UNIVERSITÀ DEGLI STUDI FIRENZE

Scuola di Scienze
Matematiche Fisiche
e Naturali

Corso di Laurea Magistrale in
Biologia Molecolare e Applicata

Realization of a dynamic genome-scale metabolic model for bacterial growth on complex medium and its application to study the production of recombinant proteins

Realizzazione di un modello metabolico dinamico genome-scale per la crescita batterica su terreno complesso e sua applicazione per studiare la produzione di proteine ricombinanti

Supervisor
Dr. Marco Fondi

Co-supervisor
Dr. Elena Perrin

Candidate
Tania Alonso Vásquez

Anno Accademico 2019/2020

Abstract

Genome-scale metabolic models give a great description of the organism at a gene-protein-reaction associations level, and their main objective is to predict metabolic fluxes of a certain organism. This can be done from a steady-state point of view, through a Flux Balance Analysis (FBA), or from a time-varying perspective, through the use of a Dynamic Flux Balance Analysis (DFBA).

In 2014, a tool named DFBAlab was developed by Gómez, Höffner and Barton. This MATLAB-based code provides a structured model of a biochemical process, where the environmental conditions are taken into account to predict the microorganism's dependency on the substrate concentrations. The uniqueness of this code relies on the using of lexicographic optimization and linear programming (LP) feasibility problem, which means that in addition to prioritize the objective functions (biomass, amino acids uptake, etc.), it sets the feasibility cost vector as the top priority objective.

The organism of interest for this thesis was the Antarctic bacterium *Pseudoalteromonas haloplanktis* TAC125 (*Ph*TAC125), a strain of a gram-negative marine γ-proteobacteria. Such strains thrive permanently in sea water at about -2°C to +4°C but are also anticipated to endure long term frozen conditions when entrapped in the winter ice pack. *Ph*TAC125 translational efficiency even in cold condition and the consequently rapid growth justifies its increasing use in biotechnological applications, such as high quality production of recombinant eukaryotic proteins.

The aim of this thesis was to build a dynamic genome-scale metabolic model, using DFBAlab algorithm, that could describe *Ph*TAC125's growth on complex medium, particularly when having 19 amino acids as carbon sources, and its further application to study the production of recombinant proteins.

In order to do this, recently obtained metabolomics data were used, and a broad fitting method was developed to obtain the kinetic constants V_{max} and K_m, using several fitting algorithms and

then comparing them through the use of Akaike's Information Criterion. The kinetic constants were used to implement the DFBAlab code and the results were qualitatively similar to the experimental observations, *i.e.* the amino acids uptake order was the same *in vitro* than *in silico*. It is important to say that the model was not sensible to six amino acids.

On the other hand, the dynamic model was used to study the production of recombinant protein CDKL5. Given that the medium was different, new kinetic constants were obtained and DFBAlab was implemented. The dynamic model qualitatively described the nutrients uptake and was able to predict *Ph*TAC125's growth difference when producing the recombinant protein CDKL5 versus when it is not.

In conclusion, the realization of the dynamic genome-scale metabolic model was succesfully performed, it is able to describe the metabolic dynamics of *Ph*TAC125, qualitatively matching previous experimental data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 *Pseudoalteromonas haloplanktis* TAC 125

The marine environment occupies three quarters of the Earth's surface, which means that it is home to a significant amount of biodiversity, hosting more living organisms, especially microorganisms, than any other environment [1]. More than 70% of the marine surface experiments yearly temperatures below 15°C [2], in fact polar regions account for another 15% of the Earth's surface, possessing unusual microbiotopes such as porous rocks in Antarctic dry valleys hosting microbial communities surviving at -60°C. These circumstances make it necessary for life to develop a remarkable adaptation to cold conditions [3].

This led to a deeper study of marine psychrophilic microorganisms (*i.e.* microorganisms growing well at temperatures around the freezing point of water), whose ability to survive and proliferate at low temperatures implies that they have overcome key barriers inherent to permanently cold environments, such as reduced enzyme activity, decreased membrane fluidity, altered transport of nutrients and waste products, decreased rates of transcription, translation and cell division, protein cold-denaturation, to name a few [4].

In mesophilic organisms, the exposure to sudden temperature changes, both upshifts and downshifts, induces the over-expession of heat or cold-shock proteins, involved in transcription, translation, protein folding and the regulation of membrane fluidity [5], however, this cold-shock response is different in psychrophilic microorganisms since there is a lack of repression of house-keeping protein synthesis and the presence of cold-acclimation proteins (Caps). Many of the cold-shock proteins observed in mesophiles act as Caps in psychrophiles, being constitutively rather than transiently expressed at low temperatures [4].

Amongst bacterial genera that can be isolated from the marine environment, one of the most frequent is *Pseudoalteromonas*, a highly diffuse obligatory marine bacteria that are a subgroup of Gram-negative γ-proteobacteria [1]. A typical representative is the bacterium *Pseudoalteromonas haloplanktis*, and the strain TAC125 has been isolated from sea water sampled along the Antarctic ice-shell. Such strains thrive permanently in sea water at about -2°C to +4°C but are also anticipated to endure long term frozen conditions when entrapped in the winter ice pack [6].

P. haloplanktis TAC125 genome is made of two chromosomes, the replication origin of the first chromosome (chrI) is found in a region that is highly conserved in γ-proteobacteria, unlike the second chromosome (chrII) whose pattern is likely to be caused by unidirectional replication [1]. The genome of *P. haloplanktis* TAC125 contains 19 genes presumably coding for known RNA binding proteins or RNA chaperones. However, this bacterium has several features that were not expected, like the prominent absence of a RNA/nucleoid-associated cold-shock gene ubiquitous in γ-proteobacteria, *hns*, which could mean that its activity was not enough to promote growth at low temperatures. In addition, it was found one specific region in chrI coding for several calcium-dependent proteins, as well as a specific gene in chrII that may regulate cell volume and resistance to cold conditions. These findings are important since calcium is known to be involved in cold adaptation and formation of exopolysaccharides in bacteria [1].

Psychrophilies like *P. haloplanktis* produce cold-adapted enzymes that have high specific activities at low temperatures, often up to an order of magnitude higher than those observed for their mesophilic counterparts, such as *Erwinia chrysanthemi* (Fig. 1.1). This enzymatic activity is thought to adapt structural flexibility and kinetic properties to cope with the freezing effect of the cold habitats [7]. One example could be the resistance to protein aging features involving asparagine cyclization and deamidation, and the high number of rRNA and tRNA genes (106), which might explain its translational efficiency even in cold condition and the consequently rapid growth [1] [2]. This latter observation justifies an increasing use of *P. haloplanktis* in biotechnological applications, e.g. high quality production of recombinant eukaryotic proteins [1] such as cyclin-dependent kinase-like (CDKL), present in various eukaryotas including humans. These kinases have an essential role in signaling and development processes, hence they are relevant for studying neurological disorders [8].

Another important characteristic of cold environments is the solubility of gases, especially oxygen, which is increased at low temperatures while radicals are stabilized. As a consequence, psychrophiles are exposed to higher concentrations of reactive oxygen species (ROS) [3]. However, *P. haloplanktis* TAC125 is remarkably well adapted to protection against ROS under these conditions, a feature that could be very useful for expression of foreign proteins in the cold. For example, it lacks a series of activities that result in reactive oxygen species production (*i.e.* deletion of ROS-producing metabolic pathways) and contemporaneously the direct use of dioxygen through the presence of dioxygenases incorporating it into oxidized macromolecules [2] [3].

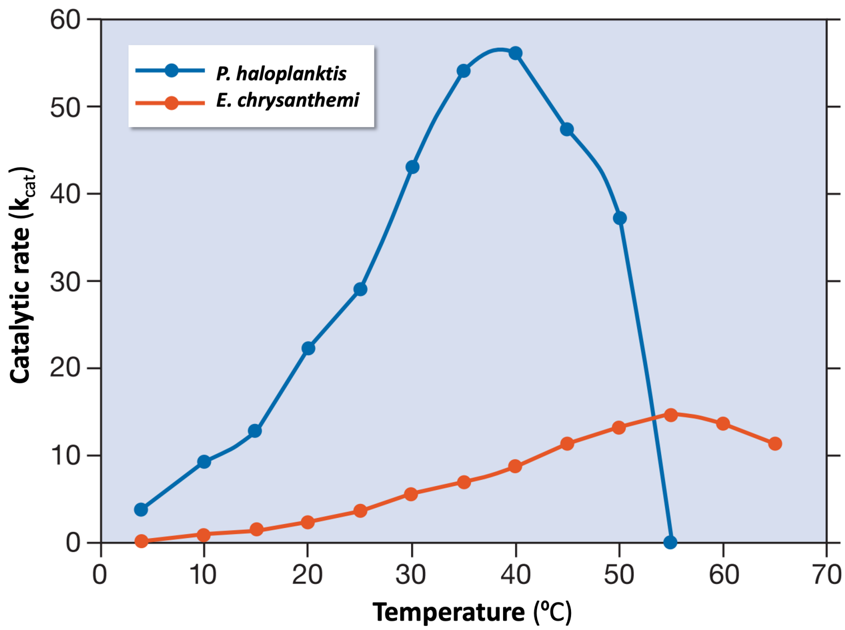

Figure 1.1: Thermodependence of enzymatic activity for the cold-adapted cellulase from *Pseudoalteromonas haloplanktis* (blue) and its mesophilic homologue *Erwinia chrysanthemi* (red). Adapted from [4].

In 2010, Piette et al. [3] performed a study to evaluate the optimal growth temperature of *P. haloplanktis* TAC125 and compared it with that of the mesophile *Escherichia coli*. The former was grown in a marine broth while the latter was grown in LB broth. Exponential growth was measured through the doubling time of the bacterial population, being 4°C the optimal growth temperature for TAC125, while *E. coli* failed to grow exponentially below 10°C (Fig. 1.2), the generation time of *P. haloplanktis* is moderately increased when the culture temperature is decreased. This means that reaction rates are less affected by a decrease in temperature as compared with mesophilic enzymes.

In other words, high enzyme-catalysed reaction rates maintain metabolic fluxes and cellular functions at low temperatures (as shown in figure 1.1), whereas the weak temperature dependence of enzyme activity counteracts the effect of cold temperatures on biochemical reaction rates.

In order to work with *P. haloplanktis* TAC125, it was important to understand how it grew.

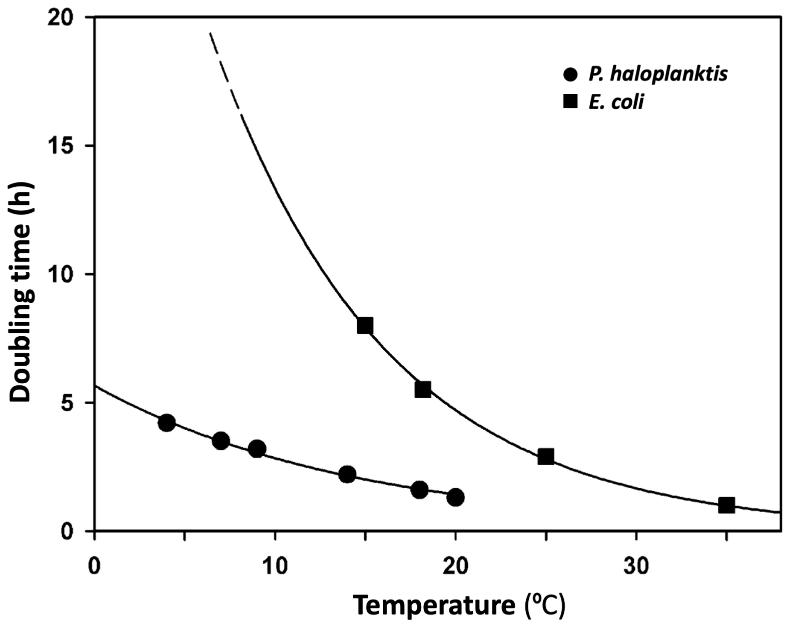

Figure 1.2: Temperature–dependence of growth for *P. haloplanktis* and for the mesophile *E. coli*. Solid lines are best-fit of the data to a single-exponential equation. The dashed line indicate *E. coli* fails to grow exponentially below approx. 10°C. Figure from [3].

Marine bacteria live in a medium generally unbalanced in terms of carbon, nitrogen and phosphorous supply in sulfur sources, and since TAC125 is adapted to fast growth, this suggests that it regularly encounters a fairly rich medium, probably due to the formation of a biofilm.

One interesting feature of *P. haloplanktis* is its lack of cAMP-CAP complex that regulates carbon availability in related organisms such as vibrios and *Shewanella*, plus it does not possess a phosphoenolpyruvate-dependent phosphotransferase system for the transport and first metabolic step of carbohydrate degradation, explaining its lack of growth on glucose or related sugars [1]. Hence, this bacterium had to be adapted to grow on rich medium, as proved by Wilmes et al. in 2010 [9], where the growth was compared when cultivated on soy peptone versus casamino acids, where *P. haloplanktis* grew exponentially with a constant growth rate. A mineral salt-based fed-batch medium with casamino acids as a substrate (which had a different concentration of single amino acids), was a suitable substrate for a high cell density cultivation of *P. haloplanktis* cells. The amino acid analysis revealed glutamate, serine, threonine, aspartate and, if applicable, leucine as potential limiting amino acids. But only glutamate turned out to be the preferred amino acid with the highest influence on growth and thus is the most promising candidate as a highly concentrated carbon and nitrogen source for a fed-batch feeding solution [9]. According to these data, there is a hierarchical use of carbon sources. Recent works have shown that *P. haloplanktis* can grow as well in a L-glutamate, D-gluconate medium (GG) which allowed the bacterium not only to grow at subzero temperatures, but the production of a recombinant protein. This result is being used to further investigate some basic science issues and will be instrumental in the production of difficult proteins [10].

1.2 Genome-scale metabolic modelling

Metabolism can be defined as the complete set of chemical reactions that occur in living organisms in order to maintain life. Enzymes are the main players in this process as they are responsible for catalyzing the chemical reactions. The enzyme–reaction relationships can be used for the reconstruction of a network of reactions, which leads to a model of metabolism [11]. Such a model must be predictive of events at the molecular scale and capable of explaining the high-level behavior of the cell as a whole [12].

Genome-scale metabolic models (GEMs) computationally describe gene-protein-reaction (GPR) associations for entire metabolic genes in an organism, and can be simulated to predict metabolic fluxes for various systems-level metabolic studies [13]. The first step is to build the metabolic networks, which are primarily reconstructed from the information that is present in their genome and in the literature, involving steps such as functional annotation of the genome, assignment of gene and reaction localization, determination of the biomass composition, estimation of energy requirements, definition of model constraints, identification of the associated reactions and determination of their stoichiometry [14] [11] [15]. Establishing a set of the biochemical reactions that constitute a reaction network in the target organism culminates in a data-base of chemical equations. Reactions are then organized into pathways, pathways into sectors (such as amino acid synthesis), and ultimately into genome-scale networks (Fig. 1.3); thus, network reconstructions represent an organized process for genome-scale assembly of disparate information about a target organism [15].

The next step is to mathematically represent the network reconstruction. This conversion translates a reconstructed network into a chemically accurate mathematical format that becomes the basis for a genome-scale model, *i.e.* the mathematical representation of metabolic reactions [15]. Hence, it is employed a stoichiometric matrix (S matrix) to represent all the coefficients in metabolic reactions where the ijth element represents the stoichiometric coefficient of the ith metabolite in the jth reaction in the GEM (Fig. 1.4a). If the coefficient is positive, the metabolite is produced; otherwise, it is consumed [16].

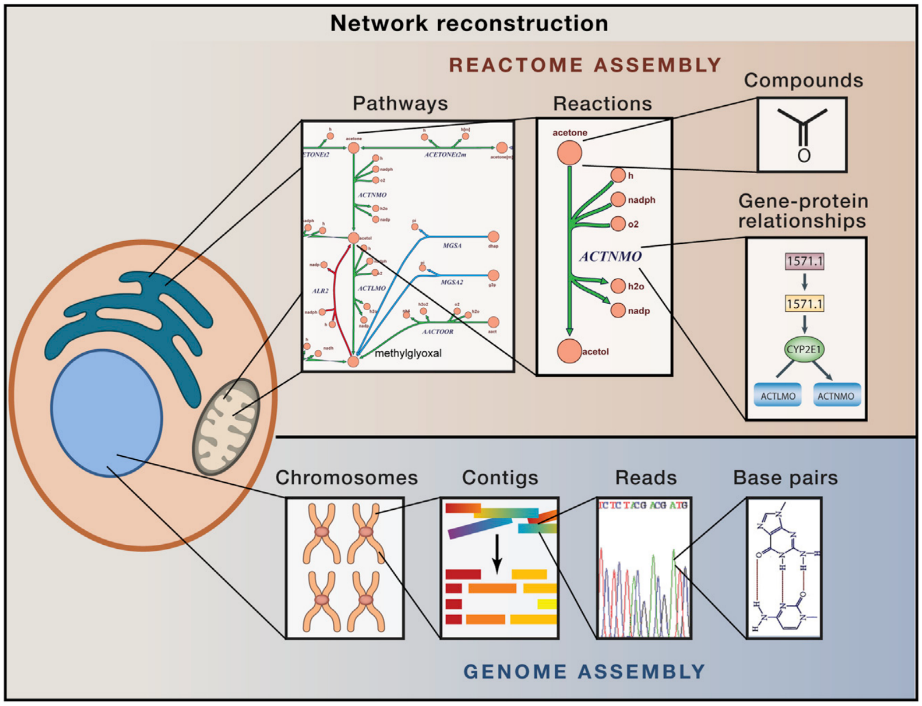

Figure 1.3: Information required for the network reconstruction process. Figure from [15].

The matrix of stoichiometries imposes flux balance constraints on the network, ensuring that the total amount of any compound being produced must be equal to the total amount being consumed at steady state. The flux vector, a mathematical object, is a list of all such flux values for a single point in the space, and represents a "state" of the network that is directly related to the physiological function that the network produces [15], these vectors are being constrained by upper an lower bounds that will allow the model to be representative and have a solution space (Fig. 1.4b). The challenges in constraint-based modelling lie in identifying and imposing the necessary and dominant constraints to define this solution space, as well as in probing the solution space in a manner such that physiologically relevant fluxes or phenotypes are determined [17].

Finally, there can be different approaches, those that predict gene expression and those that use gene expression data from a particular state to predict other phenotypes. The important feature is that GEMs can predict gene expression with no previous input expression measurements: they can compute protein abundances that are required to (optimally) achieve integrated physiological functions. Since they are based on fundamental constraints and optimality principles, GEMs can therefore be used to predict optimal expression and regulatory states [18].

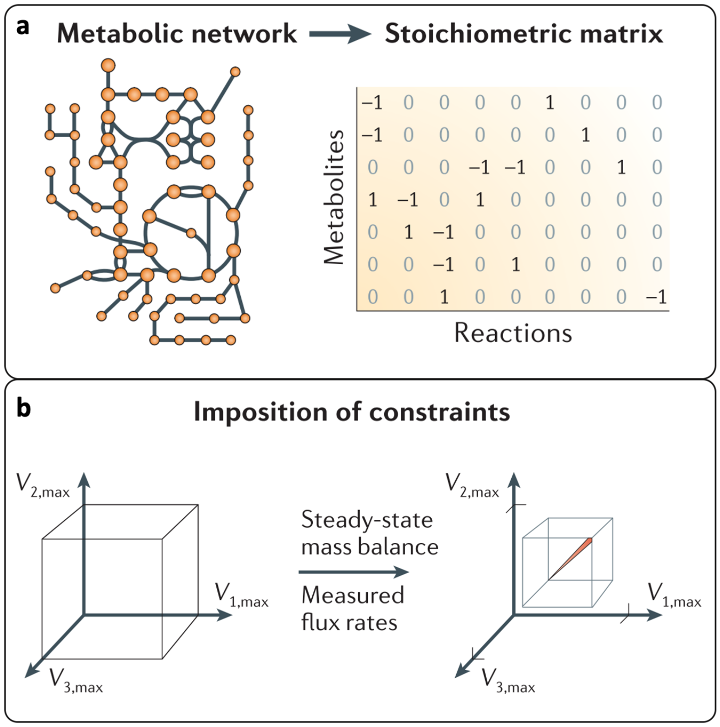

Figure 1.4: **a.** The stoichiometric matrix is constructed using the information contained in the metabolic network. **b.** Imposition of constraints through the use of upper and lower bounds for the flux vectors, the box is the solution space for the GEM. Adapted from [17].

In 1999, a metabolic model of *Haemophilus influenza* became the first genome-scale metabolic reconstruction to be published [19], and in the decades since, the field of genome-scale metabolic network analysis has expanded so rapidly that it led to an exponential growth on the amount of GEMs from 2009 to 2019 [14] [13]. In the figure 1.5 it can be seen a phylogenetic tree of all GEMs reconstructed until 2019.

The GEMs for model organisms that have high scientific, industrial, and/or medical values have been updated several times since their initial reconstruction as more relevant biological information became available over the years. This is possible thanks to the up-to-date experimental information on GPR associations and cell growth under various conditions (such as in gene knockouts or when different carbon sources are used) [13].

The genome-scale metabolic model that concerns this study is the one constructed by Fondi et al. in 2015 of *Pseudoalteromonas haloplanktis* TAC125 (iMF721), which was not only the first one reconstructed for this bacterium, but it was the first one reconstructed so far for an Antarctic microbial strain [20].

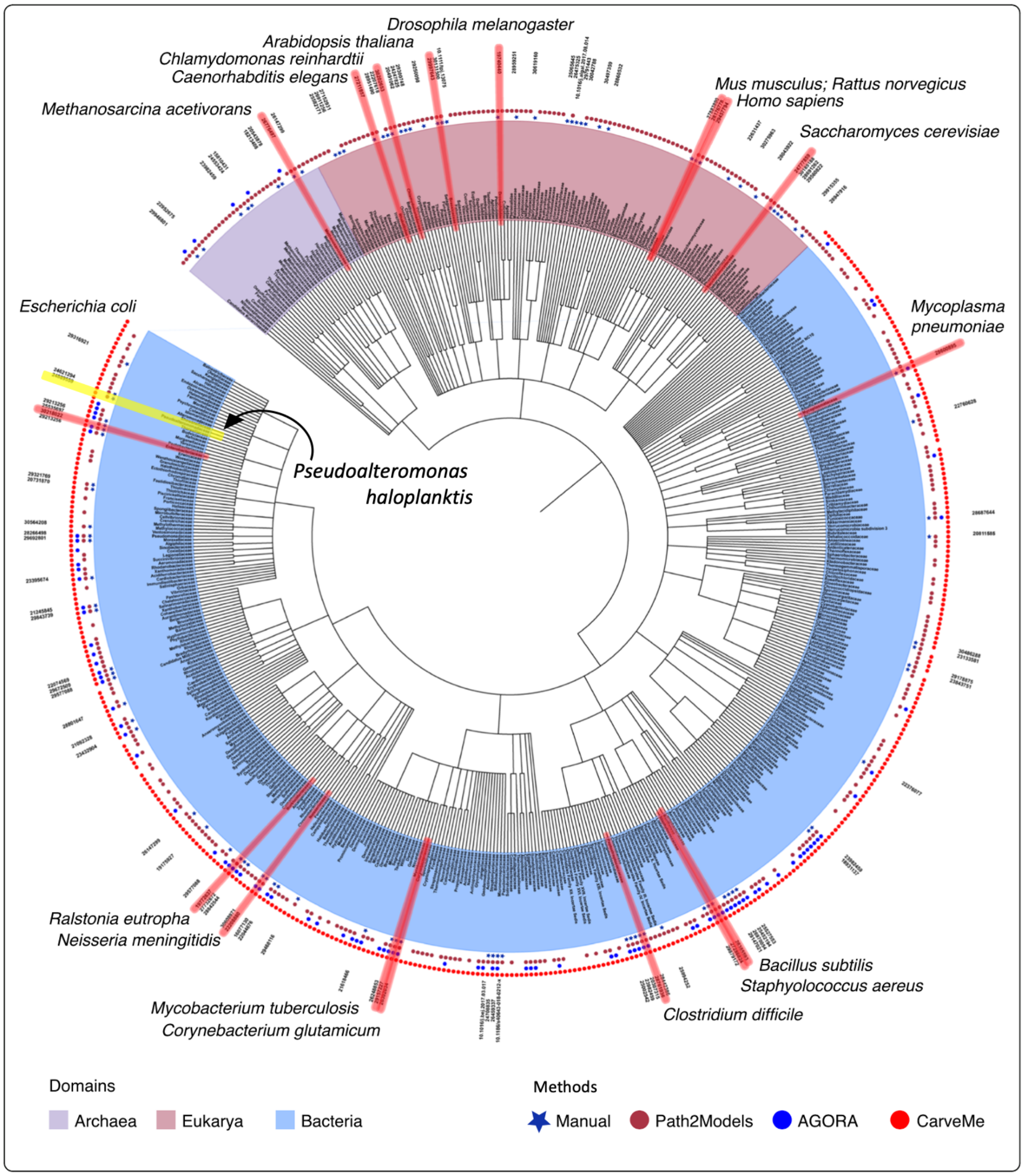

Figure 1.5: A phylogenetic tree of all GEMs reconstructed until 2019 at the family level. GEMs for 434, 40, and 117 taxonomic families of bacteria (light blue), archaea (light purple), and eukarya (pink), respectively. Organism names are labeled depending on the development methods used. *Pseudoalteromonas haloplanktis*, who belongs to Pseudoalteromonadaceae family, is indicated in yellow. Adapted from [13].

The integration of expression data with functional modelling of *Pseudoalteromonas haloplanktis* TAC125 metabolism presented in Fondi's study has allowed the identification of possible metabolic consequences derived from the re-modulation of gene expression in response to cold adaptation, results that support and extend previous observations suggesting that this bacterium depresses its general metabolism when grown at low temperature, in agreement with the reduced biomass produced at 4°C [6] [20]. Hence, this model allows the study of variations in cellular metabolic fluxes following a temperature downshift.

As said in the previous section, bacteria live in a wide range of environmental conditions that change over time. In this sense, when facing a nutritionally rich environment, bacteria first use the "preferred" compound(s) (presumably those allowing the fastest growth rate) and only later start metabolizing the other one(s) [21]. This metabolic switch due to preferences is characteristic of systems that optimize fitness [22]. It is usually observed that, as the bacterium changes from one carbon source to another, growth is temporary halted, while a new set of enzymes needed to metabolize alternative nutrients are synthesized, therefore, a small change in nutrient concentration due, for example, to compounds exhaustion, may sometimes induce a large change in the enzymatic composition of the bacterium. This was proven by Fondi et al. in 2016 [21], providing a scheme of *P. haloplanktis* TAC125 metabolic re-programming that revealed a number of nutrients switches in its metabolism when grown in a complex medium, consistent with the fact that this Antarctic organism is adapted to fast growth in a fairly rich (but probably inconstant and highly competitive) environment (plankton debris) [2]. This model highlighted the occurrence of such an adaption and the need for reprogramming a large set of reactions to maintain an efficient metabolic network, showing evident growth phases (diauxic or multi-auxic growth) [21] [23].

However, this scenario can be combined with the co-utilization of carbon sources. In 2020, Perrin et al. [24] studied this case, showing that *P. haloplanktis* can use both strategies simultaneously when multiple possible nutrients are provided in the same growth experiment. This conditions should always closely resemble the ones most found in nature. Hence, despite diauxie and co-utilization strategies have been usually thought as conflicting phenotypes, they can coexist in the same growth curve and give rise to a diversified ensemble of feeding strategies [24].

GEMs continue to play important roles in systems biology and metabolic engineering. Its current applications include: designing metabolic engineering strategies such as system-wide prediction of gene manipulation targets; biological discovery such as identification of gene functions and prediction of microbial community-wide metabolic activities; and identification of drug targets in microbial pathogens [25]. Advances in the reconstruction and use of GEMs are largely attributed to the greater availability of biological data and information, and to the establishment of automatic GEM reconstruction tools. Over time, they will become more powerful by incorporating additional biochemical information that will provide explanations of cellular processes beyond metabolism. Nevertheless, various biochemical properties, such as enzyme–substrate interactions, the structure of protein–protein complexes, and post-translational modification, still need to be considered further [13].

1.3 FBA and DFBA

Flux balance analysis (**FBA**) is the most basic and commonly used method for studying biochemical networks, in particular the genome-scale metabolic network reconstructions [26]. GEMs contain all of the known metabolic reactions in an organism and the genes that encode each enzyme, and FBA calculates the flow of metabolites through this metabolic network, thereby making it possible to predict the growth rate of an organism or the rate of production of a biotechnologically important metabolite [27].

FBA is a mathematical approach for analyzing the flow of metabolites through a metabolic network that relies on an assumption of steady-state growth and mass balance (all mass that enters the system must leave, in other words, the net sum of all the production and consumption rates of each internal metabolite within a cell is considered to be zero) [20] [15]. It is based on the stoichiometric metabolic matrix constructed for the model (Fig. 1.4a), however, they are typically underdetermined because the number of reactions in the model is usually larger than the number of metabolites. Therefore, in most cases, constraint-based analysis yields multiple feasible flux solutions [28]. To narrow down the space of feasible solutions, every reaction can also be given upper and lower bounds, which define the maximum and minimum allowable fluxes of the reactions. These balances and bounds define the space of allowable flux distributions of a system (1.4b), *i.e.* the rates at which every metabolite is consumed or produced by each reaction (e.g. nutrient utilization) and maximization of biomass production [27] [29].

In addition to the application of constraints, an *objective function* is usually defined for identifying biologically relevant flux solutions. The most commonly used objective function for FBA is the biomass objective function (BOF), which is to maximize the efficiency of biomass production, *i.e.* growth rate. This "biomass reaction" is basically a collection of all individual biomass constituents together with their fractional contributions to the overall cellular biomass, and energetic requirements for the biomass generation [28].

One of the most basic constraints imposed on genome-scale metabolic models is that of substrate, or nutrient, availability and its uptake rate. Metabolites enter and leave the systems through what are termed “exchange reactions” (*i.e.* active or passive transport mechanisms). These reactions define the extracellular nutritional environment and are either left “open” (to allow a substrate to enter the system at a specified rate) or “closed” (the substrate can only leave the system) [15]. All these can be mathematically described by the set of linear equations:

$$\frac{dX_i}{dt} = \sum_{j=1}^{M} S_{ij} v_j = 0, \quad \forall i \in N, \forall j \in M \tag{1.1}$$

Where X_i is the concentration of metabolite i, S_{ij} is the stoichiometric coefficient of the ith metabolite in the jth reaction, v_j is the flux of the jth reaction, N the entire set of metabolites and M the entire set of reactions. The upper and lower bounds of flux through each reaction act as further constraints and are expressed as:

$$l_b < v_j < u_b \tag{1.2}$$

where l_b and u_b are the lower and upper limits for reaction j, respectively [20]. Finally, in FBA, these equations are solved using linear programming to determine a feasible steady-state flux vector that optimizes a given objective function. Many computational linear programming algorithms exist, and they can very quickly identify optimal solutions to large systems of equations (e.g. COBRA Toolbox [30], a freely available Matlab toolbox) [27] [20].

In summary, the steps of the FBA are illustrated in figure 1.6. First, a genome-scale metabolic model must be reconstructed and the reactions set (Fig. 1.6 a). Then, the stoichiometric matrix must be defined, including the information of the biomass and exchange reactions (Fig. 1.6b), this matrix is multiplied by the flux vector, and since FBA relies on the assumption of steady state, the flux through each reaction has to be zero (given by Sv), which defines a system of linear equations (Eq. 1.1, Fig. 1.6c). As large models contain more reactions than metabolites, there is more than one possible solution to these equations. In order to predict the maximum growth rate, it requires defining an objective function $Z = c^T v$ (*i.e.* an inner product where c is a vector of weights indicating how much each reaction (v) contributes to the objective function). In practice, when only one reaction, such as biomass production, is desired for maximization or minimization, c is a vector of zeros with a value of 1 at the position of the reaction of interest (Fig. 1.6d). Lastly, linear programming (LP) is used to identify a flux distribution that maximizes or minimizes the objective function within the space of allowable fluxes (blue region) defined by the constraints imposed by the mass balance equations and reaction bounds, in figure 1.6e, the final solution is indicated as the point of the optimal flux v [27].

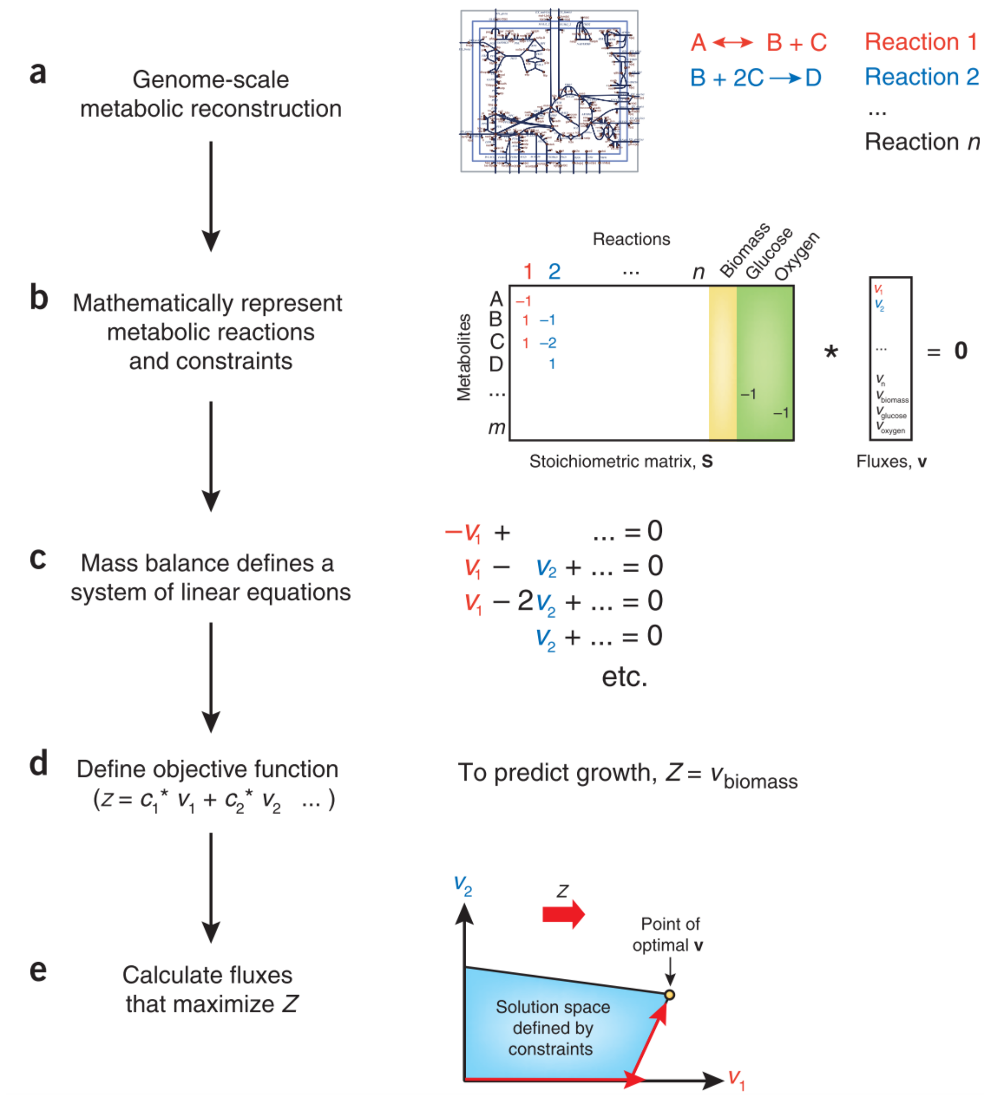

Figure 1.6: Flux balance analysis: **a.** GEM is reconstructed, an the reactions are set. **b.** The stoichiometric matrix is defined, the biomass reaction (yellow column) and exchange reactions (green columns) are incorporated. It is defined the fluxes as well. **c.** At the steady state, the sum of the fluxes is 0. **d.** The objective function is defined as a combination of all fluxes and their coefficients. **e.** Linear programming is used to identify solution space of allowable fluxes (blue region). Figure from [27].

FBA has been extended to dynamic FBA (**DFBA**), which is applicable to time-varying processes, such as batch or fed-batch cultures, and has significantly contributed to metabolic and cultural engineering applications [31]. DFBA allows partial recovery of the dynamic information lost under the conditions of the steady-state assumption (FBA), such as the metabolite concentrations [32], and since DFBA is applied to systems involving cell growth, the theoretical maximum production concentrations and yields can be estimated under conditions closer to the production process than FBA [31]. DFBA also permits the incorporation of kinetic expression when the kinetics are well characterized [32], which will allow the use of kinetic constants (V_{max} and K_m) in this thesis, that will be previously calculated.

There are three possible approaches: Dynamic Optimization Approach (DOA), which involves optimization over the entire time period of interest to obtain time profiles of fluxes and metabolite levels, by solving a nonlinear programming problem (NLP) [32] [33]. In the Static Optimization Approach (SOA), time is divided into discrete intervals and a new FBA problem is solved at time t_i after updating of the external conditions according to the FBA solution at time t_{i-1}, and performs the approximation of the dynamic response of a GEM to a changing environment [34], this optimization problem is solved using LP repeatedly during the course of the batch to obtain the flux distribution at a particular time instant [32]. Finally, the Direct Approach (DA) has been proposed by including the LP solver in the right-hand side evaluator for the ordinary differential equations (ODEs) [33]. Most of the published DFBA models use the SOA approach, which is relatively simple to implement and not as computationally demanding [35].

Several algorithms have been developed to improve dynamic FBA, such as an extension of DFBA by Succurro et al. in 2018 [34] and AdaptiveDFBA algorithm by Valverde et al. in 2019 [36]. However, the one that could address the needs of this thesis due to its lexicographic optimization implementation was DFBAlab by Gómez et al. [33].

1.4 How DFBAlab was constructed

The Dynamic Flux Balance Analysis Laboratory (DFBAlab), developed at the Massachisetts Institute of Technology by José A. Gómez, Kai Höffner and Paul I. Barton in 2014 [33], is a MATLAB-based code that provides a structured model of a biochemical process, where the environmental conditions are taken into account to predict the microorganism's dependency on the substrate concentrations.

DFBAlab provides a solution to two major difficulties in existing implementations: nonunique exchange fluxes in the solution vector of an LP and the LP becoming infeasible when evaluating the ODE right-hand side close to the boundary of feasibility. In order to do that, DFBAlab implements lexicographic optimization to obtain unique exchange fluxes and uses the LP feasibility problem to avoid obtaining infeasible LPs while running the simulation. It uses linear program solvers such as CPLEX [37], Gurobi [38] or MOSEK [39].

DFBAlab is mathematically defined as follows: let the vector X_0 be the one that contains the initial concentrations of metabolites and biomass in a culture (it would be the initial vector from equation 1.1), and if the study is evaluating more than one microorganism, it can be considered n_s as the number of microbial species in the culture.

$$\begin{aligned} x(t_0) &= x_0, \\ \dot{x}(t) &= f(t, h^1(x(t)), \ldots, h^{n_s}(x(t))), \quad \forall t \in (t_0, t_f], \end{aligned} \tag{1.3}$$

Where f is the change function (due to exchange fluxes, feed and discharge rates from the culture, mass transfer rates, etc.), whose rate can be obtained for each of the components of x. This function f is integrated to find the concentration profiles with respect to time, $x(t)$. Vector h^k is the one containing the exchange fluxes of species k, and is obtained by solving a linear program:

$$\begin{aligned} \max\ (c^k)^T v, \quad \forall v \in R^{n_r^k} \\ S^k v = 0, \\ v_{lb}^k(x(t)) \leq v \leq v_{ub}^k(x(t)), \end{aligned} \tag{1.4}$$

Where c^k is the cost vector that maximizes growth fluxes, v the flux distribution, n_r^k are the number of reactions in the metabolic network of species k, and v_{lb}^k, v_{lb}^k are lower and upper bounds as functions of the extracellular concentrations. Hence, the vector h^k then takes the solution of this linear program to find the values of the exchange fluxes, however the solution set v can be nonunique and it could not be clear which flux distribution should h^k take to carry-on with the integration [33]. To fix this problem, Höffner et al. [40] used lexicographic optimization, a strategy that enables obtaining unique exchange fluxes. This strategy requires defining an objective function for each exchange flux of interest and these have to be ordered in a priority list [41]. The highest priority objective is optimized first; then its optimum value is added as a constraint and the next objective in priority is optimized, and so on. The first objective usually is the maximization of biomass, however, this order must be provided by the user in the model description. By making all the exchange fluxes that appear in the right-hand side of equation 1.3 optimization objectives ordered by priority, unique exchange fluxes are obtained, these exchange fluxes change continuously with respect to time and the integrator is able to carry-on integration reliably [33].

Nonetheless, DFBA simulators tend to have problems with LP in equation 1.4 for it may become infeasible as time progresses. There are two situations where the LP may become infeasible:

1. The problem is truly infeasible, the solution fails and the integration should be terminated.

2. The problem is not infeasible but the LP becomes infeasible while the numerical integrator performs various operations to take a time step in equation 1.3.

An LP feasibility problem has two main characteristics: it is always feasible and its optimal objective function value is zero if and only if the original LP is feasible (it could be identify as infeasible). Since and LP feasibility problem can be constructed by adding some slack variables to the constraints, Gómez et al. [33] implemented the LP formulation in equation 1.4 as the following LP feasibility problem:

$$\begin{aligned} \min \sum_{i=1}^{n_q^k} S_{+i} + s_{-i}, \quad \forall v \in R^{n_r^k},\ \forall s_+, s_- \in R^{n_q^k} \\ S^k v + s_+ - s_- = 0, \\ v_{lb}^k(x(t)) \leq v \leq v_{ub}^k(x(t)), \end{aligned} \tag{1.5}$$

Where n_q^k are the number of metabolites in the metabolic network of species k, and S_i the ith row of S (stoichiometric matrix). When an LP is constructed in this form, a feasible solution is obtained by finding a v that be delimited by its lower and upper bounds, and then letting:

$$\begin{cases} S_i^k v < 0 \quad \text{then } s_{+i} = -S_i^k v \quad \text{and} \quad s_{-i} = 0 \\ S_i^k v > 0 \quad \text{then } s_{-i} = -S_i^k v \quad \text{and} \quad s_{+i} = 0 \end{cases} \tag{1.6}$$

DFBAlab uses the LP feasibility problem (Eq. 1.5) instead of the standard form (Eq. 1.4) to find the growth rates and exchange fluxes for each species in the culture. It sets the feasibility cost vector as the top priority objective in the lexicographic optimization scheme. Then, the second-priority linear program maximizes biomass and the subsequent lower-priority LPs obtain unique exchange fluxes. It is important to say that it is provided a penalty function that can be useful for optimization purposes. Only trajectories with penalty function value equal to zero (within some tolerance) are feasible [33].

Chapter 2

Motivation

Since the Antarctic strain *Pseudoalteromonas haloplanktis* TAC125 (*Ph*TAC125) is one of the most important model organisms of cold-adapted bacteria that is currently being exploited as a new alternative expression host for numerous biotechnological applications [20], the aim of this thesis was to build a dynamic genome-scale metabolic model, using DFBAlab algorithm, that could describe *Ph*TAC125's growth on complex medium, particularly when having 19 amino acids as carbon sources (cysteine was not included in the list because of difficulties in its unambiguous quantitation during the experiments due to its spontaneous oxidation [24]).

Furthermore, it was explored the model's application to study the production of recombinant proteins, such as CDKL5, which meant using a different experimental setup and a different simulated growth medium. The workflow implied the comparison of experimental and simulated data in order to evaluate the validity of the dynamic genome-scale metabolic model.

Chapter 3

Methods

3.1 Workflow

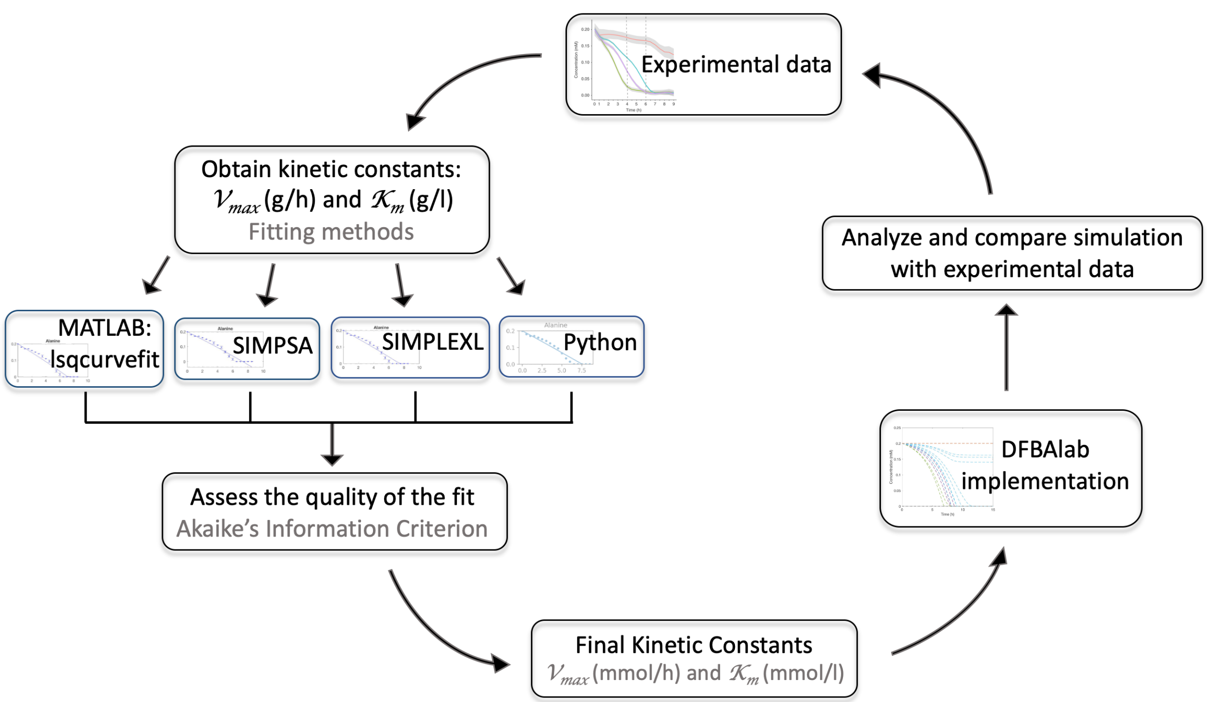

Figure 3.1: Scheme of the workflow followed in this thesis.

3.2 Softwares

- MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly. It includes the Live Editor for creating scripts that combine code, output, and formatted text in an executable notebook [42].
- The COnstraint-Based Reconstruction and Analysis Toolbox (COBRA Toolbox) is a MATLAB software suite for quantitative prediction of cellular and multicellular biochemical networks with constraint-based modelling. It implements a comprehensive collection of basic and advanced modelling methods, including reconstruction and model generation as well as biased and unbiased model-driven analysis methods, applicable to any biochemical system with prior mechanistic information [30].
- Dynamic Flux Balance Analysis laboratory (DFBAlab) is a MATLAB-based code that performs numerical integration of dynamic flux balance analysis (dFBA) systems. It provides efficient simulation of multi-culture of microbial species based on genome-scale metabolic network reconstructions for analysis, control and optimization of biochemical processes. As such, it generates dynamic predictions of substrate, biomass, and product concentrations for growth in batch or fed-batch cultures [33].
- Python is a programming language. It is used for many different applications such as an introductory programming language, however it's also used by professional software developers at places such as Google, NASA, and Lucasfilm Ltd [43].

3.3 Fitting methods

3.3.1 MATLAB: lsqcurvefit

This method is a nonlinear least-squares solver whose objective is to find x that approaches the most to the observed data. It simply provides a convenient interface for data-fitting problems, and rather than compute the sum of squares, *lsqcurvefit* requires a user-defined function to compute the vector-valued function [42], which in this case, it is the kinetic function (for differential equation solving), described in the results.

3.3.2 SIMPSA and SIMPLEXL

These two methods were used by Perrin *et. al.*, 2020 [24] with the cybernetic modeling framework. Since both of them use the same Objective function and Differential Equation System function, they are embedded in the same MATLAB script. The only differences are the specifications SIMPSA or SIMPLEXL needs.

SIMPSA is a method developed in 1997 by Cardoso *et. al.* [45], which is a simulated annealing approach to non linear programming. This algorithm has the aim to find the solution for several variables, but since in this work there are just two of them (*i.e.* the kinetic constants V_{max} and K_m), the cool-rate was set < 1 for a slow convergence, finding just one point of convergence instead of several points.

SIMPLEXL is an algorithm based on the traditional simplex method, developed by Dantzig in 1947, which was described for the minimization of a function of n variables [46] [47], however an optimization was made by Aldo Buttini in 1993 (referenced in the code).

3.3.3 Python: SciPy optimize

This method consists on solving a differential equation system following the Michaelis-Menten-Monod equation, and then using the least squares method to find the better fit, provided by the *minimize* function included in the SciPy `optimize` package.

3.4 Assessing the quality of the fit

In order to compare the quality of each one of the fits, the points that describe the curves are needed, and those values are calculated using the obtained parameters.

Once the values were ready to be compared, four statistical methods were considered, such as Chi square, R square and Good of fitness (function by MATLAB). However, the Akaike's Information Criterion (AIC) was the chosen one because it provides a measure of model quality obtained by comparing the exit of computing several different methods. AIC is a fined technique based on in-sample fit to estimate the likelihood of a model to approximate certain values. This maximum likelihood principle can most effectively be applied for the decision of the final estimate of a finite parameter model when many alternative maximum likelihood estimates are obtained corresponding to the various restrictions of the model [48]. According to Akaike's theory, the most accurate model has the smallest AIC (see table 4.6 for the results).

Chapter 4

Results and Discussion

4.1 Kinetic constants: V_{max} and K_m

In order to perform a broad analysis of how *Pseudoalteromonas haloplanktis* uses 19 amino acids as different carbon sources, the first step is to obtain the maximum velocity (V_{max}) and Michaelis-Menten constant (K_m) for the nutrient uptake (kinetic constants).

Given the experimental data obtained from Perrin *et al.* [24], a fit was performed using the Michaelis-Menten-Monod kinetics equation [44], an approach that takes into account the bacterial growth while the carbon source diminishes:

$$\frac{dS}{dt} = \frac{-V_{max} * S * (S_0 + X_0 - S)}{(K_m + S)} \tag{4.1}$$

where S is the carbon source, S_0 is the initial carbon source, dS/dt the change of the carbon source throughout time and X_0 is the initial population density. This equation is used for the Kinetic function (*kinetic_fit_mm2*).

However, the initial population density was experimentally measured with the optical density (OD) of the bacterial culture, and if this quantity is multiplied by 0,74 we obtain the biomass in g/l [10]. In this sense, it was important to convert all concentration values from mM to g/l, which was easily done by multiplying by the amino acid's molecular weight and divide by 1000 (Table 4.1).

Since just one fitting method was not describing the correct behavior of each one of the amino acids' uptake, there were used four different methods and then, using a statistical test, compared each one of them for every amino acid. The fitting methods were *lsqcurvefit* (a function by Matlab), SIMPSA (a function made by Cardoso *et. al.*, 1997 [45]), SIMPLEXL (described by Dantzig [46] and optimized by Aldo Buttini) and a function done with Python using the *minimize* function (included in the SciPy optimize package) to compare the solution of the differential equation system and the experimental data.

Amino acids	**MW (g/mol)**	**C_i (mM)**	**C_i (g/l)**
Glutamate	147,13	0,2	0,029426
Glutamine	146,14	0,2	0,029228
Arginine	174,2	0,2	0,034840
Leucine	131,17	0,2	0,026234
Asparagine	132,12	0,2	0,026424
Aspartate	133,11	0,2	0,026622
Proline	115,13	0,2	0,023026
Valine	117,151	0,2	0,023430
Isoleucine	131,17	0,2	0,026234
Threonine	119,12	0,2	0,023824
Alanine	89,09	0,2	0,017818
Lysine	146,19	0,2	0,029238
Glycine	75,07	0,2	0,015014
Tyrosine	181,19	0,2	0,036238
Phenylalanine	165,19	0,2	0,033038
Serine	105,09	0,2	0,021018
Methionine	149,21	0,2	0,029842
Histidine	155,15	0,2	0,031031
Tryptophan	204,23	0,2	0,040846

Table 4.1: Amino acids initial concentration in *mM* and *g/l* (C_i = Initial concentration; MW = molecular weight).

4.1.1 MATLAB: lsqcurvefit

Main

The code starts setting the initial values: `vmax` and `km` are estimated values of the kinetic constants, while `s0` and `bm` comes from the experimental data. The value *s0* stands for the initial amino acids' concentrations and *bm* stands for biomass (*g/l*), which was calculated with the optical density of bacterial culture (experimentally measured) multiplied by 0,74. The fitting was done three times. The first time was done with the following estimated values for `vmax` and `km`:

```
        vmax = [.1 .20 .2 .1 .2 .05 .1 .1 .25 .09 .05 .1 .2 .1 .05 .1 .08 .05 .05];
        km = [1e-3 1e-5 1e-5 1e-6 1e-4 1e-3 1e-2 1e-4 1e-3 1e-3 1e-5 1e-3 1e-5 ...
            1e-3 1e-4 1e-5 1e-7 1e-6 1e-2];
```

The second time, the initial parameters were those obtained with the first fit, and the third time, were those obtained with the second fit. So the code goes as follows:

```
%Fit for all the aa
clear all
close all

%%Setting initial values
file4 = readtable('Parameters2.xls','PreserveVariableNames',true);
%Vmax (g/h)
vmax = table2array(file4(:,2))';
%Km (g/l)
km = table2array(file4(:,3))';
%Initial concentration of amino acids (g/l)
s0 = [0.029426 0.029228 0.03484 0.026234 0.026424 0.026622 0.023026 ...
    0.0234302 0.026234 0.02382384 0.017818 0.029238 0.015014 0.036238 ...
    0.033038 0.021018 0.029842 0.031031 0.040846];
%Biomass (g/l)
bm = 0.1295*0.74;
```

The next step is to upload all the experimental data, such as amino acids' concentrations (two sets of data), transforming them from mM into g/l values, the optical density of bacterial growth (transforming the od data to biomass) and obtain the mean values and error bars.

```
%%Load data: aa-mean
file1 = 'Figure4A_rep1.txt';
file2 = 'Figure4A_rep2.txt';
delimiterIn = ',';
headerlinesIn = 1;
A = importdata(file1,delimiterIn,headerlinesIn);
B = importdata(file2,delimiterIn,headerlinesIn);

% Molecular weight (for data loading and then ordered by clusters)
MW = 1e-3*[117.151 131.17 131.17 119.1192 89.09 147.13 146.14 149.21 ...
    132.12 133.11 146.19 75.07 115.13 181.19 155.1546 204.23 165.19 ...
    105.09 174.2];
MWaa = 1e-3*[147.13 146.14 174.2 131.17 132.12 133.11 115.13 117.151 ...
    131.17 119.1192 89.09 146.19 75.07 181.19 165.19 105.09 149.21 ...
    155.1546 204.23];

%19 aa-mean
val = (A.data(:,2)+B.data(:,2)).*MW(1)/2;
ile = (A.data(:,3)+B.data(:,3)).*MW(2)/2;
.
.
ser = (A.data(:,19)+B.data(:,19)).*MW(18)/2;
arg = (A.data(:,20)+B.data(:,20)).*MW(19)/2;

%Error bar
valsd = std([A.data(:,2) B.data(:,2)],0,2).*MW(1);
ilesd = std([A.data(:,3),B.data(:,3)],0,2).*MW(2);
.
.
sersd = std([A.data(:,19),B.data(:,19)],0,2).*MW(18);
argsd = std([A.data(:,20),B.data(:,20)],0,2).*MW(19);

```

```
        % Load data: OD mean and time
        file3 = 'Figure4C.txt';
        OD = importdata(file3,delimiterIn,headerlinesIn);
        odm = (OD.data(:,2)+OD.data(:,3))/2;
        BMm = odm * 0.74;

        %Time
        t = A.data(:,1);
```

Amino acids were ordered by clusters in the **aa** vector, following the degradation ordered previously found by Perrin et al. [24]. The **aaname** an **err** vector are going to be used for plotting.

```
        % This is the order by groups of aa
        aa = {glu, gln, arg, leu, asn, asp, pro, val, ile, thr, ala, lys, gly, ...
            tyr, phe, ser, met, his, trp};
        aaname = ["Glutamate", "Glutamine", "Arginine", "Leucine", "Asparagine", ...
            "Aspartate", "Proline", "Valine", "Isoleucine", "Threonine", ...
            "Alanine", "Lysine", "Glycine", "Tyrosine", "Phenylalanine", ...
            "Serine", "Methionine", "Histidine", "Tryptophan"];
        err = {glusd, glnsd, argsd, leusd, asnsd, aspsd, prosd, valsd, ilesd, ...
            thrsd, alasd, lyssd, glysd, tyrsd, phesd, sersd, metsd, hissd, trpsd};
```

Since the aim of this code is to find the kinetic constants for each amino acid, we build a table full of zeros that will be filled with the final results (**pall**).

For the fitting part of the code, a for loop was used in order to fit the data for each amino acid (**k = 1:19**). A reaction parameters vector was built with the initial values. A matrix containing all the uploaded data and its upper and lower bounds were settled. All this elements are the inputs of **lsqcurvefit** function, along with the @*kinetic_fit_mm2* function (see its code bellow: Kinetic function). A specific graphic was plotted for each iteration, saving it in one same figure using the *subplot* function, and the kinetic constants were added to the **pall** table.

```
        pall = zeros(19,2);

        %%Fitting
        for k = 1:19
        reactionParameters = [vmax(k) km(k) s0(k) bm];
        Data_PC=[t t aa{k} BMm];
        theta0=reactionParameters;
        lb = [.01, 1e-5, s0(k), bm];
        ub = [.3, 1, s0(k), bm];

        [p,Rsdnrm,Rsd,ExFlg,OptmInfo,Lmda,Jmat]=lsqcurvefit
        (@kinetic_fit_mm2,theta0,t,Data_PC(:,3:4), lb, ub);

        c = Data_PC(:,2:3);
        tv = linspace(min(Data_PC(:,1)), max(Data_PC(:,1)), length(t));
        Cfit = kinetic_fit_mm2(p, tv)./MWaa(k);
        %Plotting figures
        figure(1);
```

```
        subplot(5,4,k);
        plot(t,Cfit(:,1),'b'); title(aaname(k));
        hold on
        errorbar(t, c(:,2)./MWaa(k),err{k}./MWaa(k),'.b');

        %Collecting Vmax and Km parameters
        format long
        pall(k,:) = p(1:2);
        end
```

Finally, the image was customized with its correct axes (Fig. 4.1) and the table with the kinetic constants (parameters) is saved as a .xls file (Tab. 4.2).

```
        %%Naming the axes:
        han=axes(figure(1),'visible','off');
        han.Title.Visible='on';
        han.XLabel.Visible='on';
        han.YLabel.Visible='on';
        ylabel(han,'\fontsize{14}Concentration (mM)');
        xlabel(han,'\fontsize{14}Time (h)');
        title(han,'\fontsize{16} With lsqcurvefit');

        %Table
        VarNames = {'Amino acids','Vmax (g/h)','Km (g/l)'};
        Names = aaname';
        T=table(Names,pall(:,1),pall(:,2),'VariableNames',VarNames);
        writetable(T,'Parameters3.xls');
```

Kinetic function

In the following code there is a nested function (*i.e.* a function defined within another function). The aim is to define the equation for the fitting, named *rate* (see the Michaelis-Menten-Monod equation 4.1) and then solve it as a differential equation (*kinetic_fit_mm2*).

It starts with the initial values of concentration and biomass, and then, the parameters are given to the *rate* function.

```
        % Non-linear fitting of the Michaelis-Menten-Monod model

        function C=kinetic_fit_mm2(theta,t)

        %Initial concentration (g/l)
        CS0 = theta(3);
        %Initial biomass (g/l)
        BM0 = 0.1295*0.74;

        Species0 = [CS0 BM0];

        %Differential equation system:
        [T,Cv]=ode45(@(t,y) rate(t,y,theta), t, Species0);

```

```
function ydot=rate(¬,y, theta)
ydot=zeros(2,1);
vmax=theta(1);
Km=theta(2);
Ci = theta(3);
BMi = theta(4);

%Michaelis-Menten-Monod equation:
ydot(1)=-vmax.*y(1)*(Ci + BMi - y(1))./(Km+y(1));

end

C=Cv(:,1:2);

end
```

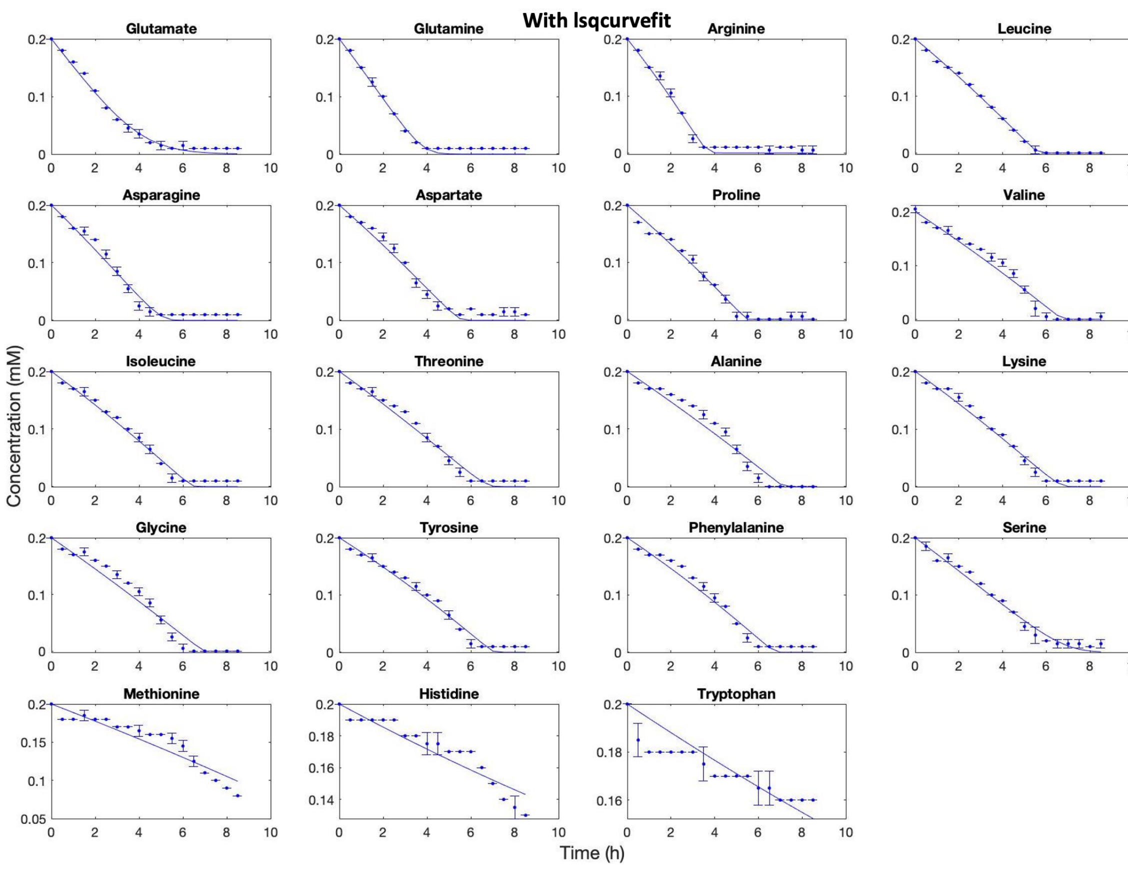

Figure 4.1: Amino acids uptake by *Pseudoalteromonas haloplanktis* following the degradation order found by Perrin *et al.*, 2020 [24]. The fit method used was *lsqcurvefit* by MATLAB.

Amino acids	V_{max} (g/h)	K_m (g/l)	Amino acids	V_{max} (g/h)	K_m (g/l)
Glutamate	0,107969	0,013320	Alanine	0,024011	0,000030
Glutamine	0,083979	0,002933	Lysine	0,042071	0,000522
Arginine	0,088833	0,000671	Glycine	0,021187	0,000034
Leucine	0,044154	0,000184	Tyrosine	0,046507	0,000210
Asparagine	0,053598	0,001152	Phenylalanine	0,044644	0,000161
Aspartate	0,047202	0,000494	Serine	0,034223	0,001920
Proline	0,039771	0,000024	Methionine	0,017235	0,000030
Valine	0,032828	0,000066	Histidine	0,137301	0,323246
Isoleucine	0,038777	0,000241	Tryptophan	0,279822	0,844227
Threonine	0,034756	0,000468			

Table 4.2: Maximum velocity and Michaelis constant for each amino acid obtained with *lsqcurvefit* by MATLAB as the fitting method.

4.1.2 SIMPSA and SIMPLEXL

Main

The code starts as the previous one (lsqcurvefit), *i.e.* importing the experimental data of the amino acids' concentration values, the biomass data and the time vector. However, it is defined a `Data` matrix with all of the aminoacids' mean and the time vector. It is done this way in order to be used in the objectiveFunction script.

```
clear all
close all

%%Load dat: aa-mean
file1 = 'Figure4A_rep1.txt';
file2 = 'Figure4A_rep2.txt';
delimiterIn = ',';
headerlinesIn = 1;
A = importdata(file1,delimiterIn,headerlinesIn);
B = importdata(file2,delimiterIn,headerlinesIn);

% Molecular weight (for data loading and then ordered by clusters)
MW = 1e-3*[117.151 131.17 131.17 119.1192 89.09 147.13 146.14 149.21 ...
    132.12 133.11 146.19 75.07 115.13 181.19 155.1546 204.23 165.19 ...
    105.09 174.2];
MWaa = 1e-3*[147.13 146.14 174.2 131.17 132.12 133.11 115.13 117.151 ...
    131.17 119.1192 89.09 146.19 75.07 181.19 165.19 105.09 149.21 ...
    155.1546 204.23];

%19 aa-mean (g/l)
val = (A.data(:,2)+B.data(:,2)).*MW(1)/2;
ile = (A.data(:,3)+B.data(:,3)).*MW(2)/2;
.
.
ser = (A.data(:,19)+B.data(:,19)).*MW(18)/2;
arg = (A.data(:,20)+B.data(:,20)).*MW(19)/2;
```

```

%Error bar
valsd = std([A.data(:,2) B.data(:,2)],0,2).*MW(1);
ilesd = std([A.data(:,3),B.data(:,3)],0,2).*MW(2);
.
.
sersd = std([A.data(:,19),B.data(:,19)],0,2).*MW(18);
argsd = std([A.data(:,20),B.data(:,20)],0,2).*MW(19);

% OD mean
file3 = 'Figure4C.txt';
OD = importdata(file3,delimiterIn,headerlinesIn);
odm = (OD.data(:,2)+OD.data(:,3))/2;

%Biomass (g/l)
BMm = odm * 0.74;

%Time
vTime = A.data(:,1);

% This is the order by groups of aa
aa = {glu, gln, arg, leu, asn, asp, pro, val, ile, thr, ala, lys, gly, ...
    tyr, phe, ser, met, his, trp};
aaname = ["Glutamate", "Glutamine", "Arginine", "Leucine", "Asparagine", ...
    "Aspartate", "Proline", "Valine", "Isoleucine", "Threonine", ...
    "Alanine", "Lysine", "Glycine", "Tyrosine", "Phenylalanine", ...
    "Serine", "Methionine", "Histidine", "Tryptophan"];
err = {glusd, glnsd, argsd, leusd, asnsd, aspsd, prosd, valsd, ilesd, ...
    thrsd, alasd, lyssd, glysd, tyrsd, phesd, sersd, metsd, hissd, trpsd};

%Data matrix
Data = [vTime glu gln arg leu asn asp pro val ile thr ala lys gly tyr phe ...
    ser met his trp];
```

The fit was done three times for both methods. The first time it was done with these estimated values for `vmax` and `km`:

```
%For SIMPSA:
vmax= [0.077516 0.073205 0.090023 0.044131 0.0508381 0.044456 0.037376 ...
    0.031659 0.042409 0.036098 0.041137 0.041348 0.0248259 0.047499 ...
    0.045926 0.027391 0.017013 0.010768 0.56459];
km= [0.004186 0.000048 0.000049 0.000419 0.0006 0.000063 0.000033 ...
    0.000079 0.000221 0.000132 0.00021 0.000092 0.00078 0.00031 0.000184 ...
    0.000046 0.000364 0.000959 1.523861];

%For SIMPLEX:
vmax= [0.199999990 0.197725481 0.198131475 0.157049833 0.0448949 ...
    0.0449999 0.0378840682 0.036661369 0.036508738 0.022746 0.0326134321 ...
    0.0436398755 0.008308 0.053724821 0.076912766 0.039141558 ...
    0.0089995423 0.005943 0.56459];
km= [0.001518 0.000051 0.000053 0.0004916 0.00000599 0.000653 0.00002830 ...
    0.000079 0.0003894 0.00002398 0.0000248 0.00009 0.00003 0.00003145 ...
    0.000201 0.0007568 0.0002 0.0006 1.523861];
```

After the second time, the resulted parameters were used as estimated values for the third fit, loading the data the same way it was loaded for the first method (*lsqcurvefit*).

```
%% Initial guess for SIMPSA
file4 = readtable('ParametersSIMPSA2.xls','PreserveVariableNames',true);
vmax = table2array(file4(:,2))';
km = table2array(file4(:,3))';

%% Initial guess for SIMPLEXL
file4 = readtable('ParametersSIMPLEXL2.xls','PreserveVariableNames',true);
vmax = table2array(file4(:,2))';
km = table2array(file4(:,3))';
```

The table of zeros `pall` was build to be filled with the kinetic constants calculated by the algorithms. Then the for loop starts with the inputs for the SIMPSA and SIMPLEX algorithms. Those are the `ModelParameter` and the `Data_PC` matrix that contains only the time column and the amino acid of the k iteration. The inputs needed are the `OPTIONS` value for setting the cool-rate value for the SIMPSA algorithm and an `options` vector for SIMPLEXL algorithm. In both cases, the input function is `objectiveFuntion`.

After the algorithms were implemented solving the differential equations, the solutions of the parameters calculated, saved in the `parCal` matrix, are used as an input for the `odeSystem` function, which compare how distant the points obtained with those parameters are from the experimental data (least square method).

Finally, at the end of the for loop, a specific graphic for each amino acid was plotted, saving them in the same figure using the *subplot* function, and the parameters tested (in `sol`) were saved in the `pall` matrix.

```
pall = zeros(19,2);

for k = 1:19
ModelParameters = [vmax(k) km(k)];
Data_PC = [Data(:,1) Data(:,k+1)];

%For SIMPSA
OPTIONS = SIMPSASET('COOL_RATE',0.8);

%For SIMPLEXL
options=foptions;
%1 = displays intermidiate results
options(1)=1;
%Precision required for the solution value to X
options(2)=.0001;
%Precision required for the value of the functional error to the solution
options(3)=.0001;
%Maximum number of iterations after which the search for the minimum stops
options(14)=1500;

% SIMPSA algorithm
[parCal,FVAL,EXITFLAG,OUTPUT] = ...
    SIMPSA('objectiveFunction',ModelParameters,...
zeros(size(ModelParameters)),2*ones(size(ModelParameters)),OPTIONS,Data_PC);

```

```
        % SIMPLEXL algorithm
        [parCal,¬] = ...
            SIMPLEXL('objectiveFunction',ModelParameters,options,[],Data_PC);

        %Testing the parameters calculated:
        y0 = s0(k);
        [time,sol] = ode45(@(t,y) odeSystem(t,y,parCal), vTime, y0);


        %Plotting the results to experimental data
        figure(1);
        subplot(5,4,k)
        plot(time,sol(:,1)./MWaa(k),'b'); title(aaname(k));
        hold on
        errorbar(time, Data_PC(:,2)./MWaa(k),err{k}./MWaa(k),'.b');

        %Collecting Vmax and Km parameters
        format long
        pall(k,:) = parCal(1:2);

        end
```

At the end of the script, the image was customized with its correct axis (Fig. 4.2, Fig. 4.3) and the table with the kinetic constants (parameters) was saved as a .xls file for each method (Table 4.3, Table 4.4).

```
        %Naming the axis of the figure
        han=axes(figure(1),'visible','off');
        han.Title.Visible='on'; han.XLabel.Visible='on'; han.YLabel.Visible='on';
        ylabel(han,'\fontsize{14}Concentration (mM)');
        xlabel(han,'\fontsize{14}Time (h)');
        %Titles for each method:
        title('\fontsize{16}With SIMPSA');
        title('\fontsize{16}With SIMPLEXL');

        %Table
        VarNames = {'Amino_acids','Vmax (g/h)','Km (g/l)'};
        Names = aaname(1:19)';
        T=table(Names,pall(:,1),pall(:,2),'VariableNames',VarNames);
        %Saving for each method:
        writetable(T,'ParametersSIMPSA3.xls');
        writetable(T,'ParametersSIMPLEXL3.xls');
```

It is important to know that this script is run in two separate times: once for the SIMPSA method and an other time for the SIMPLEXL method. This is due to the amount of calculations that are involved, and doing it all at once could be too slow.

In that sense, when running the script for the former algorithm, all lines setting parameters of the latter algorithm must be commented.

Ode System

This function has the Michaelis-Menten-Monod equation for the fitting, which is going to be used by the objective function in order to solve it as a differential equation system. Here the initial concentration and biomass are set, as well as the reaction parameters, *i.e.* the kinetic constants.

```
function ddy = odeSystem(¬,y,reactionParameters)

ddy=zeros(1,1);

%Initial concentration and biomass
Ci = 0.02754584;
BMi = 0.1295*0.74;

%vmax
vmax = reactionParameters(1);
%km
Km = reactionParameters(2);

%Michaelis-Menten-Monod equation
ddy(1)= -vmax.*y(1)*(Ci + BMi - y(1))./(Km+y(1));
```

Objective fuction

This function is the input function that is requested by the SIMPSA and SIMPLEXL methods. It needs a `Data` matrix as an input, that was called `Data_PC` in the main script. Using that input, the time and concentration vectors were set, as well as the initial concentration value (`y0`), so the *ode45* function could be used.

After that, a figure of every try is plotted and shown with the *shg* command, so each fit try could be seen. Once the solution of the parameters is calculated, they are send to a least squares test, to defined if they are describing the experimental data in an accurate way.

```
function scarto = objectiveFunction(reactionParameters,Data)

reactionParameters=abs(reactionParameters);

%Setting the time and experimental data vectors
vTime = Data(:,1);
aa = Data(:,2);
%Initial concentration
y0 = aa(1);

[Time,sol] = ode45(@(t,y) odeSystem(t,y,reactionParameters), vTime, y0);

figure(2);
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);
plot(vTime,sol,'k',vTime,aa,'ok','LineWidth', 2);
xlabel('Time(h)');
```

```
ylabel('Concentration (g/l)');

shg

%Least squares test

method=3;
switch method
case 1
scarto=(1/length(vTime))*sum((sol-aa).^2+(sol-aa).^2)/2;
case 2
scarto=(1/length(vTime))*sum((aa.*(aa-sol)).^2)/2;
case 3
x1D=aa(2:length(aa));
x2D=[x1D;x1D(end)];
dx2D=x2D-aa;

scarto=(1/length(vTime))*sum((dx2D.*(aa-sol)).^2);
end

end
```

Amino acids	V_{max} (g/h)	K_m (g/l)	Amino acids	V_{max} (g/h)	K_m (g/l)
Glutamate	0,077516	0,004186	Alanine	0,021529	0,000021
Glutamine	0,073350	0,000119	Lysine	0,041356	0,000017
Arginine	0,090056	0,000038	Glycine	0,143405	0,069198
Leucine	0,043012	0,000005	Tyrosine	0,047640	0,000003
Asparagine	0,050838	0,000600	Phenylalanine	0,045926	0,000184
Aspartate	0,044456	0,0000634	Serine	0,027391	0,000046
Proline	0,037390	0,000028	Methionine	0,017013	0,000364
Valine	0,031659	0,000079	Histidine	0,010768	0,000959
Isoleucine	0,041124	0,001669	Tryptophan	0,749563	1,982687
Threonine	0,034729	0,000984			

Table 4.3: Maximum velocity and Michaelis constant for each amino acid obtained with *SIMPSA* as the fitting method.

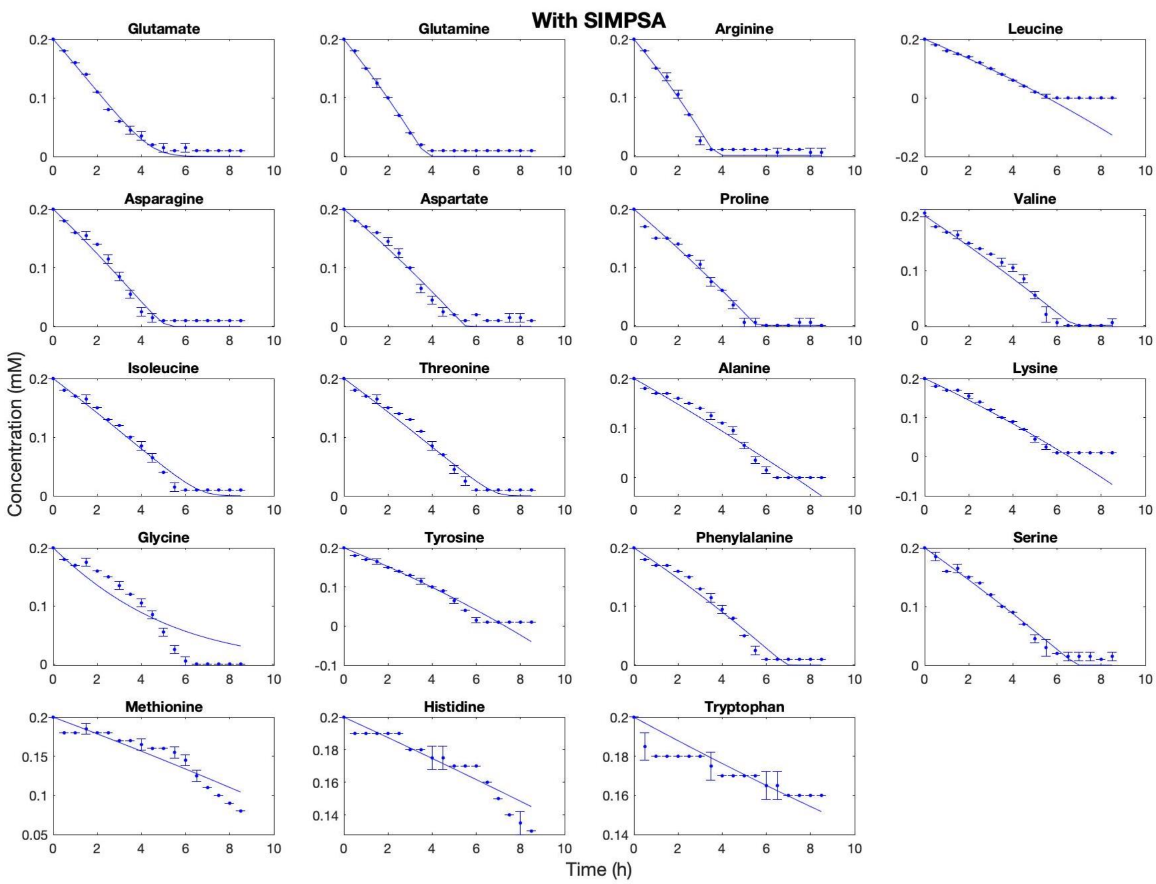

Figure 4.2: Amino acids uptake by *Pseudoalteromonas haloplanktis* following the degradation order found by Perrin *et al.*, 2020 [24]. The fit method used was *SIMPSA*.

Amino acids	V$_{max}$ (g/h)	K$_m$ (g/l)	Amino acids	V$_{max}$ (g/h)	K$_m$ (g/l)
Glutamate	0,077516	0,004186	Alanine	0,021137	0,000021
Glutamine	0,073205	0,000048	Lysine	0,041348	0,000092
Arginine	0,090023	0,000049	Glycine	0,018259	0,000058
Leucine	0,044131	0,000419	Tyrosine	0,049499	0,000031
Asparagine	0,048381	0,000006	Phenylalanine	0,045926	0,000184
Aspartate	0,044456	0,000063	Serine	0,027391	0,000046
Proline	0,037376	0,000033	Methionine	0,017013	0,000364
Valine	0,031659	0,000079	Histidine	0,010768	0,000959
Isoleucine	0,037409	0,000021	Tryptophan	6,84E+13	1,85E+14
Threonine	0,032098	0,000032			

Table 4.4: Maximum velocity and Michaelis constant for each amino acid obtained with *SIMPLEXL* as the fitting method.

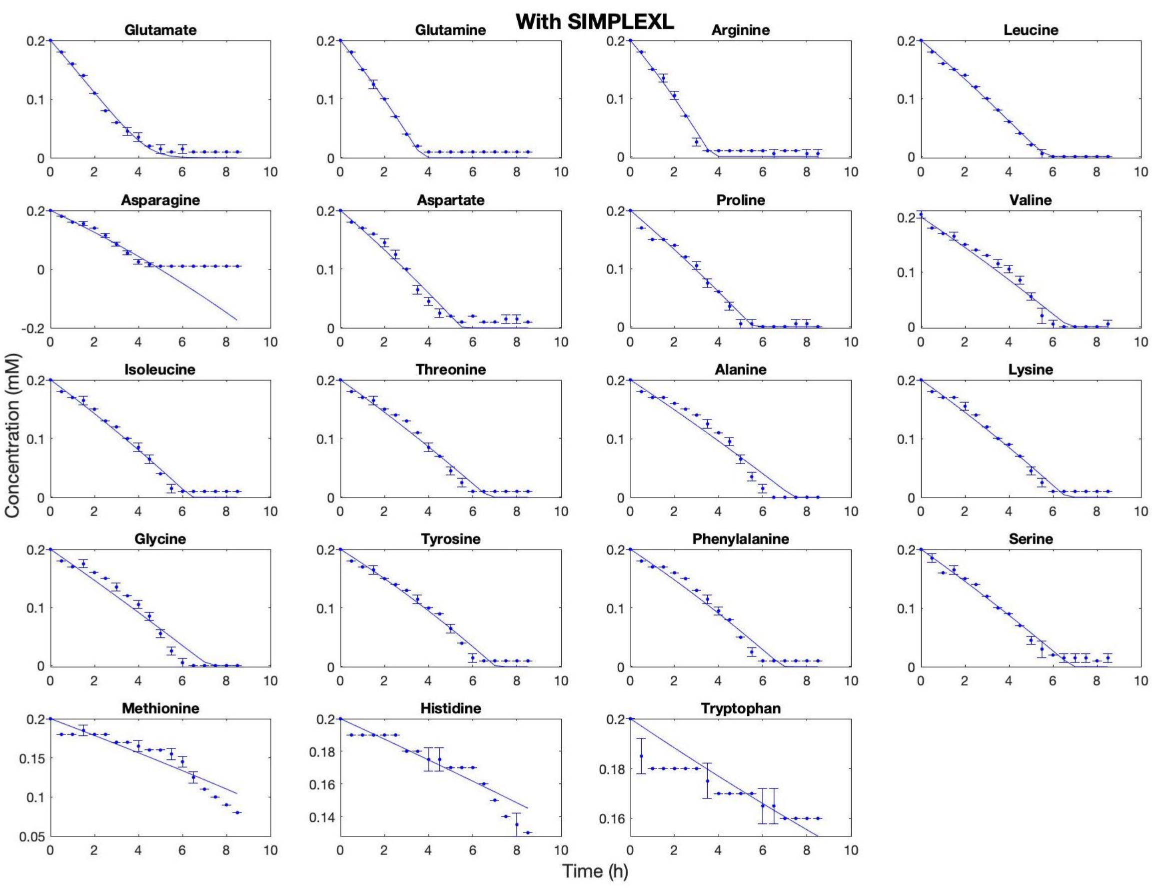

Figure 4.3: Amino acids uptake by *Pseudoalteromonas haloplanktis* following the degradation order found by Perrin *et al.*, 2020 [24]. The fit method used was *SIMPLEXL*.

4.1.3 Python: SciPy optimize

In this case, there is only one script, and the function is defined here. It starts by importing the libraries needed. Then uploading the data and setting the molecular weight vector and the parameters vectors (kinetic constants). However, the main difference with the other fitting methods is that an offset is defined in order to prevent the function going to negative values.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from scipy.optimize import minimize

#Importing data
data = pd.read_excel('data.xls')
data = data.values

name = ["Glutamate", "Glutamine", "Arginine", "Leucine", "Asparagine", ...
    "Aspartate", "Proline", "Valine", "Isoleucine", "Threonine", ...
    "Alanine", "Lysine", "Glycine", "Tyrosine", "Phenylalanine", ...
    "Serine", "Methionine", "Histidine", "Tryptophan"];

#Molecular weight:
MWaa = 1e-3*np.array[147.13 146.14 174.2 131.17 132.12 133.11 115.13 ...
    117.151 131.17 119.1192 89.09 146.19 75.07 181.19 165.19 105.09 ...
    149.21 155.1546 204.23];

#Kinetic constants vectors to be filled:
vmax_all = np.zeros(len(MWaa))
Km_all = np.zeros(len(MWaa))
#Offset to prevent the functions to go bellow zero.
offset_all = np.zeros(len(MWaa))
```

This method was used just one time given its efficiency, and the estimated values for the kinetic constants were:

```
#Estimated values of the parameters.
vmax_0_vec = ...
    np.array([0.077,0.074,0.088,0.044,0.054,0.048,0.041,0.037,0.042, ...
0.036,0.04,0.043,0.04,0.048, 0.047,0.030,1.102,0.452,1.007])
Km_0_vec = np.array([4.03e-03, 6.15e-04, 2.36e-04, 5.81e-05, 5.30e-04, ...
    9.57e-05, 1.77e-05, 2.36e-04, 2.60e-04, 1.06e-04, 1e-05, 1.22e-05, ...
    1e-05, 1.51e-05, 1.00e-04, 1.69e-06,5.00e-01, 2.02e-01, 4.36e-01])
```

The for loop starts in the following part of the code. A `cFact` factor is defined to be used for plotting in mM instead of g/l. The initial concentration, the biomass and the time vector are set, as well as the parameters V_{max} and K_m.

```
for k in range(1,19):
cFact = 1/MWaa[k-1]

Ci = data[0,k]
BMi = 0.1295*0.74

#Time vector
tvec=np.arange(0,9,0.5)

offset_0 = data[-1,k]
vmax_0 = vmax_0_vec[k-1]
Km_0 = Km_0_vec[k-1]
```

Both of the needed functions are defined here. The first one solves the differential equation system using the Michaelis-Menten-Monod equation (Eq. 4.1) and the second one returns the difference between the calculated value and the experimental one, which will be minimize at the time of the plotting.

Printing every graphic is done using an analog function of python, also named *subplot* (Fig. 4.4). And it is here where the points to be graphed are choose according to which ones are closest to the experimental data (*minimize* function). Finally, the parameters are saved in one matrix (Table 4.5).

```
#Diferential equation system:
def f_aux(t, offset,vmax,Km):

#offset=0
y0 = Ci - offset # function evaluation at t=0
def ddy(y,t):

#Michaelis-Menten-Monod equation:
return -vmax*y*(Ci + BMi - y)/(Km + y)
return offset + odeint(ddy,y0,t)[:,0]

#Least squares function
def f_aux_2(x):
offset,vmax,Km = x
return sum((f_aux(tvec,offset,vmax,Km) - data[:,k])**2)

#Print(f_aux_2([offset,vmax,Km]))
plt.subplot(5,4,k)
#Minimizing the error
sol = minimize(f_aux_2,np.array([offset_0,vmax_0,Km_0]))
params = sol.x
solODE = f_aux(tvec,*params)

plt.plot(tvec,solODE*cFact,'C0-')
plt.plot(data[:,0],data[:,k]*cFact,'C0.')
plt.ylim(0,0.2)
plt.title(name[k-1],fontsize=11)

#Printing the parameters for each amino acid
print(params)
offset_all[k-1],vmax_all[k-1],Km_all[k-1]=params

#Saving the parameters
```

```
print('\nd#    vmax        km')
for i in range(len(vmax_all)):
print('{0:02d}  {1:.5f}  {2:.7f}'.format(i+1,vmax_all[i],Km_all[i]))

plt.tight_layout(-0.1)
plt.show()
```

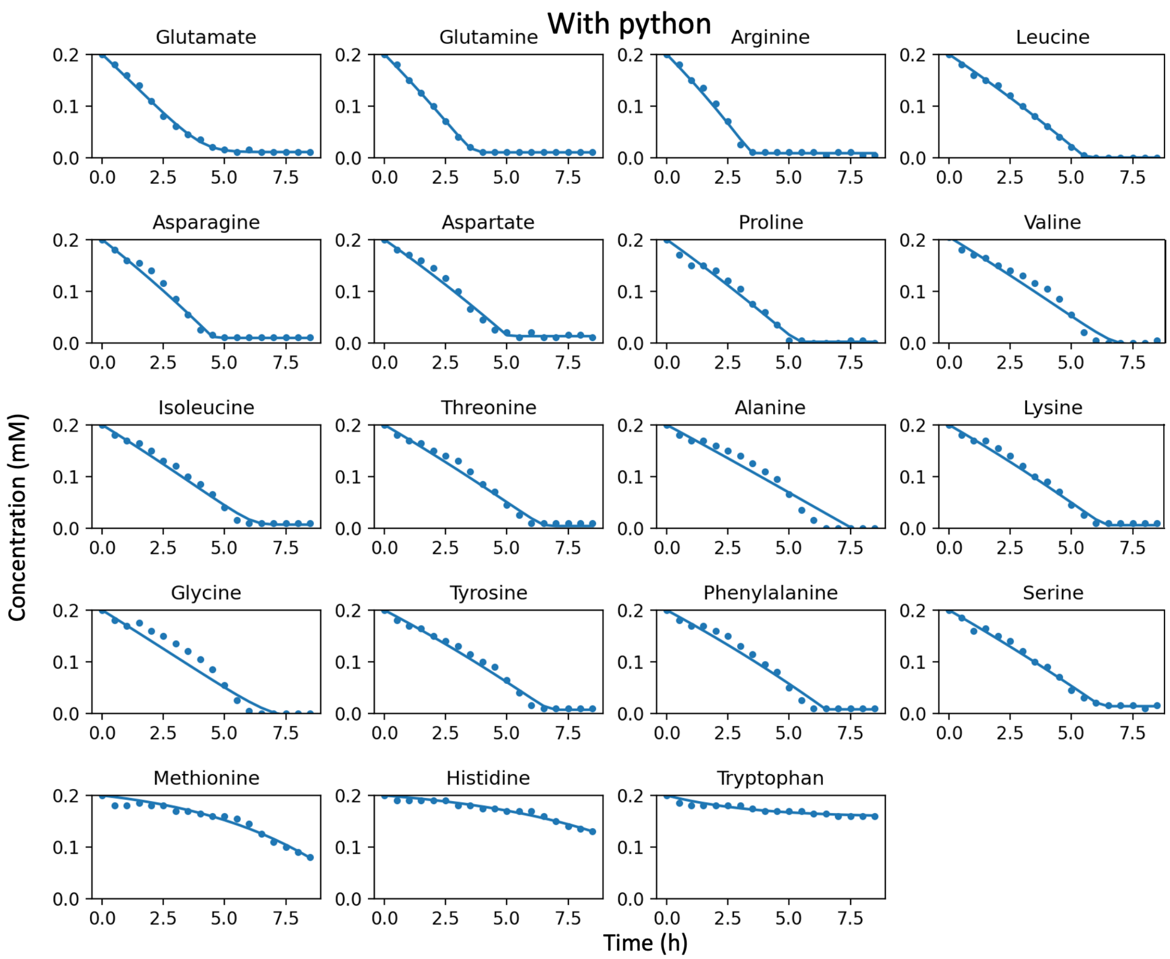

Figure 4.4: Amino acids uptake by *Pseudoalteromonas haloplanktis* following the degradation order found by Perrin *et al.*, 2020 [24]. Obtained with *Python's* fitting method.

Amino acids	**V$_{max}$ (g/h)**	**K$_m$ (g/l)**	**Amino acids**	**V$_{max}$ (g/h)**	**K$_m$ (g/l)**
Glutamate	0,077010	0,004016	Alanine	0,045210	0,017377
Glutamine	0,073660	0,000615	Lysine	0,041690	0,000225
Arginine	0,086810	0,000185	Glycine	0,024200	0,000740
Leucine	0,043690	0,000059	Tyrosine	0,046900	0,000267
Asparagine	0,050790	0,000013	Phenylalanine	0,044410	0,000003
Aspartate	0,045220	0,000016	Serine	0,029470	0,000001
Proline	0,040830	0,000183	Methionine	1,166420	0,536755
Valine	0,036340	0,000742	Histidine	0,472550	0,217166
Isoleucine	0,041350	0,000998	Tryptophan	1,007000	0,435997
Threonine	0,034790	0,000288			

Table 4.5: Maximum velocity and Michaelis constant for each amino acid obtained with *Python's* fitting method.

4.1.4 The fitting methods

The *lsqcurvefit* and *SIMPSA* method were compiled three times, looking forward to obtain better results. After the third time, the values of V_{max} and K_m obtained are shown in the tables 4.2 and 4.3, respectively, and the trajectories of the fit are observed in the figures 4.1 and 4.2, respectively.

However, the *SIMPLEXL* method was compiled three times obtaining negative parameters the last two times. Since it can not be explained from the biological point of view, the parameters that were taken as an acceptable result were those obtained the first time (Table 4.4, Figure 4.3).

Python's fitting method was compiled just one time, and the results obtained are shown in the table 4.5, and figure 4.4.

It can be noticed that the fit obtained with the *SIMPSA* method for leucine, alanine, lysine and tyrosine (Fig. 4.2), had the problem of taking negative concentration values, which may be related with the absence of an offset, letting the values go further. The same problem can be observed with the fit obtained with the *SIMPLEXL* method for asparagine (Fig. 4.3). It is worth to say that neither of the kinetic constants corresponding the previous fittings for those amino acids were chosen by the Akaike's Information Criteria.

4.1.5 Assessing the quality of the fit: AIC

Obtaining the data from the fit

The code used here needs to load the kinetic constants calculated with each of the fits, then with a for loop solve the differential equation and save each value. The code needs to be run four times, once for each method, commenting the parts that correspond to the others.

```
% Obtaining the data set for each fit

clear all
close all

%Setting the initial concentration and time vectors
s0 = [0.029426 0.029228 0.03484 0.026234 0.026424 0.026622 0.023026 ...
    0.0234302 0.026234 0.02382384 0.017818 0.029238 0.015014 0.036238 ...
    0.033038 0.021018 0.029842 0.031031 0.040846];
vTime = linspace(0,8.5,18)';

%Load the data and generate the point of the curve:

par = readtable('ParametersLSCURVE3.xls','PreserveVariableNames',true);
% par = readtable('ParametersSIMPSA3.xls','PreserveVariableNames',true);
% par = readtable('ParametersSIMPLEXL1.xls','PreserveVariableNames',true);
% par = readtable('ParametersPython.xls','PreserveVariableNames',true);
vmax = table2array(par(:,2))';
km = table2array(par(:,3))';

pall = zeros(18,19);

for k = 1:19
reactionParameters = [vmax(k) km(k)];
y0 = s0(k);
[Time,sol] = ode45(@(t,y) odeSystem(t,y,reactionParameters), vTime, y0);
pall(:,k) = sol(:,1);

end

%Saving all the values in the same matrix
varNames = ...
    ["Time","Glutamate","Glutamine","Arginine","Leucine","Asparagine",...
"Aspartate","Proline","Valine","Isoleucine","Threonine","Alanine","Lysine",...
"Glycine",  "Tyrosine","Phenylalanine","Serine","Methionine","Histidine",...
"Tryptophan"];
T=table(vTime,pall(:,1),pall(:,2),pall(:,3),pall(:,4),pall(:,5),pall(:,6),...
pall(:,7),pall(:,8),pall(:,9),pall(:,10),pall(:,11),pall(:,12),pall(:,13),...
pall(:,14),pall(:,15),pall(:,16),pall(:,17),pall(:,18),pall(:,19),...
'VariableNames',varNames);
writetable(T,'Data_parLSCURVE.xls');
% writetable(T,'Data_parSIMPSA.xls');
% writetable(T,'Data_parSIMPLEXL.xls');
% writetable(T,'Data_parPy.xls');
```

Akaike's Information Criterion

The method used to compare the four fitting methods was the Akaike's Information Criterion, which returned negative values, meaning how much information is lost depending on the method used.

Since the aim of this code is to compare the experimental data to the results obtained from the fitting methods, the first thing that has to be done is to upload all of them, and proceed with a for loop to compare each amino acid's value obtained with the parameters the fitting methods gave as a result, to the experimental data.

Finally, the results were saved as a .xls file and since the AIC estimates the relative amount of information lost by a given model: the less information a model loses, the higher the quality of that model, hence, the smaller the value is, the better (Table 4.6).

```
% Uploading experimental data
Data1 = readtable('ExperimentalData.xls','PreserveVariableNames',true);
Data = table2array(Data1);
vTime = Data(:,1);

% Names:
aaname = ["Glutamate", "Glutamine", "Arginine", "Leucine", "Asparagine", ...
    "Aspartate", "Proline", "Valine", "Isoleucine", "Threonine", ...
    "Alanine", "Lysine", "Glycine", "Tyrosine", "Phenylalanine", ...
    "Serine", "Methionine", "Histidine", "Tryptophan"];

%Load the data to compare with:
datcalLS = readtable('Data_parLSCURVE.xls','PreserveVariableNames',true);
datcalSIMPSA = readtable('Data_parSIMPSA.xls','PreserveVariableNames',true);
datcalSIMPLEXL = ...
    readtable('Data_parSIMPLEXL.xls','PreserveVariableNames',true);
datcalPY = readtable('Data_parPy.xls','PreserveVariableNames',true);

pall = zeros(18,4);

for k = 1:19

datLS = table2array(datcalLS(:,k+1));
datSIMPSA = table2array(datcalSIMPSA(:,k+1));
datSIMPLEXL = table2array(datcalSIMPLEXL(:,k+1));
datPy = table2array(datcalPY(:,k+1));

% AIC estimation: The lowest value corresponds to the better fit
z1 = iddata(Data(:,k+1),datLS,0.08);
z2 = iddata(Data(:,k+1),datSIMPSA,0.08);
z3 = iddata(Data(:,k+1),datSIMPLEXL,0.08);
z4 = iddata(Data(:,k+1),datPy,0.08);
np = 2;
sys1 = tfest(z1,np);
sys2 = tfest(z2,np);
sys3 = tfest(z3,np);
sys4 = tfest(z4,np);
aicLS = aic(sys1,'nAIC');
aicSIMPSA = aic(sys2,'nAIC');
aicSIMPLEXL = aic(sys3,'nAIC');
aicPy = aic(sys4,'nAIC');
pall(k,:) = [aicLS aicSIMPSA aicSIMPLEXL aicPy];

end

fitname = ["Aminoacids","lsqcurvefit","SIMPSA","SIMPLEXL","Python"];
Names = aaname(1:19)';
T=table(Names, pall(:,1), pall(:,2), pall(:,3), pall(:,4), ...
    'VariableNames', fitname);
writetable(T,'CheckFit_AIC.xls');
```

In the table 4.6, the values shown in bold are the ones that correspond to the models that lose less information. Here it can be seen, for example, that SIMPLEXL method's result for tryptophan is the one that loses the more information, which was expected since the kinetic constants were so much higher (Table 4.4) than those obtained with the other methods.

Amino acids	lsqcurvefit	SIMPSA	SIMPLEXL	Python
Glutamate	-11,642	-11,091	-11,684	**-11,846**
Glutamine	-11,088	-9,620	-9,598	**-11,299**
Arginine	-10,183	-9,747	-9,933	**-10,255**
Leucine	**-14,558**	-11,080	-8,101	-12,289
Asparagine	**-11,984**	-10,080	-10,950	-11,499
Aspartate	-7,940	-11,967	**-12,110**	-10,360
Proline	-10,243	**-12,888**	-10,026	-10,733
Valine	**-12,856**	-9,738	-12,029	-7,615
Isoleucine	-12,747	-11,417	-12,815	**-12,836**
Threonine	-8,996	-11,662	-8,035	**-13,128**
Alanine	-8,116	-12,554	**-12,691**	-10,731
Lysine	-12,309	-8,649	-11,858	**-12,358**
Glycine	-14,641	-11,770	-13,001	**-14,876**
Tyrosine	-11,474	-11,190	**-11,836**	-9,862
Phenylalanine	-9,027	**-11,620**	-9,671	-10,864
Serine	-8,878	-7,778	**-13,105**	-9,716
Methionine	-5,802	-12,099	**-12,099**	-7,604
Histidine	-5,401	-9,978	**-14,004**	-9,521
Tryptophan	-14,703	**-15,113**	-8,212	-11,934

Table 4.6: Results of the Akaike's Information Criterion when comparing the four fitting methods. Those who lose the less information are shown in bold.

After identifying the parameters that better described the amino acids' behaviour, the kinetic constants were put in a single file and then, using the molecular weight information, the values were converted from *g/h* and *g/l*, to *mmol/h* and *mmol/l*, for V_{max} and K_m, respectively (Table 4.7). It was done this way because DFBAlab code has a vector with the molecular weights, to convert the units to *g/h* and *g/l* just in the cases needed.

	Values after AIC		*Values for DFBAlab*	
Amino acids	$\mathbf{V}_{max}$ **(g/h)**	$\mathbf{K}_m$ **(g/l)**	$\mathbf{V}_{max}$ **(mmol/h)**	$\mathbf{K}_m$ **(mmol/l)**
Glutamate	0,077010	0,004016	0,523415	0,027292
Glutamine	0,073660	0,000615	0,504037	0,004207
Arginine	0,086810	0,000185	0,498335	0,001061
Leucine	0,044154	0,000184	0,336616	0,001405
Asparagine	0,053598	0,001152	0,405673	0,008723
Aspartate	0,044456	0,000063	0,333978	0,000475
Proline	0,037390	0,000028	0,324767	0,000241
Valine	0,032828	0,000066	0,280221	0,000562
Isoleucine	0,041350	0,000998	0,315240	0,007608
Threonine	0,034790	0,000288	0,292060	0,002421
Alanine	0,021137	0,000021	0,237260	0,000231
Lysine	0,041690	0,000225	0,285177	0,001537
Glycine	0,024200	0,000740	0,322366	0,009857
Tyrosine	0,049499	0,000031	0,273186	0,000171
Phenylalanine	0,045926	0,000184	0,278019	0,001114
Serine	0,027391	0,000046	0,260646	0,000439
Methionine	0,017013	0,000364	0,114023	0,002439
Histidine	0,010768	0,000959	0,069399	0,006180
Tryptophan	0,749563	1,982687	3,670192	9,708108

Table 4.7: Kinetic constants obtained from the best fitting methods and the converted units, used for the simulation with DFBAlab.

4.2 DFBAlab implementation

For the construction of a Dynamic Flux Balance Analysis model, once the kinetic parameters V_{max} and K_m were obtained, the *Dynamic Flux Balance Analysis laboratory* code was implemented, developed by Gomez et. al. [33]. The model used was iMF721, a gene-scale metabolic network model of *Pseudoalteromonas haloplanktis* TAC125 developed by Fondi et. al. [20].

The use of this tool require the installation of COBRA Toolbox v. 3.0 [30] and a linear program (LP) solver, such as Gurobi or CPLEX, the former was installed. The code contains three key scripts: main.m, DRHS.m and RHS.m. In addition, there are two other files: evts.m and SetModel.m, where the latter is the one that converts an SMBL file into a .mat file, that can be used in MATLAB simulations [49], and sets some initial conditions.

Since the DFBAlab code was wrote by Jose A. Gómez [50], in this thesis there are going to be specified only the lines that were changed in order to set the simulation of the genome-scale metabolic network model used.

Set Model

Given that the genome-scale metabolic model iMF721 [20] is in SBML format, the function `readCBModel` is used to transform it into a .mat file, a function of Cobra Toolbox.

Further more, the lower bound of the Schatz medium reactions ($1\,g/l\,KH_2PO_4$, $1\,g/l\,NH_4NO_3$, $10\,g/l\,NaCl$, $0.2\,g/l\,MgSO_4 \times 7\,H_2O$, $0.01\,g/l\,FeSO_4 \times 7\,H_2O$, $0.01\,g/l\,CaCl_2 \times 2\,H_2O$; [51]) are set as infinite (-1000) because they are not suppose to be consumed in the time of the simulation, following the experimental conditions set by Perrin et. al. [24]. The specific salt and its ID reaction are shown on table 4.8, and they conform the `RxnListExchangeSchatz` cell array.

Salt	Reactions ID	Salt	Reactions ID
Mg	$EX\_cpd00254\_e$	K	$EX\_cpd00205\_e$
HO_7P_2	$EX\_cpd00012\_e$	Cl	$EX\_cpd00099\_e$
Na	$EX\_cpd00971\_e$	HO_4P	$EX\_cpd00009\_e$
H	$EX\_cpd00067\_e$	H_4N	$EX\_cpd00013\_e$
Ca	$EX\_cpd00063\_e$	Fe	$EX\_cpd10515\_e$
O_4S	$EX\_cpd00048\_e$	NO_3	$EX\_cpd00209\_e$

Table 4.8: Exchange reactions ID for each mineral in Schatz medium.

On the other hand, the lower bound of the amino acids uptake exchange reactions `EX_reactions` and specially the biomass exchange reaction `'EX_Biomass_e'` are set as 0, because from this point of view, it is seen as a closed system.

```
% Set Model

initCobraToolbox;
TAC125Model = readCbModel('tac125_model.xml');
FBAsolutionAsIs = optimizeCbModel(TAC125Model,'max');
GrowthAllEXOpen = FBAsolutionAsIs.f;
EX_reactions= TAC125Model.rxns(¬cellfun(@isempty, ...
    regexp(TAC125Model.rxns,'^EX_')));
NumberOfEXReactions = length(EX_reactions);
%Setting lb of exchange reactions
TAC125Model = changeRxnBounds(TAC125Model, EX_reactions, 0, 'l'); %-3.65
FBAsolutionNoEX = optimizeCbModel(TAC125Model,'max');
FBAsolutionAllClosed = FBAsolutionNoEX.f;
%Setting lb of Schatz medium exchange ractions
RxnListExchangeSchatz = { 'EX_cpd00254_e', 'EX_cpd00012_e', ...
    'EX_cpd00971_e', 'EX_cpd00067_e', 'EX_cpd00063_e', 'EX_cpd00048_e', ...
    'EX_cpd00205_e', 'EX_cpd00099_e', 'EX_cpd00009_e', 'EX_cpd00013_e', ...
    'EX_cpd10515_e', 'EX_cpd00209_e' };
TAC125Model = changeRxnBounds(TAC125Model, RxnListExchangeSchatz, -1000, ...
    'l');
% Setting lb of Biomass exchange reaction
TAC125Model = changeRxnBounds(TAC125Model, 'EX_Biomass_e', 0, 'l');
save('TAC125Model')
```

Main

This file sets the simulation, so it is important to specify the path of the Function's file (it comes with the DFBAlab folder), and the number of models used for the simulation, which in this case is just one (iMF721, which has to be in .mat format). The `INFO` structure will have all the information needed for the simulation, starting with the default bound.

For the amino acids' uptake exchange reaction it was used the `findRxnIDs` command to find each one of the IDs (Table 4.9), so it could be arrange as the `exID` array and consequently, put it in the `INFO` structure.

Amino acids	Reactions ID	Amino acids	Reactions ID
Glutamate	1133	Alanine	1153
Glutamine	1167	Lysine	1148
Arginine	1163	Glycine	1118
Leucine	1155	Tyrosine	1103
Asparagine	1164	Phenylalanine	1160
Aspartate	1154	Serine	1094
Proline	1107	Methionine	1159
Valine	1162	Histidine	1166
Isoleucine	1158	Tryptophan	1161
Threonine	1169		

Table 4.9: Exchange reactions ID for each amino acid.

```
clear all
addpath('/DFBAlab/Functions');
INFO.nmodel = 1

%Loading model
load TAC125Model.mat
model{1} = TAC125Model;
DB(1) = 2000;
INFO.DB = DB;

% exID array
exID{1} = [1133, 1167, 1163, 1155, 1164, 1154, 1107, 1162, 1158, 1169, ...
    1153, 1148, 1118, 1103, 1160, 1094, 1159, 1166, 1161];
INFO.exID = exID;
```

The next part of the code states the cost vectors, which are going to be maximized. The list includes all the 19 amino acids, and it has three parameters. The first one, *Ci(j).sense* maximization, *Ci(j).rxns* the reaction corresponding to this cost vector, *Ci(j).wts* is the weight of the cost vector, *i.e.* if it adds or not.

```
minim = 1;
maxim = -1;

% Maximize growth
C{1}(1).sense = maxim;
C{1}(1).rxns = [1156];
C{1}(1).wts = [1];
% Maximize Glutamate
C{1}(2).sense = maxim;
C{1}(2).rxns = [1133];
C{1}(2).wts = [1];
% Maximize Glutamine
.
.
.
% Maximize Tryptophan
C{1}(20).sense = maxim;
C{1}(20).rxns = [1161];
C{1}(20).wts = [1];

INFO.C = C;
```

After setting the cost vectors, the initial conditions have to be specified. In this part, the units used are *mM* (*mmol/l*) for all the amino acids, and *g/l* for the biomass. Since they are not used in the same equation given (this time is used the Michaelis-Menten equation), the different units are not an issue. The time interval is specified as well (h).

```
% Initial conditions
% Y1 = Volume (l)
% Y2 = Biomass TAC125 (g/l)
```

```
% Y3 = Glutamate (mmol/l)
% Y4 = Glutamine (mmol/l)
% Y5 = Arginine (mmol/l)
% Y6 = Leucine (mmol/l)
% Y7 = Asparagine (mmol/l)
% Y8 = Aspartate (mmol/l)
% Y9 = Proline (mmol/l)
% Y10 = Valine (mmol/l)
% Y11 = Isoleucine (mmol/l)
% Y12 = Threonine (mmol/l)
% Y13 = Alanine (mmol/l)
% Y14 = Lysine (mmol/l)
% Y15 = Glycine (mmol/l)
% Y16 = Tyrosine (mmol/l)
% Y17 = Phenylalanine (mmol/l)
% Y18 = Serine (mmol/l)
% Y19 = Methionine (mmol/l)
% Y20 = Histidine (mmol/l)
% Y21 = Tryptophan (mmol/l)
% Y22 = Penalty
Y0 = [.1 0.096 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 ...
    0.2 0.2 0.2 0.2 0.2 0];

% Time of simulation (h)
tspan = [0,15];
```

It is important to select the LP solver, which in this case is Gurobi (1), and the LP tolerance, which has to be larger than 10^{-9} [49]. In this case, the tolerance that worked was 10^{-8}.

```
INFO.LPsolver = 1; % CPLEX = 0, Gurobi = 1.
INFO.tol = 1E-8;
```

The information of the fluxes regarding the biomass growth was saved in the *GrowthRate.xls* file, and in the last part of the main script was loaded to be printed. The results of the simulation were also plotted and the colors used correspond to those used in the Perrin *et. al.* [24] work. The green one is for the first cluster of amino acids found, purple for the second, blue for the third and orange for the fourth (Fig. 4.5).

```
%Growth rate data
data_GR = readtable('GrowthRate.xls','PreserveVariableNames',true);
GR = table2array(data_GR(:,1));
T_GR = linspace(0,15,930)';

%Plotting all the results
figure(1)
%Biomass growth:
subplot(2,1,1);
h=plot(T,Y(:,2)/0.74,'-b');
set(gca,'fontsize',14)
ylabel('O.D.');
title('Growth curve','FontSize',18);
%Growth rate:
```

```
        axes('Position',[.7 .64 .18 .2])
        box on
        plot(T_GR,GR,'-b')
        ylabel('Growth Rate (h^-^1)');
        xlabel('Time (h)');
        %Amino acid's uptake:
        subplot(2,1,2);
        h=plot(T,Y(:,3),'--.', T,Y(:,4),'--.', T,Y(:,5),'--.', T,Y(:,6),'--.', ...
            T,Y(:,7),'--.', T,Y(:,8),'--.', T,Y(:,9),'--.', T,Y(:,10),'--.', ...
            T,Y(:,11),'--.', T,Y(:,12),'--.', T,Y(:,13),'--.', T,Y(:,14),'--.', ...
            T,Y(:,15),'--.', T,Y(:,16),'--.', T,Y(:,17),'--.', T,Y(:,18),'--.', ...
            T,Y(:,19),'--.', T,Y(:,20),'--.', T,Y(:,21),'--.', T,Y(:,22),'-.');
        %Coloring
        set(h(1),'linewidth',1.5); set(h(1),'Color',[0.4660 0.6740 0.1880]); %Glu
        set(h(2),'linewidth',1.5); set(h(2),'Color',[0.4660 0.6740 0.1880]); %Gln
        .
        .
        .
        set(h(17),'linewidth',1.5); set(h(17),'Color',[0.8500 0.3250 0.0980]); %Met
        set(h(18),'linewidth',1.5); set(h(18),'Color',[0.8500 0.3250 0.0980]); %His
        set(h(19),'linewidth',1.5); set(h(19),'Color',[0.8500 0.3250 0.0980]); %Trp
        set(h(20),'linewidth',1.5); set(h(20),'Color',[0 0 0]); %Penalty
        legend('Glu (2)','Gln (1)','Arg-','Leu-','Asn (3)','Asp (4)','Pro ...
            (6)','Val-','Ile (5)','Thr (7)','Ala (8)','Lys (11)','Gly (10)','Tyr ...
            (13)','Phe (12)','Ser (9)','Met-','His-','Trp-','Penalty');
        set(gca,'fontsize',14)
        ylabel('Concentration (mM)');
        xlabel('Time (h)','FontSize',15);
        title('Degradation dynamics for 19 amino acids (Michaelis-Menten ...
            eq.)','FontSize',17);
```

DRHS

The DRHS function takes time, the y vector and the INFO structure and returns the right hand side vector of the ODE or DAE system [20]. The values are assign from the y vector, the feed rates and the biomass feed concentrations are set to 0, as well as the mass transfer expressions, since it is a closed system. The molecular weight and the substrate feed concentrations were specified.

```
        function dy = DRHS(t, y, INFO)

        % Y1 = Volume (l)
        % Y2 = Biomass TAC125 (g/l)
        % Y3 = Glutamate (mmol/l)
        % .
        % .
        % .
        % Y21 = Tryptophan (mmol/l)
        % Y22 = Penalty

        %Assign values from y vector
        Vol = y(1);
        X(1) = y(2);
        for i=1:19
        S(i) = y(2+i);
        end
```

```
        %Feed rates
        Fin = 0;
        Fout = 0;

        % Biomass Feed concentrations
        Xfeed(1) = 0;
        Xfeed(2) = 0;

        % Mass transfer expressions
        MT(1) = 0;
        .
        .
        .
        MT(19) = 0;

        %Molecular weight
        MW(1) = 147.13/1000; %Glutamate
        .
        .
        .
        MW(19) = 204.23/1000; %Tryptophan

        %Substrate feed concentrations
        Sfeed(1) = 0;
        .
        .
        .
        Sfeed(19) = 0;
```

The second part of this script consists in the `solveModel` function, and the flux variable is a matrix with rows corresponding to each model and columns corresponding to each cost vector. Therefore, *flux(i,j)* corresponds to the optimal value of cost vector j and model i with the order defined in 'main.m'. In this simulation, i is always equal to 1, because it is done with just one model, iMF721 [20, 49].

The penalty vector contains the objective function values of the LPs. If *penalty(i)*>0, model i corresponds to an infeasible LP. Otherwise, model i is feasible.

```
        %% Update bounds and solve for fluxes
        [flux,penalty] = solveModel(t,y,INFO);

        %Fluxes
        v(1,1) = flux(1,1); %Biomass
        v(1,2) = flux(1,2); %Glutamate
        .
        .
        .
        v(1,20) = flux(1,20); %Triptophan

        %% Dynamics
        dy = zeros(22,1);    %A column vector
        dy(1) = Fin-Fout;   %Volume
        dy(2) = flux(1,1)*X(1) + (Xfeed(1)*Fin - X(1)*Fout)/y(1);   %Biomass
        for i = 1:19  %For each amino acid
```

```
dy(i+2) = v(1,i+1)*MW(i)*X(1) + MT(i) + (Sfeed(i)*Fin - S(i)*Fout)/y(1);
end
dy(22) = penalty(1);
end
```

RHS

Finally, the RHS function takes time, the `y` vector and the `INFO` structure as inputs and returns two matrices containing the upper and lower bounds (*ub* and *lb*, respectively) for the fluxes specified in the *exID* cell in 'main.m'. This *ub* and *lb* are functions of time and states (`y` values).

If the *lb* is negative, then it is defined as 0, otherwise it is defined by the Michaelis-Menten equation (4.2), where the kinetic constants V_{max} and K_m are in *mmol/h* and *mmol/l* instead of *g/h* and *g/l*, respectively. The *ub* is always set as 0.

$$\frac{dS}{dt} = \frac{-V_{max} * S}{(K_m + S)} \tag{4.2}$$

```
function [lb,ub] = RHS( t,y,INFO )

% Y1 = Volume (l)
% Y2 = Biomass TAC125 (g/l)
% Y3 = Glutamate (mmol/l)
% .
% .
% .
% Y21 = Tryptophan (mmol/l)
% Y22 = Penalty

% Vmax in mmol/h e Km in mmol/l

% Glutamate
if (y(3)<0)
lb(1,1) = 0;
else
lb(1,1) = -0.523415.*y(3)./(0.027292+y(3));
end
ub(1,1) = 0;

%Glutamine
.
.
.

% Tryptophan
if (y(21)<0)
lb(1,19) = 0;
else
lb(1,19) = -3.670192.*y(21)./(9.708108+y(21));
end
ub(1,19) = 0;

end
```

The simulation performed by DFBAlab showed that the amino acids' behaviour was similar to the one observed experimentally. The order of the clusters of amino acids was the same except for isoleucine, which belongs to the third cluster (blue) and its concentration diminishes first than that of proline, which belongs to the second cluster (Table 4.10).

It is important to notice that the model is not sensible to arginine (first cluster), leucine (second cluster), valine (third cluster), and all the amino acids belonging to the fourth cluster: methionine, histidine and tryptophan. However, neither of the last three were completely used as a carbon source *in vitro* [24].

On the other hand, there is one behaviour observed with the bacterial growth *in vitro* that is not present in the simulation: the two lags of the growth rate, and infact, the *in silico* growth curve seems to be a smooth one (Fig. 4.5a and 4.6a).

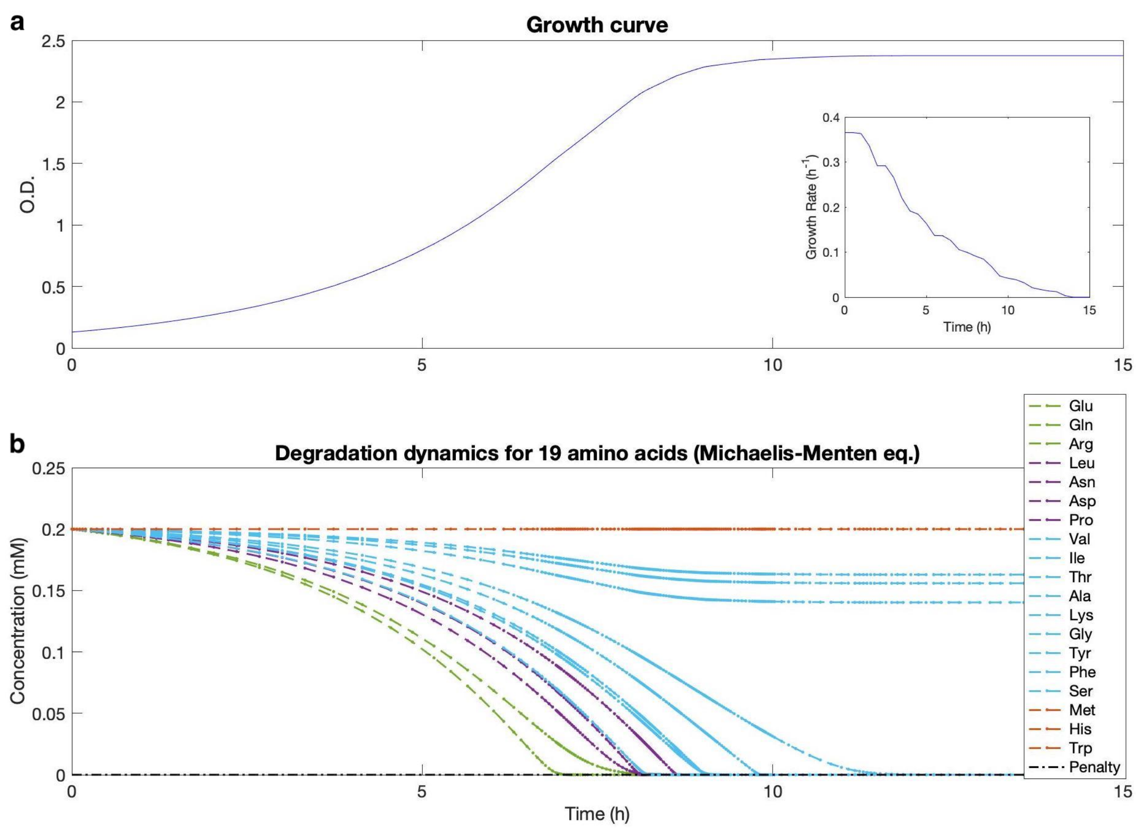

Figure 4.5: **a.** Biomass growth curve in optical density, and growth rate. **b.** Amino acids uptake by *Pseudoalteromonas haloplanktis*.

As reported in previous studies of *Pseudoalteromonas haloplanktis* TAC125, glutamate appeared to be the most important amino acid controlling the growth. It was observed that it was

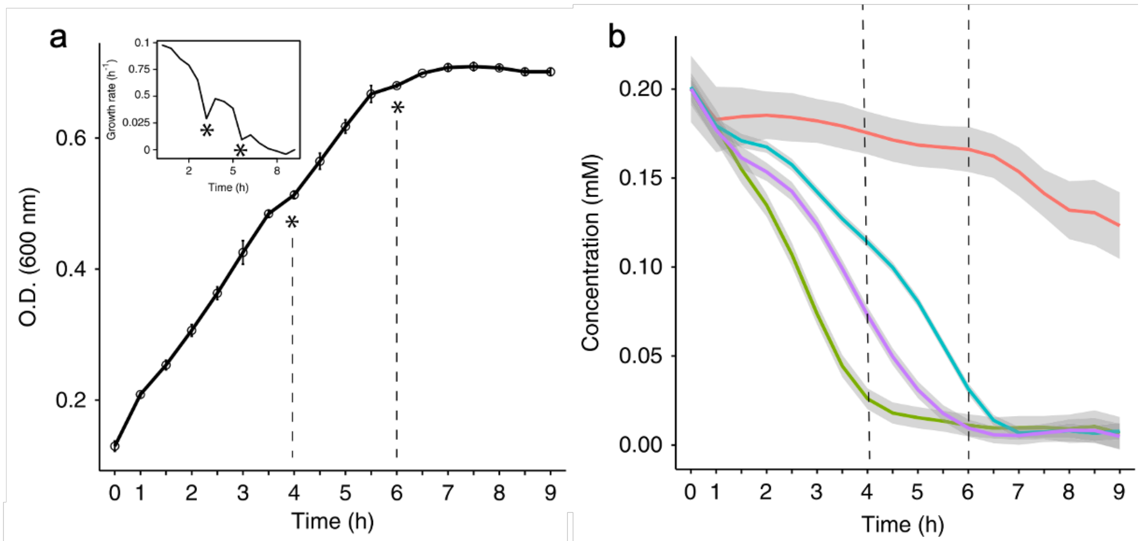

Figure 4.6: Experimental data obtained from Perrin *et. al.* [24]. **a** Biomass growth curve in optical density, and growth rate. **b** Amino acids uptake by *Pseudoalteromonas haloplanktis*, green, violet, blue and orange correspond to cluster one, two, three and four, respectively.

consumed with the highest rate compared to all the other analyzed amino acids, and when it was exhausted, the growth declined immediately [9]. This growth decline is what can be observed in Perrin's work (Fig. 4.6a) but not in the *in silico* experiment (Fig. 4.5a). The results of this thesis agree with the preference of *Ph*TAC125 for glutamate (Table 4.10), following glutamine, which was not included in Wilmes' study [9]. However, the dynamic model might not be describing the high uptake rate that is shown *in vitro*, hence further work needs to be done to address this problem.

Order	Amino acids	Time (h)	Order	Amino acids	Time (h)
1	Glutamine	6,94	11	Lysine	-
2	Glutamate	8,00	12	Phenylalanine	-
3	Asparagine	8,07	13	Tyrosine	-
4	Aspartate	8,08	-	Arginine	-
5	Isoleucine	8,17	-	Leucine	-
6	Proline	8,60	-	Valine	-
7	Threonine	9,00	-	Methionine	-
8	Alanine	9,07	-	Histidine	-
9	Serine	9,80	-	Tryptophan	-
10	Glycine	11,50			

Table 4.10: Order of amino acids uptake and the specific time in which it goes to zero.

4.3 Application of the dynamic model

Glutamate and Gluconate

To study how *Pseudoalteromonas haloplanktis* TAC125 uses as carbon sources gluconate and glutamate in Schatz medium, the utilization rate was experimentally calculated. These results were used to obtain the kinetic constants (V_{max} and K_m) that could describe glutamate's and gluconate's behaviour the better, in order to perform a simulation with DFBAlab.

The three methods used (*lsqcurvefit, SIMPSA* and *SIMPLEXL*) gave different results after being compiled three times, and those from the third one were take into account (Table 4.11). Then, they were compared using the Akaike's Information Criterion (Table 4.12), resulting that the *lsqcurvefit* method lost the less information for glutamate, and the *SIMPLEXL* method for gluconate (Fig. 4.7). The molecular weight of each acid was used in order to obtain the units needed by DFBAlab code (MW$_{glu}$=147,13 g/mol and MW$_{gluc}$=196,16 g/mol), and the initial values of the biomass and the acids where those used for the experiment, so the simulation could be properly analyzed. The kinetic constants used for the code were as it follows:

For Glutamate: $V_{max} = 0,921517$ mmol/h and $K_m = 0,000070$ mM.

For Gluconate: $V_{max} = 1,215330$ mmol/h and $K_m = 0,000333$ mM.

Fitting method	**Glutamate** $\mathbf{V}_{max}$ **(g/h)**	**Glutamate** $\mathbf{K}_m$ **(g/l)**	**Gluconate** $\mathbf{V}_{max}$ **(g/h)**	**Gluconate** $\mathbf{K}_m$ **(g/l)**
lsqcurvefit	0,135583	0,000010	0,242051	0,000018
SIMPSA	0,098171	0,000021	0,238394	0,000000
SIMPLEXL	0,098308	0,006553	0,238399	0,000065

Table 4.11: Kinetic constants obtained using *lsqcurvefit, SIMPSA* and *SIMPLEXL* as the three fitting methods.

Acids	**lsqcurvefit**	**SIMPSA**	**SIMPLEXL**
Glutamate	**-4,151601**	-3,603616	-3,603254
Gluconate	-1,962716	-1,723312	**-1,994047**

Table 4.12: Results of Akaike's Information Criterion when comparing the three fitting methods. Those who lose the less information are shown in bold.

Once the kinetic constants were used to run the simulation, the biomass growth was 18% higher than the one experimentally obained (1,83 O.D. and 1,55 O.D., respectively; see table 4.13 and fig. 4.8). However, the degradation dynamics of both glutamate and gluconate were slower than expected. Experimentally, *Ph*TAC125 consumed 2,6 mM of glutamate and 5,0 mM of gluconate, while in the simulation, it consumed 0,8 mM of glutamate and 1,5 mM of gluconate. It took 12 hours for the bacteria to consume the expected amount of glutamate (2,6 mM) and 12,3 hours for gluconate. These values are indicated in figure 4.8.

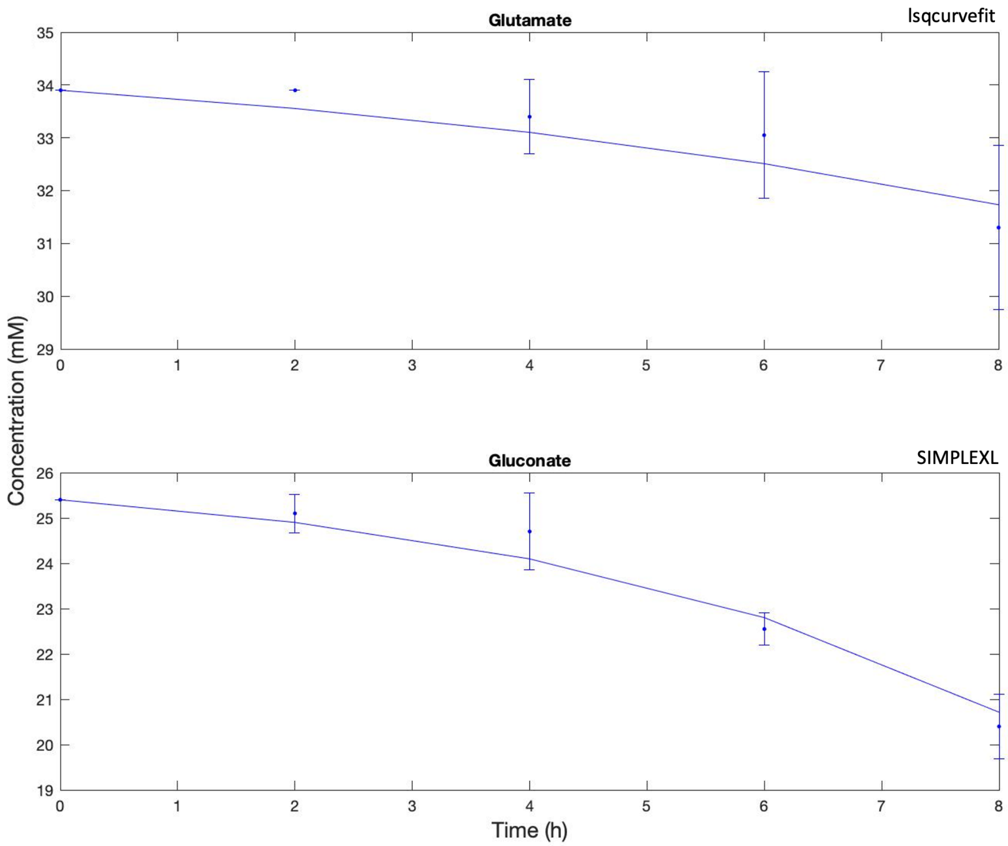

Figure 4.7: Glutamate and gluconate uptake by *Pseudoalteromonas haloplanktis.*

Recombinant protein: CDKL5

Pseudoalteromonas haloplanktis TAC125 was experimentally used as an expression platform to assess the production of a recombinant protein (CDKL5). Studies in cell and animal models have shown that the X-linked serine/threonine kinase, cyclin-dependent kinase-like 5, is important for neurite outgrowth and dendritic spine development as well as for functional neuronal plasticity. It has also been implicated as an important regulator of cellular responses to oxidative stress [53].

Hence, a CDKL5 deficiency, caused by dominantly acting loss-of-function variants in the X-linked gene, could lead to neurological diseases such as early-onset epileptic encephalopathies [52] [54].

The expression of CDKL5 was done through the use of a plasmid and IPTG (isopropyl-β-D-thiogalactopyranoside) as an inductor of the *lac* operon's promoter, which triggered the transcription of the wanted protein. The experiment had two different setups, one where IPTG was used to perform the induction of the protein, an one were it was not present, being the wild type.

In order to perform a simulation of this experimental setup, the SetModel.m script was updated with the reaction of the protein with the `addReaction` command. This was done following the stoichiometric equation that corresponds to the protein production. Then, it was added as an exchange reaction, and the growth was set. The medium was as in the previous experiments (Schatz), with the addition of gluconate and glutamate, but since the concentrations were much higher than they were in the previous experiment (51 mM and 68 mM, respectively), it was not necessary setting a limit before eight hours. The reactions ID were `EX_cpd00222_e` for gluconate, `EX_cpd00223_e` for glutamate and 1326 for the protein.

```
% Set Model

% initCobraToolbox;
changeCobraSolver('mosek');
TAC125Model = readCbModel('tac125_model.xml');
%TAC125Model = checkCobraModelUnique(TAC125Model, 'T')
FBAsolutionAsIs = optimizeCbModel(TAC125Model,'max');
GrowthAllEXOpen = FBAsolutionAsIs.f;
%% Close ex reactions
EX_reactions= TAC125Model.rxns(¬cellfun(@isempty, ...
    regexp(TAC125Model.rxns,'^EX_')));
NumberOfEXReactions = length(EX_reactions);
TAC125Model = changeRxnBounds(TAC125Model, EX_reactions, 0, 'l');
FBAsolutionNoEX = optimizeCbModel(TAC125Model,'max');
FBAsolutionAllClosed = FBAsolutionNoEX.f;
%% Medium
RxnListExchangeSchatz = { 'EX_cpd00254_e' , 'EX_cpd00012_e' , ...
    'EX_cpd00971_e' , 'EX_cpd00067_e' , 'EX_cpd00063_e' , 'EX_cpd00048_e' ...
    , 'EX_cpd00205_e' , 'EX_cpd00099_e' , 'EX_cpd00009_e' , ...
    'EX_cpd00013_e' , 'EX_cpd10515_e', 'EX_cpd00209_e' };
TAC125Model = changeRxnBounds(TAC125Model, RxnListExchangeSchatz, -1000, ...
    'l');
TAC125Model = changeRxnBounds(TAC125Model, 'EX_Biomass_e', 0, 'l');
RxnExchangeGlcn_Glu = {'EX_cpd00222_e' , 'EX_cpd00023_e'};
TAC125Model = changeRxnBounds(TAC125Model, RxnExchangeGlcn_Glu, ...
    [-0.665937670858393  -0.346287588846364], 'l');
%% Protein Reaction
TAC125Model = addReaction(TAC125Model, 'CDKL5', ' 0.426 cpd00052[c] + ...
    0.574 ...
cpd00062[c] + 59 cpd00035[c] + 6 cpd00084[c] + 63 cpd00041[c] + ...
76 cpd00023[c] + 32 cpd00066[c] + 72 cpd00033[c] + 49 cpd00119[c] + ...
39 cpd00322[c] + 84 cpd00039[c] + 101 cpd00107[c] + 20 cpd00060[c] +  ...
59 cpd00132[c] + 80 cpd00129[c] + 53 cpd00053[c] + 76 cpd00051[c] +  ...
140 cpd00054[c] + 56 cpd00161[c] + 41 cpd00156[c] + 6 cpd00065[c] +  ...
32 cpd00069[c] + 2288.574 cpd00002[c] + 2286.426 cpd00038[c] ...
```

```
        -> 0.01 cdkl5[c] + 2288 cpd00018[c] + 2286 cpd00031[c] + 4572 cpd00009[c]');
        % Optimize 'EX_cdkl5[c]' for protein production
        TAC125Model = addExchangeRxn(TAC125Model, 'cdkl5[c]' , 0,  1000);
        % for cdkl5 prodution,set to biomass reaction the follows constraint
        MAXgrowth_rate  =optimizeCbModel(TAC125Model);
        MAXgrowth_rate = MAXgrowth_rate.f;
        TAC125Model = changeRxnBounds(TAC125Model, 'RXNbiomass', ...
            MAXgrowth_rate*0.74, 'b');

        save('TAC125Model')
```

There were two main changes in the main.m and RHS.m scripts in order to produce the protein:

- In the main.m script: the order of the lexicographic optimization was changed, letting the simulation optimized the protein production first, and then the biomass growth; and the initial concentration of the protein production was not zero because given that the DFBAlab code had the production equation multiplied by it, it would not show any protein production, thus the initial concentration was set to 10^{-5}.
- In the RHS.m script: the lower bound for the protein optimization was set to zero, while the upper bound to infinite, so it could grow freely.

Finally, the DFBAlab code was used to optimize only the biomass, so it could be compared with the wild type experiment, *i.e.* when the inductor was not used and *Pseudoalteromonas haloplanktis* grew without producing the recombinant protein. This way, the growth difference could be analyzed and compared with the experimental data (Fig. 4.9).

It can be noticed that the biomass growth is lower when the bacteria is producing the protein, it grows 23,4% less when the bacteria produces the recombinant protein CDKL5 (Fig. 4.9a). It is interesting to see that in both cases (IPTG and wild type) the simulation gave lower values than expected, which were up to 3,5 O.D. for the wild type and 2,6 O.D. when producing the protein (table 4.14), however the behavior of the decrease in growth when the protein is being produced was maintained, being 25,7% for experimental data. When talking about the protein production, there was a 35% of increase, which corresponds to the production rate (Fig. 4.9b).

Once again, the dynamic model provided a qualitative description of the experiments.

Time (h)	Biomass (O.D.)	Glutamate (mM)	Gluconate (mM)
0	0,22	33,9	25,4
2	0,34	33,9	25,1
4	0,48	33,4	24,7
6	1,02	33,1	22,6
8	1,55	31,3	20,4

Table 4.13: Experimental results of biomass growth and the uptake of glutamate and gluconate.

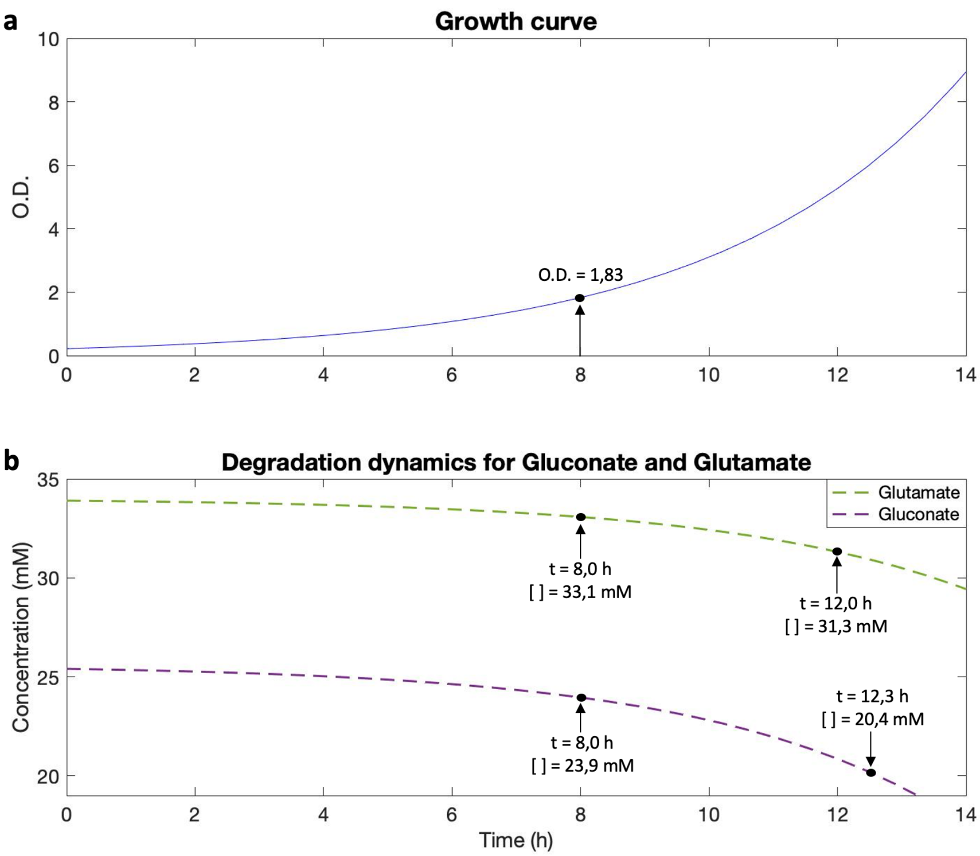

Figure 4.8: **a.** Biomass growth curve, the optical density predicted at hour eight is indicated. **b.** Glutamate and gluconate uptake by *Pseudoalteromonas haloplanktis*. For each acid, it is indicated the concentration after eight hours, and the time it took to reach the final concentrations experimentally obtained (see table 4.13).

Time (h)	Experiment		Simulation	
	WT	**CDKL5**	**WT**	**CDKL5**
0	0,9	0,9	0,9	0,9
4	2,0	1,8	1,6	1,4
8	3,5	2,6	2,6	1,9

Table 4.14: Experimental and simulation results of biomass growth (optical density) for the wild type and when the protein is being produced (WT = wild type, CDKL5 = inductor of transcription, IPTG, is present).

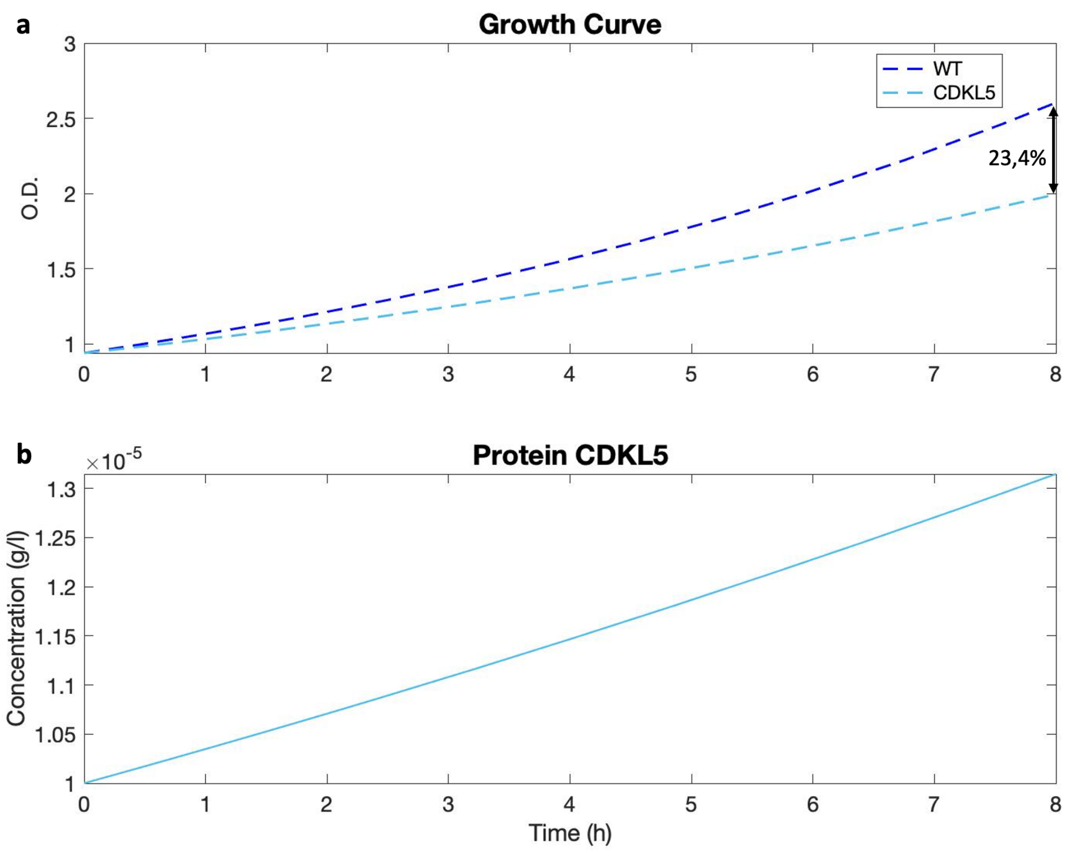

Figure 4.9: **a.** Biomass growth curve for *P. haloplanktis* wild type (dark blue) and producing the protein CDKL5 (light blue), the percentage of growth difference at hour eight is indicated. **b.** Protein production.

Chapter 5

Conclusions

The realization of a dynamic genome-scale metabolic model for *Pseudoalteromonas haloplanktis* TAC125 required several steps in order to obtain the best kinetic constants for the model to work correctly.

In this thesis, a broad and reliable method of fitting data was developed using several fitting algorithms and their subsequently comparison to evaluate how well the information experimentally obtained was conserved and described. From this analysis, it cannot be chosen just one good method to use. Instead, the use of at least two of them and then do the comparison is encouraged. It is worth to say that after testing four statistical methods, Akaike's Information Criterion (AIC) was the best option to carry out the foretold comparison.

Furthermore, the DFBAlab code was adapted to be used for the study of the dynamics of *P. haloplanktis* growing in a complex medium. This model gave results that are qualitatively similar to the experimental observations made by Perrin et al. [24], however, these *in silico* experiments gave the possibility to analyzed the behavior of each amino acid separately, instead of by clusters. Yet, more work has to be done in order to explain why this model did not show the lags in growth rate after the exhaustion of glutamine and glutamate (cluster one), and consecutively the one of asparagine, aspartate, isoleucine and proline (cluster two-three).

Additionally, when the model was applied for the study of the production of recombinant proteins, in particular CDKL5, the first step was to analyze the dynamics of the bacteria growing in a different medium, composed only by glutamate and gluconate instead of 19 amino acids. The model described the system with some shortcomings, giving a broad qualitative perspective.

The second step of the study of the production of CDKL5 gave once again qualitatively similar results to those of the experiment. For example, it could be well determined the reduction of the biomass growth when the bacteria is producing the recombinant protein. Still, the model was not able to show how the production of the protein is being affected by the decrease of the biomass, this is due to the equation used by the model that describe it, since it is based only in the production rate previously calculated. The predictions could be significantly improved if the amount of experimental data were larger.

All in all, it can be concluded that the realization of the dynamic genome-scale metabolic model was succesfully performed even when the results are mostly qualitative. The model is able to describe the metabolic dynamics of *P. haloplanktis*, qualitatively matching previous experimental data. Nonetheless, more work needs to be done in order to increase its robustness.

Bibliography

[1] Emanuele Bosi, Marco Fondi, Valerio Orlandini, Elena Perrin, Isabel Maida, Donatella de Pascale, Maria Luisa Tutino, Ermenegilda Parrilli, Angelina Lo Giudice, Alain Filloux, and Renato Fani. The pangenome of (Antarctic) Pseudoalteromonas bacteria: Evolutionary and functional insights. *BMC Genomics*, 18(1):1–18, 2017.

[2] Claudine Médigue, Evelyne Krin, Géraldine Pascal, Valérie Barbe, Andreas Bernsel, Philippe N. Bertin, Frankie Cheung, Stéphane Cruveiller, Salvino D'Amico, Angela Duilio, Gang Fang, Georges Feller, Christine Ho, Sophie Mangenot, Gennaro Marino, Johan Nilsson, Ermenegilda Parrilli, Eduardo P.C. Rocha, Zoé Rouy, Agnieszka Sekowska, Maria Luisa Tutino, David Vallenet, Gunnar Von Heijne, and Antoine Danchin. Coping with cold: The genome of the versatile marine Antarctica bacterium Pseudoalteromonas haloplanktis TAC125. *Genome Research*, 15(10):1325–1335, 2005.

[3] Florence Piette, Salvino D'Amico, Caroline Struvay, Gabriel Mazzucchelli, Jenny Renaut, Maria Luisa Tutino, Antoine Danchin, Pierre Leprince, and Georges Feller. Proteomics of life at low temperatures: Trigger factor is the primary chaperone in the Antarctic bacterium Pseudoalteromonas haloplanktis TAC125. *Molecular Microbiology*, 76(1):120–132, 2010.

[4] Salvino D'Amico, Tony Collins, Jean Claude Marx, Georges Feller, and Charles Gerday. Psychrophilic microorganisms: Challenges for life. *EMBO Reports*, 7(4):385–389, 2006.

[5] Sangita Phadtare. Recent developments in bacterial cold-shock response. *Current Issues in Molecular Biology*, 6(2):125–136, 2004.

[6] Florence Piette, Salvino D'Amico, Gabriel Mazzucchelli, Antoine Danchin, Pierre Leprince, and Georges Feller. Life in the cold: A proteomic study of cold-repressed proteins in the antarctic bacterium Pseudoalteromonas haloplanktis TAC125. *Applied and Environmental Microbiology*, 77(11):3881–3883, 2011.

[7] Glenn C. Johns and George N. Somero. Evolutionary Convergence in Adaptation of Proteins to Temperature: A 4-Lactate Dehydrogenases of Pacific Damselfishes (Chromis spp.). *Molecular Biology and Evolution*, 21(2):314–320, 2004.

[8] Peter Canning, Kwangjin Park, João Gonçalves, Chunmei Li, Conor J. Howard, Timothy D. Sharpe, Liam J. Holt, Laurence Pelletier, Alex N. Bullock, and Michel R. Leroux. CDKL Family Kinases Have Evolved Distinct Structural Features and Ciliary Function. *Cell Reports*, 22(4):885–894, 2018.

[9] Boris Wilmes, Angelika Hartung, Michael Lalk, Manuel Liebeke, Thomas Schweder, and Peter Neubauer. Fed-batch process for the psychrotolerant marine bacterium Pseudoalteromonas haloplanktis. *Microbial Cell Factories*, 9:1–9, 2010.

[10] F. Sannino, M. Giuliani, U. Salvatore, G. A. Apuzzo, D. de Pascale, R. Fani, M. Fondi, G. Marino, M. L. Tutino, and E. Parrilli. A novel synthetic medium and expression system for subzero growth and recombinant protein production in Pseudoalteromonas haloplanktis TAC125. *Applied Microbiology and Biotechnology*, 101(2):725–734, 2017.

[11] Baart G.J.E. and Martens D.E. Genome-scale metabolic models: Reconstruction and analysis. *Methods in Molecular Biology (Methods and Protocols)*, 799, 2012.

[12] Edward J. O'Brien, Joshua A. Lerman, Roger L. Chang, Daniel R. Hyduke, and Bernhard Palsson. Genome-scale models of metabolism and gene expression extend and refine growth phenotype prediction. *Molecular Systems Biology*, 9(693), 2013.

[13] Changdai Gu, Gi Bae Kim, Won Jun Kim, Hyun Uk Kim, and Sang Yup Lee. Current status and applications of genome-scale metabolic models. *Genome Biology*, 20(1):1–18, 2019.

[14] Matthew A. Oberhardt, Bernhard Palsson, and Jason A. Papin. Applications of genome-scale metabolic reconstructions. *Molecular Systems Biology*, 5(320):1–15, 2009.

[15] Edward J. O'Brien, Jonathan M. Monk, and Bernhard O. Palsson. Using genome-scale models to predict biological capabilities. *Cell*, 161(5):971–987, 2015.

[16] Cheng Zhang and Qiang Hua. Applications of genome-scale metabolic models in biotechnology and systems medicine. *Frontiers in Physiology*, 6(JAN):1–8, 2016.

[17] Aarash Bordbar, Jonathan M. Monk, Zachary A. King, and Bernhard O. Palsson. Constraint-based models predict metabolic and associated cellular functions. *Nature Reviews Genetics*, 15(2):107–120, 2014.

[18] Edward J. O'Brien and Bernhard O. Palsson. Computing the functional proteome: Recent progress and future prospects for genome-scale models. *Current Opinion in Biotechnology*, 34:125–134, 2015.

[19] Jeremy S. Edwards and Bernhard O. Palsson. Systems properties of the Haemophilus influenzae Rd metabolic genotype. *Journal of Biological Chemistry*, 274(25):17410–17416, 1999.

[20] Marco Fondi, Isabel Maida, Elena Perrin, Alessandra Mellera, Stefano Mocali, Ermenegilda Parrilli, Maria Luisa Tutino, Pietro Liò, and Renato Fani. Genome-scale metabolic reconstruction and constraint-based modelling of the Antarctic bacterium Pseudoalteromonas haloplanktisTAC125. *Environmental Microbiology*, 17(3):751–766, 2015.

[21] Marco Fondi, Emanuele Bosi, Luana Presta, Diletta Natoli, and Renato Fani. Modelling microbial metabolic rewiring during growth in a complex medium. *BMC Genomics*, 17(1):1–17, 2016.

[22] Mukund Thattai and Boris I. Shraiman. Metabolic switching in the sugar phosphotransferase system of Escherichia coli. *Biophysical Journal*, 85(2):744–754, 2003.

[23] J Monod. The Growth of Bacterial Cultures. *Annual Review of Microbiology*, 3(1):371–394, 1949.

[24] Elena Perrin, Veronica Ghini, Michele Giovannini, Francesca Di Patti, Barbara Cardazzo, Lisa Carraro, Camilla Fagorzi, Paola Turano, Renato Fani, and Marco Fondi. Diauxie and co-utilization of carbon sources can coexist during bacterial growth in nutritionally complex environments. *Nature Communications*, 11(1):1–16, 2020.

[25] Won Jun Kim, Hyun Uk Kim, and Sang Yup Lee. Current state and applications of microbial genome-scale metabolic models. *Current Opinion in Systems Biology*, 2:10–18, 2017.

[26] Nathan E. Lewis, Harish Nagarajan, and Bernhard O. Palsson. Constraining the metabolic genotype-phenotype relationship using a phylogeny of in silico methods. *Nature Reviews Microbiology*, 10(4):291–305, 2012.

[27] Jeffrey D. Orth, Ines Thiele, and Bernhard O. Palsson. What is flux balance analysis? *Nature Biotechnology*, 28(3):245–248, 2010.

[28] Huili Yuan, C. Y.Maurice Cheung, Peter A.J. Hilbers, and Natal A.W. van Riel. Flux balance analysis of plant metabolism: The effect of biomass composition and model structure on model predictions. *Frontiers in Plant Science*, 7(APR2016):1–13, 2016.

[29] Karthik Raman and Nagasuma Chandra. Flux balance analysis of biological systems: Applications and challenges. *Briefings in Bioinformatics*, 10(4):435–449, 2009.

[30] L. Heirendt, S. Arreckx, T. Pfau, S. N. Mendoza, A. Richelle, A. Heinken, H. S. Haraldsdóttir, J. Wachowiak, S. M. Keating, V. Vlasov, S. Magnusdóttir, C. Y. Ng, G. Preciat, A. Žagare, S. H.J. Chan, M. K. Aurich, C. M. Clancy, J. Modamio, J. T. Sauls, A. Noronha, A. Bordbar, B. Cousins, D. C. El Assal, L. V. Valcarcel, I. Apaolaza, S. Ghaderi, M. Ahookhosh, M. Ben Guebila, A. Kostromins, N. Sompairac, H. M. Le, D. Ma, Y. Sun, L. Wang, J. T. Yurkovich, M. A.P. Oliveira, P. T. Vuong, L. P. El Assal, I. Kuperstein, A. Zinovyev, H. S. Hinton, W. A. Bryant, F. J. Aragón Artacho, F. J. Planes, E. Stalidzans, A. Maass, S. Vempala, M. Hucka, M. A. Saunders, C. D. Maranas, N. E. Lewis, T. Sauter, B. Palsson, I. Thiele, and R. M.T. Fleming. Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. *Nature Protocols*, 14(3):639–702, 2019.

[31] Yuki Kuriya and Michihiro Araki. Dynamic flux balance analysis to evaluate the strain production performance on shikimic acid production in Escherichia coli. *Metabolites*, 10(5), 2020.

[32] Radhakrishnan Mahadevan, Jeremy S Edwards, and Francis J Doyle. Dynamic flux balance analysis of diauxic growth. *Biophysical Journal*, 83(3):1331–1340, 2002.

[33] Jose A. Gomez, Kai Höffner, and Paul I. Barton. DFBAlab: A fast and reliable MATLAB code for dynamic flux balance analysis. *BMC Bioinformatics*, 15(1):1–10, 2014.

[34] Antonella Succurro, Daniel Segrè, and Oliver Ebenhöh. Emergent sub-population behavior uncovered with a community dynamic metabolic model of Escherichia coli diauxic growth. *bioRxiv*, 4(1):1–16, 2019.

[35] Leandro H. Watanabe, Matthias König, and Chris J. Myers. Dynamic Flux Balance Analysis Models in SBML. *bioRxiv*, pages 1–8, 2018.

[36] Jósé R. Valverde, Sonia Gullón, Clara A. García-Herrero, Iván Campoy, and Rafael P. Mellado. Dynamic metabolic modelling of overproduced protein secretion in Streptomyces lividans using adaptive DFBA. *BMC Microbiology*, 19(1):1–13, 2019.

[37] IBM ILOG CPLEX Optimizer. User's manual for cplex.

[38] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[39] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.

[40] K. Höffner, S. M. Harwood, and P. I. Barton. A reliable simulator for dynamic flux balance analysis. *Biotechnology and Bioengineering*, 110(3):792–802, 2013.

[41] Jose Alberto Gomez and Prof Paul I Barton. Dynamic Flux Balance Analysis using DFBAlab Authors : Process Systems Engineering Laboratory 77 Massachusetts Avenue Dynamic Flux Balance Analysis using DFBAlab Abstract. Technical report, Process Systems Engineering Laboratory - MIT, 2014.

[42] MATLAB. *version 9.8.0 (R2020a)*. The MathWorks Inc., Natick, Massachusetts, 2020.

[43] Python Language Reference. *version 3.9.1*. The Python Software Foundation, http://www.python.org, 2020.

[44] S. Simkins and M. Alexander. Models for mineralization kinetics with the variables of substrate concentration and population density. *Applied and environmental microbiology*, 47(6):1299–1306, 1984.

[45] M. F. Cardoso, R. L. Salcedo, S. Feyo De Azevedo, and D. Barbosa. A simulated annealing approach to the solution of minlp problems. *Computers and Chemical Engineering*, 21(12):1349–1364, 1997.

[46] George B. Dantzig. *Origins of the Simplex Method*, page 141–151. Association for Computing Machinery, New York, NY, USA, 1990.

[47] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[48] Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1998.

[49] Jose A. Gomez and Paul I. Barton. *Dynamic Flux Balance Analysis using DFBAlab*, pages 353–370. Springer New York, New York, NY, 2018.

[50] J. A. Gomez. *Simulation, Sensitivity Analysis, and Optimization of Bioprocesses using Dynamic Flux Balance Analysis*. PhD thesis, Massachusetts Institute of Technology, 2018.

[51] Albert Schatz and Carlton Bovell. Growth and hydrogenase activity of a new bacterium, hydrogenomonas facilis. *Journal of Bacteriology*, 63(1):87–98, 1952.

[52] Charlotte Kilstrup-Nielsen, Laura Rusconi, Paolo La Montanara, Dalila Ciceri, Anna Bergo, Francesco Bedogni, and Nicoletta Landsberger. What we know and would like to know about CDKL5 and its involvement in epileptic encephalopathy. *Neural Plasticity*, 2012, 2012.

[53] Ralph D. Hector, Owen Dando, Nicoletta Landsberger, Charlotte Kilstrup-Nielsen, Peter C. Kind, Mark E.S. Bailey, and Stuart R.Cobb Cobb. Characterisation of CDKL5 Transcript Isoforms in Human and Mouse. *PLoS ONE*, 11(6):1–22, 2016.

[54] Ralph D. Hector, Vera M. Kalscheuer, Friederike Hennig, Helen Leonard, Jenny Downs, Angus Clarke, Tim A. Benke, Judith Armstrong, Mercedes Pineda, Mark E.S. Bailey, and Stuart R. Cobb. CDKL5 variants . *Neurology Genetics*, 3(6):e200, 2017.

Ringraziamenti

Vorrei ringraziare Marco Fondi perché è stato sempre molto disponibile con me prima ancora che mi iscrivessi alla magistrale. Grazie a lui e a Alessio Mengoni, che mi hanno suggerito di iscrivermi a questa università, ho avuto la fortuna di seguire questo bellissimo percorso di studi. Grazie ancora per avermi mostrato il mondo della bioinformatica e della biologia dei sistemi, discipline delle quali mi sono completamente innamorata. Grazie alla mia correlatrice, Elena Perrin. Grazie ai miei compagni di laboratorio Chris, Stefano, Gabriel e Iacopo per le birre bevute e la cioccolata mangiata in lab.

Vorrei ringraziare anche tutti i miei amici italiani. A Francesco, per averci fatto sentire a casa sin dal primo giorno. A Mery, per essere stata come una sorella, grazie davvero. A Baccio, per tutte quelle serate a base di pasta al pesto, nachos e birre. A Elettra, Elisa, Tommi, Nicco, Bea, Zern e Giulia. A Caterina per tutto l'aiuto che mi ha datto, praticamente è stata la mia guida. A Matteino, per tutte quelle chiacchierate senza senso e la musica condivisa.

And last but not least, vorrei ringraziare i miei bambini: Giusy, Benedetta, Paolo, Nicole, Ilaria, Chicca, Jessica, Roberta, Dario, Lore Paci e, ovviamente, la fantomatica Queen, per tutti gli amari che non erano mai gli ultimi, per le serate nelle quali siamo stati rapiti e le colazioni passate a raccontarci i nostri sogni, per tutte le mozzarelle e biscotti all'amarena che mi hanno portata al cielo, per aver organizzato il nostro matrimonio vero verissimo e per tutte le risate che abbiamo condiviso. Siete fantastici.

Agradecimientos

A ti Santiago, te agradezco todo, por siempre creer en mi, por apoyarme y porque aún en los momentos más difíciles, siempre me hiciste ver que había una salida. Te amo.

Quiero agradecer a mis padres, Sol y Emilio, que a pesar de mis locuras, siempre han estado ahí para apoyarme y con todo el amor del mundo, sin ustedes, mi vida no sería igual. Agradezco también a Rosa, quien con el tiempo se ha convertido en una segunda madre, dándome todo su apoyo y amor. Les agradezco infinitamente haber estado muy cerca de mi a pesar de la distancia.

Gracias a Max, por todas las pláticas, visitas y viajes que hemos hecho en estos años, además de proporcionar la playlist para escribir la tesis. Ya sabes que eres como un hermano para mi. Gracias a Jano por todas esas llamadas y por escuchar mis angustias, y obviamente a mi Negrito. Gracias a Sofía por todo el apoyo y las risas que me ha dado, soy la más feliz de habernos reencontrado. Gracias a Viri, Rafa y por supuesto, Megara, han sido un verdadero apoyo en momentos difíciles y muy divertidos en momentos tranquilos. Gracias a Luis, Jero, Martín, Jerry y Natalia por esas cotorreadas de cada semana y llevarme a otros mundos. Gracias a JuanFris, se te extraña seriamente. Gracias a Hecti y Ponchi que siempre han estado presentes y que me traen muchísima nostalgia. Gracias a Paco, Carlos, Anabela, Poncho y Lulú por su apoyo incondicional. Gracias a la abuela Rosa, quien me adoptó como su nieta y me dio todo el amor. Te extrañaremos siempre.

Quiero agradecer a mis hermanos, Ale y Bebé, quienes me ofrecieron todo su amor y sus ánimos para seguir adelante a pesar de las circunstancias.

A mi hermano Emilio, con todo mi corazón, sé que nos volveremos a ver.