(3,4);

P11 = Network(1,1);
P12 = Network(1,2);
P13 = Network(1,3);
P21 = Network(2,1);
P22 = Network(2,2);
P23 = Network(2,3);
P31 = Network(3,1);
P32 = Network(3,2);
P33 = Network(3,3);

kcat = k_s(1);
k1_f = k_s(2);
k2_f = k_s(3);
k3_f= k_s(4);
k1_g = k_s(5);
k2_g = k_s(6);
k3_g = k_s(7);
kr_g = k_s(8);

ro_1 = ro_s(1);
ro_2 = ro_s(2);
ro_3 = ro_s(3);

```

```
sigma_1 = sigma_s(1);
sigma_2 = sigma_s(2);
sigma_3 = sigma_s(3);
sigma_r = sigma_s(4);

S1_c = Substr_con(1);
S2_c = Substr_con(2);
S3_c = Substr_con(3);

a1 = ro_1*P11 + ro_2*P21 + ro_3*P31;
a2 = ro_1*P12 + ro_2*P22 + ro_3*P32;
a3 = ro_1*P13 + ro_2*P23 + ro_3*P33;

b1=ro_1*M11 + ro_2*M21 + ro_3*M31;
b2=ro_1*M12 + ro_2*M22 + ro_3*M32;
b3=ro_1*M13 + ro_2*M23 + ro_3*M33;
br=ro_1*M1r + ro_2*M2r + ro_3*M3r;

%%%%%%%%%%%%%%%%%%%%%%%%%% STARTS OPTIMIZATION %%%%%%%%%%%%%%%%%%%%%%%%%%

n = 5;

if nargin == 7          %%%%%%%% SEARCH WITH STANDARD GRID %%%%%%%%%%%%%%%%%%

disp('Please wait! searching for a maximum with a grid (125)...
 over the x_s with ranges for x1 and x2 and T,...
  [0 , S1/k_1] , [0 , S2/k_2] and [0 , S3/k_3]:')

x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
x3_range = [0.01,min(S3_c/k_s(4),1/ro_s(3))]

t_start = tic;

[B_S_OPT,X_OPT,exitflag_fmin,exitflag_optH] =...
 Maximizer(Substr_con,x1_range,x2_range,x3_range,max_iter,precision,T)

if B_S_OPT(3) == 0
disp('Optimization didnt succeed with given starting point')
else
disp('optimization succeeded')
end

disp('Minutes it took to find a solution:')

time = round(toc(t_start)/60,3)

disp('Solution vector [x1 x2 x3 beta1 beta2 beta3 mu] :')

x0 = [X_OPT , B_S_OPT.']


disp('Zeros of balanced growth and optimality equations:')

contG(x0)
%{
fun = @contG;

if method == 'fs'

disp('Point found by fsolve:')

z = fsolve(fun,x0)

disp('Image of the point found:')

contG(z)

```

```
176 else
177
178 disp('Point found by lsqnonlin:')
179
180 lb = zeros(size(x0)); %lower bound of zero
181 ub = Inf*ones(size(x0));
182 z = lsqnonlin(fun,x0,lb,ub)
183
184 end
185 %}
186 end
187
188
189 if nargin == 8        %%%%%%%%%% SEARCH WITH GIVEN POINT %%%%%%%%%%%%
190
191 disp('fminsearch starts with user-given starting point:')
192
193 y = in_point
194
195 opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3));
196
197 options = optimset('Display','iter','PlotFcns',...
198 @optimplotfval,'TolFun',0.001,'maxiter',max_iter);
199
200 [x,fval,exitflag_fmin,output] = fminsearch(opt_M,y,options)
201
202 [mu_opt,exitflag_optH,b_s_opt]=optH(Substr_con,x(1),x(2),x(3),-fval);
203
204 B_S_OPT = b_s_opt;
205 X_OPT = x;
206 MU_OPT = mu_opt;
207
208 if MU_OPT == 0
209 disp('Optimization didnt succeed with given starting point')
210 else
211 disp('Optimization succeeded')
212 end
213
214 disp('Solution vector [x1 x2 x3 beta1 beta2 beta3 mu] :')
215
216 x0 = [X_OPT , B_S_OPT.']
217 %{
218 disp('Zeros of steady state and optimality equations:')
219
220 contG(x0)
221
222 fun = @contG;
223
224 if method == 'fs'
225
226 disp('Point found by fsolve:')
227
228 z = fsolve(fun,x0)
229
230 disp('Image of the point found:')
231
232 contG(z)
233
234 else
235
236 disp('Point found by lsqnonlin:')
237
238 lb = zeros(size(x0)); %lower bound of zero
239 ub = Inf*ones(size(x0));
240 z = lsqnonlin(fun,x0,lb,ub)
241
242 end
243 %}
```

```
end

end
```

contG.m

```
function [z] = contG(x);
%used to get a point which is on the curve to start the continuation

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
kcat k1_f k2_f k3_f k1_g k2_g k3_g kr_g...
 sigma_1 sigma_2 sigma_3 sigma_r S1_c S2_c S3_c T

S_1 = S1_c;
S_2 = S2_c;
S_3 = S3_c;

x_1 = x(1);
x_2 = x(2);
x_3 = x(3);
B1 = x(4);
B2 = x(5);
B3 = x(6);
MU = x(7);

syms x1 x2 x3 S1 S2 S3 mu beta_1 beta_2 beta_3

f1 = @(x1,S1) 5 * kcat * (1/(1+ T)) * ((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2) kcat * ((S2-k2_f*x2)/(1+S2+x2));
f3 = @(x3,S3) kcat * ((S3-k3_f*x3)/(1+S3+x3));

g1 = @(x1,x2,x3) k1_g*x1*x2/(1+x1+x2+x1*x2) + k1_g*x2*x3/(1+x2+x3+x2*x3);
g2 = @(x1,x2,x3) k2_g*x1*x2/(1+x1+x2+x1*x2) + k2_g*x2*x3/(1+x2+x3+x2*x3);
g3 = @(x1,x2,x3) k3_g*x1*x2/(1+x1+x2+x1*x2) + k3_g*x2*x3/(1+x2+x3+x2*x3);
gr = @(x1,x2,x3) kr_g*x1*x2/(1+x1+x2+x1*x2) + kr_g*x2*x3/(1+x2+x3+x2*x3);


%{
f1 = @(x1,S1) kcat * ((S1-k1_f*x1)/(1+S1+x1));
f2 = @(x2,S2) kcat * ((S2-k2_f*x2)/(1+S2+x2));
f3 = @(x3,S3) kcat * ((S3-k3_f*x3)/(1+S3+x3));

g1 = @(x1,x2,x3) k1_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
g2 = @(x1,x2,x3) k2_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
g3 = @(x1,x2,x3) k3_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
gr = @(x1,x2,x3) kr_g*x1*x2/(1+x1+x2+x1*x2) + kr_g*x2*x3/(1+x2+x3+x2*x3);
%}

F1 = @(beta_1,beta_2,beta_3,mu)   x1*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
 (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 + (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 +...
  (sigma_r-br)*mu  ) - (P11*f1(x1,S1)/mu-M11)*beta_1 - (P12*f2(x2,S2)/mu-M12)*beta_2 -...
   (P13*f3(x3,S3)/mu-M13)*beta_3 + M1r*mu;
F2 = @(beta_1,beta_2,beta_3,mu)   x2*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
 (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 + (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 +...
  (sigma_r-br)*mu  ) - (P21*f1(x1,S1)/mu-M21)*beta_1 - (P22*f2(x2,S2)/mu-M22)*beta_2 -...
   (P23*f3(x3,S3)/mu-M23)*beta_3 + M2r*mu;
F3 = @(beta_1,beta_2,beta_3,mu)   x3*(  (a1*f1(x1,S1)/mu + sigma_1-b1)*beta_1 +...
 (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 + (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 +...
  (sigma_r-br)*mu  ) - (P31*f1(x1,S1)/mu-M31)*beta_1 - (P32*f2(x2,S2)/mu-M32)*beta_2 -...
   (P33*f3(x3,S3)/mu-M33)*beta_3 + M3r*mu;

Fr = @(beta_1,beta_2,beta_3,mu) beta_1/g1(x1,x2,x3) +...
 beta_2/g2(x1,x2,x3) + beta_3/g3(x1,x2,x3) + mu/gr(x1,x2,x3) - 1;

dF1dx1 = diff(F1,x1);
```

```
dF1dx2 = diff(F1,x2);
dF1dx3 = diff(F1,x3);
dF1dbeta_1 = diff(F1,beta_1);
dF1dbeta_2 = diff(F1,beta_2);
dF1dbeta_3 = diff(F1,beta_3);

dF2dx1 = diff(F2,x1);
dF2dx2 = diff(F2,x2);
dF2dx3 = diff(F2,x3);
dF2dbeta_1 = diff(F2,beta_1);
dF2dbeta_2 = diff(F2,beta_2);
dF2dbeta_3 = diff(F2,beta_3);

dF3dx1 = diff(F3,x1);
dF3dx2 = diff(F3,x2);
dF3dx3  = diff(F3,x3);
dF3dbeta_1 = diff(F3,beta_1);
dF3dbeta_2 = diff(F3,beta_2);
dF3dbeta_3 = diff(F3,beta_3);

dFrdx1 = diff(Fr,x1);
dFrdx2 = diff(Fr,x2);
dFrdx3 = diff(Fr,x3);
dFrdbeta_1 = diff(Fr,beta_1);
dFrdbeta_2 = diff(Fr,beta_2);
dFrdbeta_3 = diff(Fr,beta_3);

dF1dx1 = matlabFunction(dF1dx1); %@(S1,S2,S3,beta_1,beta_2,beta_3,mu,x1,x2,x3)
dF1dx2 = matlabFunction(dF1dx2); %@(S2,beta_2,mu,x1,x2)
dF1dx3  = matlabFunction(dF1dx3); %@(S3,beta_3,mu,x1,x2)
dF1dbeta_1 = matlabFunction(dF1dbeta_1); %@(S1,mu,x1)
dF1dbeta_2 = matlabFunction(dF1dbeta_2); %@(S2,mu,x1,x2)
dF1dbeta_3 = matlabFunction(dF1dbeta_3); %@(S3,mu,x3)

dF2dx1 = matlabFunction(dF2dx1); %@(S1,beta_1,mu,x1,x2)
dF2dx2 = matlabFunction(dF2dx2); %@(S1,S2,S3,beta_1,beta_2,beta_3,mu,x1,x2,x3)
dF2dx3  = matlabFunction(dF2dx3);  %@(S3,beta_3,mu,x2,x3)
dF2dbeta_1 = matlabFunction(dF2dbeta_1); %@(S1,mu,x1,x2)
dF2dbeta_2 = matlabFunction(dF2dbeta_2); %@(S2,mu,x2)
dF2dbeta_3 = matlabFunction(dF2dbeta_3); %@(S3,mu,x2,x3)

dF3dx1 = matlabFunction(dF3dx1); %@(S1,beta_1,mu,x1,x3)
dF3dx2 = matlabFunction(dF3dx2); %@(S2,beta_2,mu,x2,x3)
dF3dx3  = matlabFunction(dF3dx3);  %@(S1,S2,S3,beta_1,beta_2,beta_3,mu,x1,x2,x3)
dF3dbeta_1 = matlabFunction(dF3dbeta_1); %@(S1,mu,x1,x3)
dF3dbeta_2 = matlabFunction(dF3dbeta_2); %@(S2,mu,x2,x3)
dF3dbeta_3 = matlabFunction(dF3dbeta_3); %@(S3,mu,x3)

dFrdx1 = matlabFunction(dFrdx1); %@(beta_1,beta_2,beta_3,mu,x1,x2,x3)
dFrdx2 = matlabFunction(dFrdx2); %@(beta_1,beta_2,beta_3,mu,x1,x2,x3)
dFrdx3  = matlabFunction(dFrdx3); %@(beta_1,beta_2,beta_3,mu,x1,x2,x3)
dFrdbeta_1 = matlabFunction(dFrdbeta_1); %@(x1,x2,x3)
dFrdbeta_2 = matlabFunction(dFrdbeta_2); %@(x1,x2,x3)
dFrdbeta_3 = matlabFunction(dFrdbeta_3); %@(x1,x2,x3)

A1 = zeros(4);
A2 = zeros(4);
A3 = zeros(4);

A1(1,1) = dF1dx1(S_1,S_2,S_3,B1,B2,B3,MU,x_1,x_2,x_3);
A1(1,2) = dF1dbeta_1(S_1,MU,x_1);
A1(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);
A1(1,4) = dF1dbeta_3(S_3,MU,x_1,x_3);

A1(2,1) = dF2dx1(S_1,B1,MU,x_1,x_2);
A1(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
A1(2,3) = dF2dbeta_2(S_2,MU,x_2);
A1(2,4) = dF2dbeta_3(S_3,MU,x_2,x_3);
```

```
129
130 A1(3,1) = dF3dx1(S_1,B1,MU,x_1,x_3);
131 A1(3,2) = dF3dbeta_1(S_1,MU,x_1,x_3);
132 A1(3,3) = dF3dbeta_2(S_2,MU,x_2,x_3);
133 A1(3,4) = dF3dbeta_3(S_3,MU,x_3);
134
135 A1(4,1) = dFrdx1(B1,B2,B3,MU,x_1,x_2,x_3);
136 A1(4,2) = dFrdbeta_1(x_1,x_2,x_3);
137 A1(4,3) = dFrdbeta_2(x_1,x_2,x_3);
138 A1(4,4) = dFrdbeta_3(x_1,x_2,x_3);
139
140 A2(1,1) = dF1dx2(S_2,B2,MU,x_1,x_2);
141 A2(1,2) = dF1dbeta_1(S_1,MU,x_1);
142 A2(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);
143 A2(1,4) = dF1dbeta_3(S_3,MU,x_1,x_3);
144
145 A2(2,1) = dF2dx2(S_1,S_2,S_3,B1,B2,B3,MU,x_1,x_2,x_3);
146 A2(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
147 A2(2,3) = dF2dbeta_2(S_2,MU,x_2);
148 A2(2,4) = dF2dbeta_3(S_3,MU,x_2,x_3);
149
150 A2(3,1) = dF3dx2(S_2,B2,MU,x_2,x_3);
151 A2(3,2) = dF3dbeta_1(S_1,MU,x_1,x_3);
152 A2(3,3) = dF3dbeta_2(S_2,MU,x_2,x_3);
153 A2(3,4) = dF3dbeta_3(S_3,MU,x_3);
154
155 A2(4,1) = dFrdx2(B1,B2,B3,MU,x_1,x_2,x_3);
156 A2(4,2) = dFrdbeta_1(x_1,x_2,x_3);
157 A2(4,3) = dFrdbeta_2(x_1,x_2,x_3);
158 A2(4,4) = dFrdbeta_3(x_1,x_2,x_3);
159
160 A3(1,1) = dF1dx3(S_3,B3,MU,x_1,x_3);
161 A3(1,2) = dF1dbeta_1(S_1,MU,x_1);
162 A3(1,3) = dF1dbeta_2(S_2,MU,x_1,x_2);
163 A3(1,4) = dF1dbeta_3(S_3,MU,x_1,x_3);
164
165 A3(2,1) = dF2dx3(S_3,B3,MU,x_2,x_3);
166 A3(2,2) = dF2dbeta_1(S_1,MU,x_1,x_2);
167 A3(2,3) = dF2dbeta_2(S_2,MU,x_2);
168 A3(2,4) = dF2dbeta_3(S_3,MU,x_2,x_3);
169
170 A3(3,1) = dF3dx3(S_1,S_2,S_3,B1,B2,B3,MU,x_1,x_2,x_3);
171 A3(3,2) = dF3dbeta_1(S_1,MU,x_1,x_3);
172 A3(3,3) = dF3dbeta_2(S_2,MU,x_2,x_3);
173 A3(3,4) = dF3dbeta_3(S_3,MU,x_3);
174
175 A3(4,1) = dFrdx3(B1,B2,B3,MU,x_1,x_2,x_3);
176 A3(4,2) = dFrdbeta_1(x_1,x_2,x_3);
177 A3(4,3) = dFrdbeta_2(x_1,x_2,x_3);
178 A3(4,4) = dFrdbeta_3(x_1,x_2,x_3);
179
180 r1 = min(abs(eigs(A1)));
181 r2 = min(abs(eigs(A2)));
182 r3 = min(abs(eigs(A3)));
183
184 FF1 = @(x1,x2,x3,beta_1,beta_2,beta_3,mu,S1,S2,S3)   x1*(  (a1*f1(x1,S1)/mu +...
185  sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 +...
186   (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
187    (P11*f1(x1,S1)/mu-M11)*beta_1 - (P12*f2(x2,S2)/mu-M12)*beta_2 -...
188     (P13*f3(x3,S3)/mu-M13)*beta_3 + M1r*mu;
189 FF2 = @(x1,x2,x3,beta_1,beta_2,beta_3,mu,S1,S2,S3)   x2*(  (a1*f1(x1,S1)/mu +...
190  sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 +...
191   (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
192    (P21*f1(x1,S1)/mu-M21)*beta_1 - (P22*f2(x2,S2)/mu-M22)*beta_2 -...
193     (P23*f3(x3,S3)/mu-M23)*beta_3 + M2r*mu;
194 FF3 = @(x1,x2,x3,beta_1,beta_2,beta_3,mu,S1,S2,S3)   x3*(  (a1*f1(x1,S1)/mu +...
195  sigma_1-b1)*beta_1 + (a2*f2(x2,S2)/mu+sigma_2-b2)*beta_2 +...
196   (a3*f3(x3,S3)/mu+sigma_3-b3)*beta_3 + (sigma_r-br)*mu  ) -...
```

```
    (P31*f1(x1,S1)/mu-M31)*beta_1 - (P32*f2(x2,S2)/mu-M32)*beta_2 -...
     (P33*f3(x3,S3)/mu-M33)*beta_3 + M3r*mu;

FFr = @(x1,x2,x3,beta_1,beta_2,beta_3,mu) beta_1/g1(x1,x2,x3) +...
 beta_2/g2(x1,x2,x3) + beta_3/g3(x1,x2,x3) + mu/gr(x1,x2,x3) - 1;


y1 =  FF1(x_1,x_2,x_3,B1,B2,B3,MU,S_1,S_2,S_3);

y2 =  FF2(x_1,x_2,x_3,B1,B2,B3,MU,S_1,S_2,S_3);

y3 =  FF3(x_1,x_2,x_3,B1,B2,B3,MU,S_1,S_2,S_3);

y4 =  FFr(x_1,x_2,x_3,B1,B2,B3,MU);

y5 = r1;
y6 = r2;
y7 = r3;

z = [y1;y2;y3;y4;y5;y6;y7];
```

visualH.m

```
function [H,k_s,ro_s,sigma_s,x1_s,x2_s,x3_s] =...
 visualH(n,rndmz,e,Tex,H_in,X1_RANGE,X2_RANGE,X3_RANGE,visu)

if rndmz == 'Y'

disp('Vector of randomization:')

rad = [ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ran(e) , ...
ran(e) , ran(e) , ran(e) , ran(e), ran(e) , ran(e) , ran(e) , ran(e)] %15 values

disp('Original constants:')

k_s = [0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5] % 9 values

ro_s = [1,1,1]

sigma_s = [1,1,1,1]

disp('Perturbed values:')

k_s = [0.25,0.5+rad(1)/2,0.5+rad(2)/2,0.5+rad(3)/2,0.5+rad(4)/2,...
0.5+rad(5)/2,0.5+rad(6),0.5+rad(7),0.5+rad(8)]

ro_s = [1+rad(9),1+rad(10),1+rad(11)]

sigma_s = [1+rad(12),1+rad(13),1+rad(14),1+rad(15)]

else
if rndmz == 'N'

disp('Using fixed constants')

k_s = [0.2500    0.4881    0.5023    0.4863    0.4998...
    0.4977    0.5234    0.5089    0.5090]

ro_s = [0.9711    1.0404    1.0225]

sigma_s = [0.9992    1.0311    0.9748    1.0167]

else
if rndmz == 'S'

disp('Using original (symmetric) constants:')
```

```
44
45 k_s = [0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5] % 9 values
46
47 ro_s = [1,1,1]
48
49 sigma_s = [1,1,1,1]
50 end
51 end
52 end
53 S1_c = 1;
54 S2_c = 1;
55 S3_c = 1;
56 Substr_con = [1,1,1];
57 ro_1 = ro_s(1);
58 ro_2 = ro_s(2);
59 ro_3 = ro_s(3);
60
61 count = 0
62
63 if nargin == 4
64
65 x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
66 x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
67 x3_range = [0.01,min(S3_c/k_s(4),1/ro_s(3))]
68
69 a_x1 = x1_range(1);
70 b_x1 = x1_range(2);
71 a_x2 = x2_range(1);
72 b_x2 = x2_range(2);
73 a_x3 = x3_range(1);
74 b_x3 = x3_range(2);
75
76 x1_s = linspace(a_x1,b_x1,n(1));
77 x2_s = linspace(a_x2,b_x2,n(2));
78 x3_s  = linspace(a_x3,b_x3,n(3));
79
80 H = zeros(n(1),n(2),n(3));
81
82 opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3),Tex);
83
84 f = waitbar(0,'please wait..');
85
86 tot = n(1)*n(2)*n(3)
87
88 t_start = tic;
89
90 for i=1:n(1)
91 for j=1:n(2)
92 for k=1:n(3)
93 if ro_1*x1_s(i)+ro_2*x2_s(j)+ro_3*x3_s(k) > 1
94 H(i,j,k)=0;
95 count=count+1
96 f = waitbar(count/tot,f);
97 else
98 H(i,j,k) = opt_M([x1_s(i),x2_s(j),x3_s(k)]);
99 count = count + 1
100 f = waitbar(count/tot,f);
101 end
102 end
103 end
104 end
105
106 time = round(toc(t_start)/60,3)
107 close(f);
108
109 end
110
111 if nargin == 5
```

```
112
113 x1_range = [0.01,min(S1_c/k_s(2),1/ro_s(1))]
114 x2_range = [0.01,min(S2_c/k_s(3),1/ro_s(2))]
115 x3_range = [0.01,min(S3_c/k_s(4),1/ro_s(3))]
116
117 a_x1 = x1_range(1);
118 b_x1 = x1_range(2);
119 a_x2 = x2_range(1);
120 b_x2 = x2_range(2);
121 a_x3 = x3_range(1);
122 b_x3 = x3_range(2);
123
124 x1_s = linspace(a_x1,b_x1,n(1));
125 x2_s = linspace(a_x2,b_x2,n(2));
126 x3_s  = linspace(a_x3,b_x3,n(3));
127
128 figure
129 for i = 1:n(3)
130 subplot(floor(power(n(3),1/2)),floor(power(n(3),1/2)),i)
131 surf(x1_s,x2_s,-H_in(:,:,i))
132 height = [0,-min(H_in(:))];
133 axs = [x1_s(1) x1_s(end) x2_s(1) x2_s(end) height];
134 axis(axs);
135 az = 0;
136 el = 90;
137 view(az, el);
138 caxis([0 -min(H_in(:))])
139 colorbar
140 xlabel('x1');
141 ylabel('x2');
142 title(['Plot of the H Function in the x1-x2-x3 Space, x3 = ' num2str(x3_s(i))]);
143 end
144
145 end
146
147 if nargin == 8
148
149 x1_range = X1_RANGE
150 x2_range = X2_RANGE
151 x3_range = X3_RANGE
152
153 a_x1 = x1_range(1);
154 b_x1 = x1_range(2);
155 a_x2 = x2_range(1);
156 b_x2 = x2_range(2);
157 a_x3 = x3_range(1);
158 b_x3 = x3_range(2);
159
160 x1_s = linspace(a_x1,b_x1,n(1));
161 x2_s = linspace(a_x2,b_x2,n(2));
162 x3_s  = linspace(a_x3,b_x3,n(3));
163
164 H = zeros(n,n,n);
165
166 opt_M = @(X) fixed_pt(Substr_con,X(1),X(2),X(3),Tex);
167
168 f = waitbar(0,'please wait..');
169
170 tot = n(1)*n(2)*n(3)
171
172 t_start = tic;
173
174 for i=1:n
175 for j=1:n
176 for k=1:n
177 %t_start2 = tic;
178 H(i,j,k) = opt_M([x1_s(i),x2_s(j),x3_s(k)]);
179 count = count + 1
```

```
f = waitbar(count/tot,f);
%time_iteration = round(toc(t_start2)/60,3)
%time = round(toc(t_start)/60,3)
%time/count
end
end
end

time = round(toc(t_start)/60,3)
close(f);

end

if nargin == 8

figure
for i = 1:30
subplot(6,5,i)
surf(x1_s,x2_s,-H_in(:,:,i))
height = [0,0.082];
axs = [x1_s(1) x1_s(end) x2_s(1) x2_s(end) height];
axis(axs);
az = 0;
el = 90;
view(az, el);
caxis([0 0.082])
colorbar
end
end

end
```

cal_fixed_alphas.m

```
function [x1,x2,x3,mu,al1,al2,al3,e1,e2,e3,r] = cal_fixed_alphas(alpha1,alpha2,alpha3);

global a1 a2 a3 b1 b2 b3 br M11 M12 M13 M1r M21 M22 M23 M2r M31 M32 M33 M3r...
 P11 P12 P13 P21 P22 P23 P31 P32 P33 ...
ro_1 ro_2 ro_3 kcat k1_f k2_f k3_f k1_g k2_g k3_g kr_g...
 sigma_1 sigma_2 sigma_3 sigma_r k_s ro_s sigma_s Enz_con

IC = [0.3;0.3;0.3;0.3;0.3;0.3;0.3;0.3];
som = [ro_s sigma_s] * IC(1:7);
IC = IC/som;
[ro_s sigma_s] * IC(1:7);

options = odeset('AbsTol',1e-9,'RelTol',1e-9);

alpha_r = 1 - alpha1 - alpha2 - alpha3;

if alpha_r < 0
x1=0;
x2=0;
x3=0;
mu=0;
e1=0;
e2=0;
e3=0;
r=0;
else
rhs = @(t,y) Substrate_Switch_System_Dummy(t,y,[alpha1 alpha2 alpha3 alpha_r]);
[~,y] = ode15s(rhs,[0,1000],IC,options);

x1 = y(end,1);
x2 = y(end,2);
x3 = y(end,3);
```

```
e1 = y(end,4);
e2 = y(end,5);
e3 = y(end,6);
r  = y(end,7);
mu = -y(end,8);
end

al1 = alpha1;
al2 = alpha2;
al3 = alpha3;

end
```

DummyFunction1.m

```
function out4 = DummyFunction1(alpha1,alpha2,alpha3)
[~,~,~,out4,~,~,~,~,~,~,~] = cal_fixed_alphas(alpha1,alpha2,alpha3);
end
```

run_script.m

```

global k1_g k2_g k3_g S1_c S2_c S3_c T kcat

DummyFunction2 = @(X) DummyFunction1(X(1),X(2),X(3));

options = optimset('Display','iter','PlotFcns',@optimplotfval,'TolFun',0.00001,'maxiter',100);

y = [0.3,0.3,0.3];

%[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

N = 100;

sub1 = linspace(0.1,5,N);

zopts = zeros(15,N);

tic

f = waitbar(0,'please wait..');


for i = 1:N

kcat = sub1(i);

f = waitbar(i/N,f);

[x,fval,exitflag_fmin,output] = fminsearch(DummyFunction2,y,options)

[x1,x2,x3,mu,al1,al2,al3,e1,e2,e3,r] = cal_fixed_alphas(x(1),x(2),x(3))

zopts(1,i) = x1;
zopts(2,i) = x2;
zopts(3,i) = x3;
zopts(4,i) = e1;
zopts(5,i) = e2;
zopts(6,i) = e3;
zopts(7,i) = r;

zopts(8,i) = al1;
zopts(9,i) = al2;
zopts(10,i) = al3;
```

```
44 zopts(11,i) = al1*k1_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
45 zopts(12,i) = al2*k2_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
46 zopts(13,i) = al3*k3_g*x1*x2*x3/(1+x1+x2+x3+x1*x2+x1*x3+x2*x3+x1*x2*x3);
47 zopts(14,i) = -mu;
48
49 zopts(15,i) = sub1(i);
50
51 end
52
53 close(f);
54
55 toc
```

Bibliography

[1] Bob Planqué, Frank Bruggeman *Metabolism And Growth Rate In Microbiology* 2018.

[2] M. Schaechter, J. L. Ingraham, F. C. Neidhardt *Microbe* 2006. ASM Press.

[3] B. Palsson *Systems Biology, Properties Of Reconstructed Networks* 2006. Cambridge University Press.

[4] H. M. Sauro *Enzyme Kinetics for Systems Biology* 2009. Ambrosius Publishing.

[5] S. Klumpp, T. Hwa *Bacterial growth: global effects on gene expression, growth feedback and proteome partition* 2014. Elsevier.

[6] Molenaar, D., van Berlo, R., de Ridder, D., Teusink, B. *Shifts in growth strategies reflect tradeoffs in cellular economics* 2009. Molecular Systems Biology.

[7] Wortel, M. T., Peters, H., Hulshof, J., Teusink, B. *Metabolic states with maximal specific rate carry flux through an elementary flux mode* 2014.

[8] Gagneur, J., Klamt, S. *BMC Bioinformatics* 2004. The FEBS Journal.

[9] Weiße, A. Y., Oyarzún, D. A., Danos, V., Swain, P. S. *Mechanistic links between cellular trade-offs, gene expression, and growth* 2015. Proceedings of the National Academy of Sciences of the United States of America.

[10] Robert Planqué, Josephus Hulshof, Bas Teusink, Johan Hendriks, Frank J. Bruggerman *Maintaining maximal metabolic flux by gene expression control* 2016.

[11] A. Cornish-Bowden *Fundamentals of Enzyme Kinetics, 4th edition*. 2004. Wiley-Blackwell.

[12] van der Vlag J, van Dam K, Postma PW. Quanti cation of the regulation of glycerol and maltose metabolism by IIAGlc of the phosphoenolpyruvate-dependent glucose phosphotransferase system in Salmonel la typhimurium. J Bact. 1994;176(12):3518–3526.

[13] Jensen PR, Michelsen O, Westerho HV. Experimental determination of control by the H+-ATPase in Escherichia coli. J Bioenerg Biomem. 1995;27:543–554.

[14] Andersen HW, Pedersen MB, Hammer K, Jensen PR. Lactate dehydrogenase has no control on lactate production but has a strong negative control on formate production in Lactococcus lactis. FEBS Journal. 2001;268(24):6379–6389.

[15] Solem C, Koebmann BJ, Jensen PR. Glyceraldehyde-3-Phosphate Dehydrogenase Has No Control over Glycolytic Flux in Lactococcus lactis MG1363. J Bact. 2003;185(5):1564–1571.

[16] Koebmann B, Solem C, Jensen PR. Control analysis as a tool to understand the formation of the las operon in Lactococcus lactis. FEBS Journal. 2005;272(9):2292–2303.

[17] Keren L, Hausser J, Lotan-Pompan M, Slutskin IV, Alisar H, Kaminski S, et al. Massively parallel interrogation of the e ects of gene expression fevels on tness. Cell. 2016;166(5):1282–1294.e18.