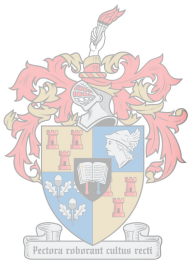
A systematic investigation into the quantitative effect of pH changes on the upper glycolytic enzymes of *Escherichia coli* and *Saccharomyces cerevisiae*

by

Christiaan Johann Swanepoel



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Biochemistry) in the Faculty of Science at Stellenbosch University

Supervisor: Prof. J. M. Rohwer

March 2018

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2018/01/16 .....

Contents

Acknowledgements

I would like to express my gratitude to the following people and organisations:

First, I would like to extend my thanks and my appreciation to Prof. Johann M. Rohwer, whose help, guidance and attention to detail was a great boon and of the utmost help during this project.

Dr. Johann Eicher, without whom this project would not have been feasible.

Arrie Arends, the best lab-manager of all time, ensuring I had everything I needed to complete this project comfortably.

To my mother, family and friends for undying support, I would like to extend heartfealt thanks.

I want to express gratitude towards the University of Stellenbosch, for its laboratories and facilities and years of fun.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Dedications

I dedicate this thesis to my mother, sister & father...
...without your love and support this would never have been possible.

List of Figures

List of Tables

List of Abbreviations

ADP . . . . . Adenosine di-phosphate
ATP . . . . . Adenosine tri-phosphate
F6P Fructose-6-phosphate
FBP Fructose-1 6-bisphosphate
G6P Glucose-6-phosphate
NMR . . . . . Nuclear magnetic resonance
PFK . . . . . Phosphofructokinase
PGI . . . . . Phosphoglucoseisomerase
TEP . . . . . Triethylphosphate

Summary

Kinetic modelling of biological phenomena in an attempt to understand the underlying dynamics and complexity of life is becoming an indispensable tool to systems biology. A new paradigm of mathematical and computational integration of experimental data has shifted the focus in biological sciences from mere characterisation and cataloguing of the components of life, to a more holistic view. The functioning of these components in dynamic interactions in non-linear biochemical networks is now a major field of interest for many biologists. Classically, enzyme kinetic assays are optimised for yielding the maximal activity of the enzyme of interest. This raises the question of how applicable the obtained kinetic parameters are for systems biology, especially when considering how the intracellular reality (in terms of pH and ionic strength and composition) affects the catalytic activity of enzymes *in vivo*. Another concern is how accurate and predictive the kinetic models, constructed from such obtained data, can be. Much effort has been directed towards the standardisation of enzyme kinetics for systems biology and *in vivo*-like assay media have been developed for the determination of enzyme kinetic parameters in both *Escherichia coli* and *Saccharomyces cerevisiae*. However, the effect of pH changes on kinetic parameters of enzymes, has been somewhat neglected in systems biology studies. With this in mind we investigated the quantitative effects elicited by pH changes on the upper glycolytic enzymes in *Escherichia coli* and *Saccharomyces cerevisiae* using NMR spectroscopy. This is especially important as recent studies have shown that intracellular pH, while remaining a tightly homeostatically controlled parameter, is not as constant as once thought and has been shown to vary in response to environmental perturbation. The investigation focused on parameter estimation and the unique identifiability of the estimated parameters. The main aim of this project is the development of a robust, reliable technique for parameter identification from experimental data using mathematical and computational approaches.

Opsomming

In 'n poging om die onderliggende dinamika en kompleksiteit van die lewe beter te verstaan, raak kinetiese modellering van biologiese fenomene 'n onontbeerlike instrument vir sisteembiologie. 'n Nuwe paradigma van wiskundige en rekenaarmatige integrasie van eksperimentele data het die fokus in biologiese wetenskappe verskuif van blote karakterisering en katalogisering van die komponente van die lewe, na 'n meer holistiese siening. Die funksionering van hierdie komponente in dinamiese interaksies in nie-lineêre biochemiese netwerke raak 'n belangrike navorsingsveld vir baie bioloë. Histories is ensiem-kinetiese essai's geoptimeer vir maksimale aktiwiteit van die betrokke ensiem. Dit laat die vraag ontstaan hoe toepasbaar die verkrygde kinetiese parameters is in terme van sisteembiologie, veral as in ag geneem word hoe die intrasellulêre werklikheid (in terme van pH en ioniese sterkte en samestelling) die katalitiese aktiwiteit van ensieme *in vivo* beïnvloed. 'n Verdere bekommernis is die akkuraatheid en voorspellingsvermoë van kinetiese modelle wat op grond van sulke data saamgestel is. Onlangs is beduidende pogings aangewend om ensiemkinetika vir sisteembiologie te standaardiseer, en essai-media wat *in vivo* toestande naboots is ontwikkel vir die bepaling van ensiem-kinetiese parameters in beide *Escherichia coli* en *Saccharomyces cerevisiae*. Die effek van pH-veranderinge op kinetiese parameters van ensieme is egter ietwat verwaarloos in sisteembiologiese studies. Met hierdie in gedagte het ons die kwantitatiewe effekte van pH veranderinge op die boonste glikolitiese ensieme in *Escherichia coli* en *Saccharomyces cerevisiae* met behulp van KMR spektroskopie bepaal. Dit is veral belangrik aangesien onlangse studies getoon het dat intrasellulêre pH, terwyl dit 'n streng homeostatiese beheerde parameter bly, nie so konstant bly as voorheen gedink is nie, en dat dit kan verander in respons op versteurings in die omgewing. Hierdie studie het gefokus op beraming van ensiem-kinetiese parameters asook die bepaling van die unieke identifiseerbaarheid van hierdie parameters. Die hoofdoel van hierdie projek is die ontwikkeling van 'n robuuste, betroubare tegniek vir parameterbepaling vanaf eksperimentele data deur gebruik te maak van wiskundige en rekenaarmatige tegnieke.

Chapter 1

Introduction

1.1 Problem Identification

Despite much of the research conducted in the biological sciences, attempting to explain and define complex biological systems, there are still large gaps in our knowledge concerning the functioning of living organisms. The standard description of biology in dictionaries has life subscribe to the following characteristics: metabolism, self-maintenance, replication through genetic material and evolution via natural selection. Although being a very descriptive approach towards understanding biological characteristics, such a view fails to take the underlying integration and complexity, and resultant dynamic nature, of biological systems into account. However, we can gain a deeper and more holistic understanding of life and its underlying mechanisms by applying certain mathematical abstractions to biological systems [1]. Through such abstraction of intracellular molecular interactions, models of these interactions can be built and manipulated to gain new information about the system under study. Models can also be used to predict the behaviour of systems in response to perturbations in their environment and facilitate the formation of novel hypotheses and identification of previously unknown interactions and properties of the systems under scrutiny [1,2].

Significant developments in experimental techniques have caused molecular biology to evolve into a new paradigm [3] of systems level studies of complex regulatory structures [4], such as biochemical networks. The availability of high quality, information-dense data has increased tremendously with the advent of high throughput experimentation in the fields of genomics, transcriptomics, proteomics and metabolomics [3–7]. There is thus an increased demand for integrated computer analyses, capable of organising these disparate, heterogeneous bodies of data into a coherent, logical whole. Systemic bio-informatics solutions are employed in an effort to elucidate and predict the behaviour of complex systems. This is achieved via the integration of experimentally obtained data and computational approaches to build mathematical models [4]. Figure 1.1 shows the model followed in systems biology investigations in terms of experimental design, data analysis and subsequent model construction.

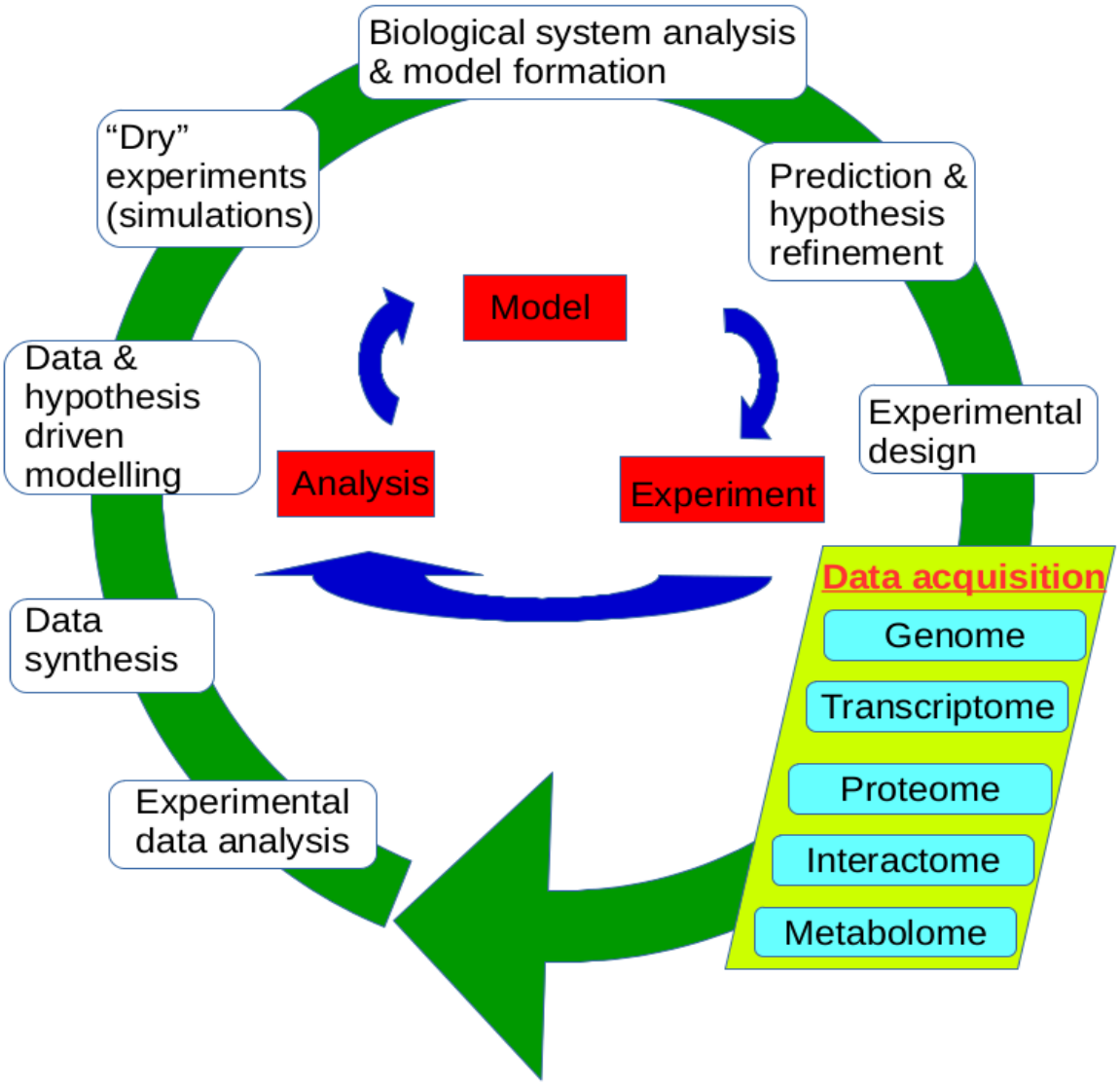


Figure 1.1: The model followed in systems biology investigations in terms of experimental design, data analysis and subsequent model construction. This is an iterative process assisting in the refining of experimental design and hypothesis formulation.

Systems biology is an interdisciplinary field of study [2, 4, 8–10] concerned with the elucidation and quantification of the complex, non-linear biomolecular interactions (from subatomic to mesoscopic), which can be seen as giving definition to living systems [9, 11]. A study of these interactions can only be achieved by the effective integration of biomolecular, mathematical and computational approaches, as a multi-scale approach is necessitated for construction and analyses of models of interactions with units ranging from the mesoscopic to time-scales [11]. Such an investigation into the mechanisms governing life will eventually lead to a more comprehensive interpretation of biological interactions and phenomena and ultimately a more holistic picture of biological function, based on the data provided by the various "-omics" approaches, than is afforded us by the traditional, reductionist approach to biology [9].

The ultimate goal for systems biologists is the utilisation of mathematical models to understand how the individual components of a system work and influence each

other. In the case of networks that describe metabolic functions, the component of interest would be the enzyme, since this is the catalyst of biomolecular interaction between molecules. Constructing such kinetic models would therefore require kinetic information about the various enzymes involved in the interaction. Building a realistic quantitative model requires accurate data from many laboratories and experiments [5, 12]. Enzyme kinetic parameters are obtained by performing enzyme kinetic assays, using the enzymes relevant to the current study [13]. Historically, since the majority of enzyme assays were carried out for the elucidation of catalytic mechanisms, most assays were conducted in conditions irrelevant to current model requirements [14]. *In vitro* enzyme assays are traditionally conducted in conditions optimal for the enzyme under study [15] (in terms of pH, ionic composition and strength, etc.) and the individual components of a system are studied in isolation. For the purpose of modelling, data obtained in this manner are not useful as the conditions are not representative of the *in vivo* situation. The result is that large numbers of enzyme kinetic assays need to be performed under conditions suitable for systems biology [14].

Of particular interest is that often the enzymes in a single pathway would each be studied at their specific optimum pH, which results in many enzymes being studied at different pH values. This is a situation far removed from the physiological truth, as the optimum pH values for all enzymes of a pathway are rarely the same [13]. Insofar as pathways are situated within the same cellular compartment, the pH would be the same for all constituents of that pathway [16]. Since intracellular pH has been shown to be much more dynamic than originally thought [17]. Investigators have recently postulated that intracellular pH change may function as the glucose signal, induced by a cellular response to glucose availability. This signal is responsible for regulating metabolic processes, such as the indirect activation of the protein kinase A (PKA) pathway via vacuolar ATPase, in response to glycolytic flux changes and subsequent intracellular pH changes [18, 19]. Furthermore, pH change in response to nutrient availability has been linked to the control of growth [17, 20] and membrane biogenesis in *Saccharomyces cerevisiae* [21].

In light of this information, the purpose of this study will be to investigate the quantitative effects elicited by pH changes on the enzyme kinetic parameters of the upper glycolytic enzymes in *Escherichia coli* and *Saccharomyces cerevisiae*: phosphoglucoseisomerase (PGI) and phosphofructokinase (PFK).

1.2 Project outline

In this project we employed a newly developed method for the rapid determination of enzyme kinetic parameters from progress curves generated by nuclear magnetic resonance (NMR) spectroscopy [22, 23]. This method has several advantages over traditional enzyme assay methodologies in that it requires fewer runs, as entire time course data are utilised and are thus more information-rich as numerous substrates, products and allosteric modifiers can be identified simultaneously. It is a more labour and cost-effective and suitable method for systems biology for parameter determination than classical enzyme kinetic assays. With this method all the glycolytic enzymes of *E. coli* have been characterised and assembled into a kinetic model [22].

The upper glycolytic enzymes were analysed using the above outlined NMR technique, at varying pH levels. The enzymes were subsequently characterised in terms of

the pH-dependence of their kinetic parameters (V_{max} and K_m values for all substrates and products, K_i values for inhibitors, etc.). Kinetic parameters were measured at pH values of 8.0 and 5.5 in addition to pH 7, as these values correspond to exponentially growing and starving *E. coli* and *S. cerevisiae* cells respectively [24–26].

The NMR method as developed by Eicher in [22] allows that the first two enzymes of glycolysis, PGI and PFK, can be assayed separately from the rest of the glycolytic pathway [23].

1.3 Aims, objectives & outline of this thesis

1.3.1 Establish baseline measurements for kinetic parameters of the upper glycolytic enzymes in *E. coli* and *S. cerevisiae* at pH 7

To establish baseline measurements of the kinetic parameters of PGI and PFK, kinetic enzyme assays were performed. Reactions were monitored using ^{31}P-NMR spectroscopy.
The first two reactions in glycolysis are catalysed by PGI and PFK. These reactions can be separated easily by the inclusion or omission of ATP:

$G6P \longrightarrow F6P$ (PGI)
$F6P + ATP \longrightarrow FBP + ADP$ (PFK)

This allows for the separate study of the PGI reaction. We can however, not study the PFK catalysed reaction in isolation, as the PGI catalysed reaction will always be coupled with the PFK reaction in a cell free extract. To study the PFK catalysed reaction we need to investigate the PGI-PFK module, as the two reactions are inseparable.

1.3.2 Investigate the quantitative effects of pH changes on the enzyme kinetic parameters of the upper glycolytic enzymes in *E. coli* and *S. cerevisiae*

The same modular approach as employed in 1.3.1 was used to determine the kinetic parameters of PGI and PFK at pH 5.5 and pH 8.

1.3.3 Kinetic parameter estimations

Parameters were estimated by fitting rate equations (in the form of ordinary differential equations) to the experimentally obtained NMR data using the Nelder-Mead simplex algorithm.

1.3.4 Determine identifiability of the obtained kinetic parameter estimations

To determine the identifiability of the parameters obtained through fitting, the Nelder-Mead algorithm was used to test each obtained parameter individually by generating

profile-likelihoods.

1.3.5 Thesis outline

The following chapters and general outline of this thesis are as follows.

First, a literature review will be presented, giving a general review of the background information on the field of systems biology and enzyme kinetics as well as a brief overview concerning parameter determination and identifiability analysis.

The experimental methods section follows, presenting the protocols for experimentation and data analysis.

In the results chapter all the results from the analysis of experimentally obtained data are presented and discussed.

In the final chapter the obtained results are discussed in the light of the projected aims and objectives, as well as the current literature concerning this topic of investigation.

All code used in this research project is presented in the appendices at the end of this thesis in script format.

Chapter 2

Literature review

2.1 General review of the field

To date, molecular biology provided a means to visualise and characterise biological phenomena mechanistically at the molecular level, and structure was seen as the major basis for the explanation of biological behaviour [27]. In light of this the 20th century constituted the era of discovery, where the components of life were separated, identified, characterised and categorised. Advances in experimental procedures and technologies have brought scientific discovery to the point where almost all the constituent molecules of a cell can be characterised experimentally [28]. The challenge in the post-genomic era is the integration of vast amounts of data generated by high throughput technologies with computational approaches to build models able to describe the dynamic and complex reaction networks that comprise living organisms. Due to the significant number of cellular components and interactions between these, cellular networks and the models describing them are immensely complex. This is a non-trivial complexity as it is exactly what allows for life to emerge from interactions between molecules, which, in isolation are non-living [28, 29]. In this new era of biology, investigators widened their view from the static, single molecule and reaction approach to dynamic interactions at the cellular and organismic level [27].

To gain proper insight into, and to conduct comprehensive analyses of the whole cell at different levels of organisation, requires a far more dynamic experimental and analytical approach than is provided by the more static perspective of genome scale approaches [6]. The methods provided by systems biology, where mathematical and biological data are integrated in the construction of computational models, are more detailed and in-depth, and provide a powerful approach for the elucidation of the biological functioning of organisms. Recent advances in computational power and infrastructure make whole cell computational models possible–the first whole cell model was published in 2012 for *Mycoplasma genitalium* [30].

The large quantity of data, collected with high throughput experimental techniques [31] and genome sequencing, has led to an increased demand for their integration into complex intracellular networks. Systems biologists developed a number of modelling approaches in an attempt to understand the dynamic, non-linear biological interactions associated with life [29]. The top-down approach attempts to infer genome-wide network behaviour through integration of whole genome transcriptomic, and high throughput protein interaction data. This approach suffers from the drawback that emergent properties of the system are not visible as the models are not

constructed with the individual properties of all the constituent components; however, these models have the advantage of being genome-wide [14, 22].

Genome-wide models can be used to study entire organisms, since they represent all known biochemical pathways of an organism [32]. In the absence of kinetic information, flux balance analysis attempts to quantitatively understand metabolic phenotypes in terms of an underlying optimisation factor, such as maximisation of growth rate. Flux balance analyses have proven successful for the determination of input and output fluxes of organisms but cannot account for the dynamic phenomena important for homeostatic control under varying environmental conditions [33].

A more suitable approach, for the current investigation, is bottom-up modelling, which is used in the silicon cell approach [29]. The bottom-up approach incorporates precise rate equations (parameterised with experimentally obtained data) to construct models of networks from their constituent molecular components. Models built with this method aim to precisely describe intracellular networks, and are predictive and accurate, exactly because they are built using experimental data of their components. Bottom-up models are more labour intensive and difficult to create, thus smaller parts of networks are focused on for more in-depth analysis. Predictive power and precision are thus chosen above network coverage [22]. Figure 2.1 is a graphical representation of top-down versus bottom-up approaches to biology.

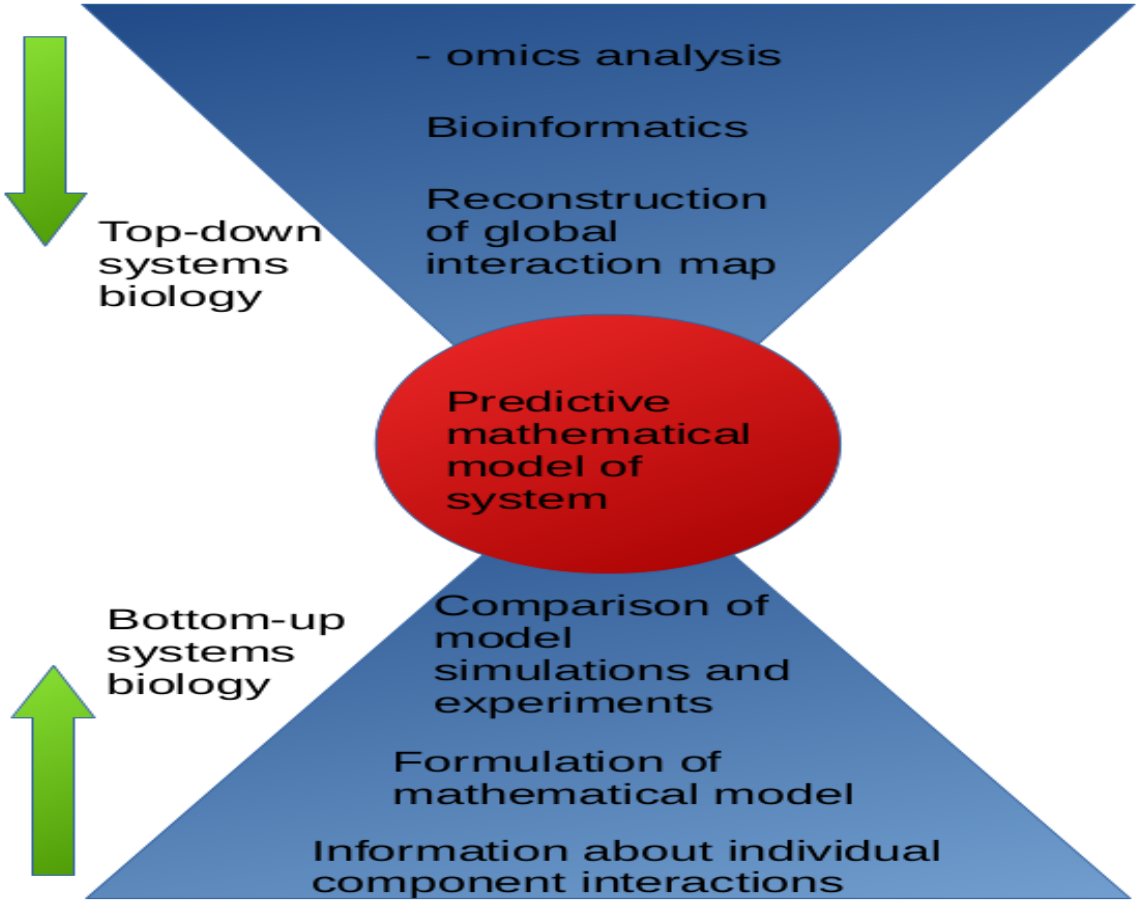


Figure 2.1: A graphical representation comparing the main aspects of top-down and bottom-up modelling.

Enzyme kinetic data constituted a large part of our biological knowledge since the publication of Michaelis and Menten's ground breaking article in 1913 [34, 35]. Nevertheless our understanding of biological processes is still inadequate, and there are still large gaps in our knowledge of the kinetic processes governing life. There is much ongoing work to create accurate bottom-up descriptions of biological networks [14].

Due to their utility in systems biology, enzyme kinetic studies have swiftly regained regard and popularity [14]. Building dynamic models in a bottom-up fashion necessitates detailed knowledge about the kinetic properties of the components of the system under investigation. Kinetic parameters, describing the catalytic activity of enzymes, are obtained by performing enzyme kinetic assays [12–16, 28, 36]. During the age of structural and functional discovery of biology, the field of enzyme kinetics was largely abandoned. With the rise in popularity of enzyme kinetics, the field is reinvigorated and significant progress has been made, such as the development of the reversible Hill equation [37,38], and subsequent derivation of the generic reversible Hill equation [39, 40].

This rate equation can describe experimental data with the same accuracy than more mechanistic equations whilst having fewer parameters. The parameters are furthermore operationally defined, making them subject to experimental determination. For example, the half-saturation constant is the concentration substance (either substrate, product or modifier) that elicits an effect that is half that of the maximum possible effect [22].

In the new epoch of biological science, adopting a more holistic, systems biology view of life, it is clear that investigation into the function of biological networks can greatly benefit from enzyme kinetic data describing the components of these networks and mathematical analysis of models built from those components [28].

Classical enzyme assays make use of initial rate data, obtained by incubating the enzyme of interest with its relevant substrates, products, co-factors and allosteric modifiers [15]. Assays are often performed under conditions designed to yield maximal enzymatic activity [12], or are assayed in the direction opposite to their natural function [15] and most assays are done in dilution to avoid interference by compounds present in the cell free extract [41]. These are all conditions rarely found *in vivo*. Insofar as *in vitro* enzyme assays are performed in such conditions, how can researchers be sure that the observed behaviour is representative of the *in vivo* behaviour of the enzymes under investigation? Enzyme assays for systems biology should provide information about enzyme activity that is as close to the *in vivo* reality as possible [23]. To address this, separate research groups set out to develop standardised, *in vivo*-like assay media with the purpose of determining enzyme kinetic parameters for *E. coli* [12] and *S. cerevisiae* [16]. In both cases, kinetic parameters obtained from media optimised for specific enzymes differed conspicuously from parameters determined using the standardised media specific for both microorganisms. For this project we will be focusing on one particular aspect that has been somewhat neglected in systems biology, namely pH changes.

2.1.1 pH Changes

pH has long been known to be a factor that affects protein activity [42], particularly when catalysis depends on the uptake of protons (such as the reduction of substrate by nitrogenase [43]) or is accompanied by proton transfer. pH also plays a major role in

enzyme catalytic activity when the catalysis of substrates depends on the protonation state of an enzyme. In glutathione redox reactions, the reduction-oxidation potentials of both glutathione reductase and dehydrogenase were shown to be dependent on their protonation state [44]. In addition, allosteric protein modifiers are also affected by changes in pH, and most, if not all enzymes are allosterically modified [45]. Further, pH changes have been shown to affect conformation changes, altering the activity of proteins [46, 47].

Recently Orij *et al.* [17] showed that although intracellular pH is a tightly controlled parameter in microorganisms it is by no means as stable and well buffered as previously thought [48]. Contrary to intuition, it is a highly dynamic parameter in *S. cerevisiae* and fluctuates significantly in response to changes in the environment, controlling the microbe's adaptive responses [44]. Evidence is mounting that pH might fulfill a signalling role and protons have been shown to act as a second messenger in *S. cerevisiae*. Investigators showed that binding of proteins to phosphatidic acid (PA) and subsequent binding of PA to transcription factor, Opi1, was strongly dependent on intracellular pH and the protonation state of the phosphate headgroup of PA. They demonstrated that a drop in intracellular pH in response to glucose starvation regulated this binding. PA acts as a pH biosensor and binding to Opi1 represses phospholipid metabolic genes, in this way linking membrane biogenesis to nutrient availability [21]. Furthermore, a role for the control of growth by intracellular pH changes in response to variance in nutrition was shown for yeast on separate occasions [17,20]. An enzyme storage mechanism was also recently demonstrated for intracellular pH in budding yeast, where a drop in intracellular pH, due to advanced starvation, caused the reversible inactivation, by filament formation, of glutamine synthase 1. Upon re-addition of nutrients the enzymes returned to their natural, active state, leading to the hypothesis that enzyme storage serves as a protection mechanism in response to starvation [49].

Taking into account that pH affects the catalytic activity of proteins, the question whether intracellular pH changes may constitute a mechanism for flux control cannot be ignored, particularly when considering the enzymes of glycolysis. Many of the reactions in the glycolytic pathway are dependent on nucleotides, such as ATP (existing in varied protonation states), as cofactors. The protonation state of nucleotides and enzymes is in turn dependent on the pH and ionic strength of their environment. To date a systematic investigation into the effects of pH change on glycolytic flux and enzyme activities has not been performed.

2.1.2 Nuclear magnetic resonance spectroscopy and kinetics

Nuclear magnetic resonance (NMR) spectroscopy has proved to be a useful tool for flux monitoring, as well as the determination of enzyme kinetic parameters *in situ* [50, 51]. NMR enables the real time determination of external glycolytic fluxes in living cells [52] and can thus be used to monitor metabolism, for example, the investigation into nitrogen metabolism in *Datura stramonium* by means of ^{15}N-NMR spectroscopy [53].

NMR is an unbiased analytical technique, applicable to the study of both pro- and eukaryotes [51,54–57]. This technique can be used to follow even the bio-transformations affected by enzymes [50, 54] and allows the quantification of metabolites from cell extracts [58, 59], making it a very useful analytical tool for systems biology.

^{31}P-NMR spectroscopy will be used in this project to determine the kinetic parameters of the upper glycolytic enzymes at different pH values, which can subsequently be integrated into metabolic models, such as that constructed by Eicher *et al.* [23]. This is beyond the scope of this study, however, the results will be compared to baseline parameters to evaluate the effects of pH changes on the kinetic parameters of the upper glycolytic enzymes in *E. coli* and *S. cerevisiae*.

2.1.3 Construction of models

The construction of models has a greater motivation than only satisfying curiosity and are built for more than just for interest's sake. Acquiring data on enzyme kinetic parameters and the subsequent construction of models is a powerful investigative tool for drug development, clinical diagnosis, agriculture and biotechnology [4, 60].

Top-down models are constructed from data based on the intact system. These data include metabolite concentrations, fluxes through the pathway, metabolic control analysis data and steady-state information. By fitting model parameters to these experimental data, kinetic parameters can be inferred.

Building bottom-up models of reaction networks, on the other hand, requires many different types of data to be representative of the biological phenomenon under investigation (see [61] for a review).

- First, the stoichiometry of the reaction network is vital as it provides information about the number and essential characteristics of compounds involved and their relation to each other. This gives an indication of the general structure of the model in terms of the connections between reactions.

- Second, detailed knowledge of the kinetics of enzymes involved in the network is necessary (such as K_m, V_{max}, K_i, etc.). This often constitutes a problem as such kinetic data are not always readily available, however, data can be experimentally obtained where it is lacking. In this case it is preferable that data is collected in conditions resembling the *in vivo* situation.

- Third is thermodynamic data about the enzymes present in the reaction network. It is important to include this data when building kinetic models whenever it is feasible, as it provides insight into the reversibility of enzyme-catalysed reactions. Furthermore, the degree of reversibility of the model constituents can shed light on the communication and feedback loops of intermediate metabolites, as well as the direction of reactions in a pathway.

- Fourth, data regarding the maximal enzyme activity is required. The maximal activity gives an indication of the rate-limiting effect of the amount enzyme available. This is due to the rate of enzyme-catalysed reactions being directly dependent on the concentration of that enzyme and its catalytic rate constant.

- Fifth and last, data concerning external and fixed metabolites, moiety-conserved cycles and the starting concentrations of all model variables is necessary for the construction of the model.

Independent data, not used in the construction of the model, is then compared to model outputs to validate the model. Once a model has been experimentally vali-

dated it can be used to make predictions, design new experiments and explore novel interactions that might not have been observed before.

A benefit of bottom-up models is that they can be used to make predictions for different conditions, as the model is based on inherent properties of enzymes (such as their kinetic and thermodynamic parameters) that do not change in response to external environment. Top-down models are a viable and powerful alternative when such data are not available or easily obtainable experimentally. However they can only make predictions based on the conditions under which they were built as they are based on data of the intact system and kinetic parameters are inferred. Thus the environmental conditions may have an effect on the parameter values, and predictions will only hold for those conditions [61].

2.1.4 Models and their applications

Medical science has undergone a paradigm shift, where a systemic approach that takes the entire biochemical network into account has become common practice [10]. For example, investigators introduced system biological strategies to elucidate the complex and multi factorial problems involved in traumatic brain injury and subsequently identified a sub network of 58 highly interactive and co-regulated proteins in association with synapse functioning [7]. The study of pathogen-host interactions has also benefited from employing systems biology strategies, where the investigation of the pathogen-host interaction as a whole, single entity has led to the elucidation of previously unknown infection mechanisms and with recourse to high throughput "-omics" data will facilitate the identification of novel therapeutics of increased efficiency for the treatment of disease and infection [9]. Computational systems biology, integrating data from various sources to construct dynamic models, has been shown and can be expected to continue to be useful in the pharmaceutical and medical industries as well as being applicable to various other industries, including agriculture. One such example is the application of a kinetic model to the accumulation of sucrose in sugar cane [62], where potential gene-engineering targets could be identified by observing control points within the biological reaction network responsible for sucrose production.

2.2 Data analysis: parameter estimation and identifiability

In systems biology, mathematical abstractions are often used to represent biological networks in terms of series of ordinary differential equations (ODEs). As these networks are often complex, models of these networks subsequently tend to be large, complex and contain multiple parameters that need to be determined to explain experimental observations.

The level of accuracy of parameter estimation is paramount, given a model that satisfactorily describes the experimental data. This is especially critical as the amount and quality of data plays a pivotal role in how accurately, and indeed if, parameters can be determined. This issue is non-trivial as knowledge about the trustworthiness of parameter estimations is fundamentally important for further exploration and reliability of model predictions. In light of this, unique parameter identifiability has long been a field of interest in molecular systems biology [63–72].

2.2.1 Parameter estimation

Models of biological networks, represented as mathematical abstractions, have a wide range of applications in many biological fields and assist investigators in understanding the functional properties of biological systems. Models allow the investigation of emergent properties of biological networks, the simulation of reactions before experiments are carried out as well as the exploration of biological phenomena that cannot necessarily be observed experimentally [63]. Parameters often have to be inferred from model comparisons to experimental data, as a limited number of parameters can be directly determined from experimentation or the existing literature and often the experiments to obtain existing parametric knowledge, have not been conducted with systems biology in mind. To name but one example, performing kinetic assays for enzymes in a pathway, each at its own optimal pH, is a situation quite removed from the physiological truth of enzymatic pathways.

Parameter estimation faces two challenges. The first is building a model that can sufficiently describe a certain biological phenomena. The second is fitting of said model to experimental data, to obtain the parameters with which to calibrate the model, so that meaningful predictions can be made. This is usually accomplished through an iterative data-fitting process, developing models based on experimentally obtained data. Data-fitting involves an objective function that endeavours to minimise the difference between experimentally observed measurements and the model simulation [72]. To effectively explore the parameter space of the system under scrutiny, a range of experiments are necessary, with varying starting concentrations of substrate, products and co-factors. However, the solutions obtained may be local minima, rather than the desired global minima. This is where the choice of minimisation method as well as the identifiability of parameters play a very important role [65, 72]. As there are many different problems investigated by systems biologists, there are many methods of minimisation, with various strengths and weaknesses, making them more or less appropriate to deal with the problem at hand. A few will be mentioned here, due to their applicability to this investigation.

Various optimisation methods are used to determine the minimum of the objective function and can be described as being either local or global. Global optimisation methods will always yield the global minimum for parameters as the entire parameter solution space is sampled, but they tend to be expensive computationally and chronologically. However, local methods, assuming that the initial parameter guess is close to the minimum, tend to converge to that minimum relatively fast [72].

For this investigation the focus lies on local optimisation methods as good initial guesses can be made about model parameters, based on existing literature. Local optimisation methods fall in two main classes: direct-search and gradient-based methods. There are certain cases where gradient-based methods are not applicable, such as when the objective function is discontinuous. In these cases direct-search methods are best implemented, thus the focus in this investigation will be on these optimisation methods.

2.2.2 Direct-search local optimisation methods

Two direct-search local optimisation methods are used more often than any others, the Hookes-Jeeves method and the Nelder-Mead Simplex method [72].

Hookes-Jeeves

Hookes and Jeeves [73] employ a pattern search method, consisting of two steps. First, a pattern of exploratory changes, both up and down from the current parameter vector are made, to investigate whether they are worse or better than the current parameter value. The perturbations are executed one parameter at a time. This creates a picture of the parameter space containing details about which perturbations had minimising effects on the objective function. Step two concerns the optimal direction, based on the information obtained in step one, in which the minimisation process needs to proceed to reach a solution. This method has been shown to coincide in convergence guarantee with methods that utilise derivatives of ODEs, so-called gradient-methods.

Nelder-Mead

The Nelder-Mead [74,75] algorithm relies on an adaptive simplex, ordinarily a polytope (a geometric object with vertices in m dimensions) of m + 1. Two-dimensionally this could be visualised as a triangle, or a tetrahedron three-dimensionally. The values of the vertices (sides) are determined by evaluating the objective function in all vertices, these vertices correspond to the parameter being tested. The calculated vertices are ordered in terms of their respective values, and the 'worst' vertex is replaced by a 'better' one by the algorithm. The objective function is subsequently tested to determine if it is indeed better than the one parameterised with the old vertex. If not, the 'best' vertex in the simplex is kept, whereas all the others are replaced by a vertex halfway on the line from the vertex that was kept. If the new objective function is indeed better than the old one the simplex is adapted by simply replacing the 'worst' vertex with the newly determined vertex. This process is repeated until all vertices have been replaced with better ones, determined by the algorithm, until the objective function has been optimally minimised. One of the benefits of this method is that the simplex is adaptive with regard to the objective function and usually needs only one to four evaluations per step.

The main difference between this and the Hookes-Jeeves method is that the Nelder-Mead improves the objective function in line with the succession of worst vertices.

In the current study, the Nelder-Mead Simplex [74, 75] method will be employed for parameter estimation.

2.2.3 Classes of identifiability: structural vs. practical

Model parameters are not always necessarily unambiguously determinable under specific experimental conditions, even given a certain quality and amount of experimental data. For biological networks, data is often scarce, are subject to certain technical or technological restrictions and limitations, and often, complex biological networks are not fully visible with currently available experimental techniques.

This can lead to parameters that may be non-identifiable. Identifiable parameters are also said to only be estimable to within a certain probability range, indicated by confidence intervals, containing the true parameter value within a desired likelihood. The accuracy to which parameters can be estimated, will in turn determine the accuracy of model predictions. Subsequently, poorly determined parameters lead to ineffective models, that may result in the non-addressability of certain biological questions the model needs to be capable of answering. Evaluation of which parameters are identifiable indicates what model predictions are feasible. The size of the

confidence intervals of identifiable parameters further indicates the reliability of model predictions.

Identifiability comes in two flavours, structural and practical, often known by the names *a priori* and *a posteriori*, respectively A parameter is said to be identifiable if the confidence intervals containing the estimated parameter are finite [63, 72].

Structural identifiability is dependent on the structure of the model, and is unrelated and independent of the quality or amount of data and can be determined before experimentation. Practical identifiability however, relies directly on the amount as well as the quality of experimentally obtained measurements used for parameter estimation. Practical non-identifiability can be remedied by collecting additional, high quality data, whereas structural non-identifiability often points to redundant parameterisation in the model, but can also shed light on functionally related or correlated parameters [63].

2.2.4 Identifiability analysis

There are several methods for the determination of identifiability of parameters of a given model, assuming sufficient quality data of the biological phenomena to be studied are available. Figure 2.2 illustrates the steps followed in the determination of identifiability.

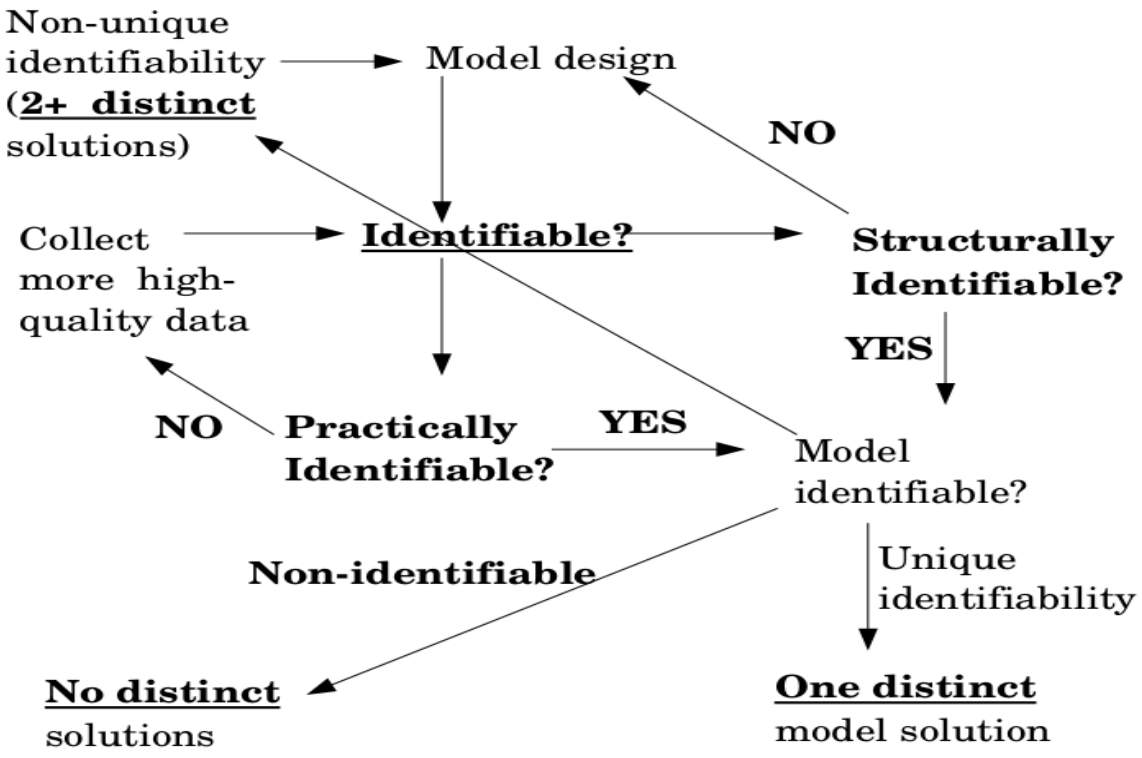


Figure 2.2: An illustration of the model for identifiability analysis and the steps for the determination of unique identifiability.

Three of these approaches to identifiability analysis will be briefly outlined here: DAISY, EAR and PL.

DAISY method

DAISY (Differential Algebra Identifiability of Systems) [76] employs a differential algebra algorithm which executes a global parameter identifiability analysis of dynamic models described by either rational or polynomial equations. The main principle is the manipulation of algebraic differential equations as polynomials dependent on the derivatives of the variables. This method permits the elimination of non-observed state variables from the system of equations and allows the investigator to find the input-output relation of the system. This input-output relation is then linearly parameterised using specific algebraic functions of the unknown parameters. Subsequently this leads to non-linear equations based on the parameter under investigation. Application of certain computational algorithms tests if one or multiple solutions are available. The main advantages of this approach is that it does not require exhaustive knowledge of mathematical modelling from the investigator as well as the method's ability to distinguish between local and global identifiability or non-identifiability of the original dynamic model [65].

EAR method

The EAR (Exact Arithmetic Rank) approach [77,78] is based on the application of the inverse function theorem to a system of algebraic equations that associates higher order derivatives of the output of said equations with the initial state of the system and parameters with respect to time at the starting time of experiments and simulations. The approach is based on methods for local algebraic observability, utilising differential algebra to establish an upper bound to the order of differentiation. This results in a series of non-linear algebraic equations in the parameters. The Jacobian matrix ranking of this system of equations provides information on the solvability of the system, and can also give insight as to the parameters involved in non-identifiability. The main advantage of this approach is that it can be employed in an automatic function in certain programming languages, where the experimenter merely inputs equations and an answer is generated [65].

PL method

The PL method is the probability-likelihood approach [63,64,79]. This method tests for non-identifiability through constituting a parameter estimation problem using data representing the system under study. The main concept is that non-identifiability is visualised as a perfectly flat valley in the parameter solution space of the estimation problem. Profile-likelihoods are generated for each parameter individually, where all other parameters are iteratively re-optimised through a series of up and down perturbations of the fitted parameter. The process is repeated for each parameter generating profile-likelihoods for each parameter individually. The benefits of this method are that structural non-identifiability can be determined using only simulated data and practical non-identifiability can be determined as well, given experimental data availability. Confidence intervals for all parameters can also be calculated with this method. They appear in one of two forms; asymptotic or finite sample confidence intervals.

Asymptotic confidence intervals are based on the curvature of the profile-likelihood of the parameter, whereas finite sample confidence intervals are based on the goodness of fit falling below certain pre-decided threshold value, indicating the level of confidence desired [63].

This method can be useful in identifying the reason for non-identifiability, pointing to missing data or redundancies or functional correlations of parameters of models.

This approach has been proven to facilitate an iterative strategy of experimental

design and can be expanded to detect non-observability of dynamics directly.

For the current investigation the PL approach will be implemented for identifiability analysis [63, 65].

Chapter 3

Methods and materials

The aim of this study was to investigate the effects of pH changes on the kinetic parameters of the upper glycolytic enzymes of *E. coli* and *S. cerevisiae*. The following chapter outlines the experimental design in terms of the methods and materials used to accomplish the investigation.

3.1 Culture and harvest of microorganisms

Both microorganisms were grown in darkness on orbital shakers rotating at 150 rpm. *E. coli* was grown at 37^{o}C and *S. cerevisiae* at 30^{o}C, in accordance with standard laboratory protocols.

Escherichia coli

Freezer stock was made from liquid cultures of *E. coli* (wild type W110) grown overnight. One-ml aliquots were taken and frozen at -80°C, in 40% sterile glycerol for future use. Original liquid cultures were grown from laboratory freezer stock.

To make working liquid cultures a sample of freezer stock was incubated overnight in M9 minimal medium (M9MM) in stoppered Erlenmeyer flasks. Dilutions of the overnight liquid cultures were subsequently plated on agar plates.

A single colony was isolated from the plate grown from the undiluted sample, and used to inoculate 3 litres M9MM to grow a working liquid culture.

Liquid cultures were allowed to grow to the mid-logarithmic growth phase followed by the harvest of cells by centrifugation using a JA-10 rotor at 12400 $\times$ g for 10 minutes at a temperature of 4°C.

Cells were harvested at an optical density of 0.7 (600nm). Three litres of liquid culture was harvested by centrifugation and cell pellets were subsequently resuspended in PIPES buffer (100mM, pH 7.2) and combined to make a 200 ml cell suspension. Aliquots were taken and divided into 100 Eppendorf tubes (2 ml) and centrifuged for 7 minutes at 16100 $\times$ g and 4°C. After discarding the supernatant cell pellets were stored at -80°C, for use throughout the duration of the experiment

Agar plate composition was 1 g KH_2PO_4, 1 g $(NH_4)_2SO_4$, 0.5 g $MgSO_4$.7H20, 20 g glucose, 10 g yeast extract and 15 g agar per litre.

M9 minimal media was prepared as follows:

Solutions were prepared and autoclaved separately to avoid precipitation of metals and in the case of glucose to prevent caremelisation. First M9 salt solution was prepared and autoclaved, the solution contained 6 g Na_2HPO_4, 3 g KH_2PO_4, 5 g NaCl, 1 g NH_4Cl per litre. Second, $MgSO_4$ (1M), $CaCl_2$ (0.1M) and glucose (20% w/v)

were prepared and autoclaved separately. Finally 979 ml M9 salts was mixed with 20 ml glucose (20% w/v), 2 ml $MgSO_4$ (1M) and 1 ml $CaCl_2$ (0.1M) to prepare 1 litre M9MM [22].

Saccharomyces cerevisiae

Freezer stock was made with the same method as employed for *E. coli*, using laboratory freezer stock of wild type *S. cerevisiae* (CEN.PK.113-7D).

The same agar composition was used as for the growth of *E. coli*. Liquid cultures were grown in yeast minimal medium (YMM). YMM contained 10 g glucose, 6.7 g yeast nitrogen base and 20.44 g potassium hydrogen phthalate (KHP) per litre, a similar composition to that used by Diogo *et al.* [80].

Harvest of *S. cerevisiae* cultures was accomplished using the method as outlined for *E. coli*. Cells were harvested at an optical density of 4.6 (600nm).

3.2 Cell extracts: preparation and protein determination

Glass beads were used to prepare crude cell extracts for both microorganisms. Extraction protocol efficiency was optimised by determining the optimal protein activity and yield versus vortex time. This was accomplished by microtitre-plate assay where PGI activity was measured at various extraction times.

Cell pellets as prepared in Section 3.1 were resuspended in either PIPES or MES-KOH buffer (100mM), at the appropriate pH (PIPES for pH values above 6.5 and MES-KOH for pH values below 6.5).

Bradford assays were used to determine the protein content of crude cell extracts from a bovine serum albumin (BSA) standard curve using linear regression [81] .

E. coli

Cell pellets were resuspended in 1 ml of the appropriate buffer and were subsequently added to 1 g glass beads (0.1 μm diameter), in a 10 mm glass test-tube, followed by 6 minutes of vortex at full speed. The sample was placed on ice every minute for 15 seconds to prevent overheating. The samples were centrifuged at 16100 $\times$ g for 10 minutes at 4°C, the supernatants were kept for further analysis.

S. cerevisiae

Cell pellets were resuspended in 1 ml of the appropriate buffer and were subsequently added to 1 g glass beads (0.25 - 0.5 mm diameter), in a 10 mm glass test-tube, followed by 8 minutes of vortex at full speed. The sample is placed on ice every 30 seconds for 30 seconds to prevent overheating. The samples were centrifuged at 16100 $\times$ g for 10 minutes at 4°C, the supernatants were kept for further analysis.

3.3 Enzyme assays

Enzyme assays were performed at pH 5.5, 7 and 8 with varying starting concentrations of substrates, products and/or co-factors (such as allosteric modifiers). Magnesium-chloride was added to each assay to ensure optimal enzyme activity as well as to provide saturating conditions for the complex formation of ATP and ADP with Mg^{2+}.

All reagents were dissolved in the appropriate buffer, dependent on the current working pH. This ensured that the final pH of the assay, containing all constituents,

is accurate, constant and correct. This meant that the pH was homogeneous and did not change due to mixing of the various components present in the assay mixture.

Kinetic enzyme assays were prepared in 5 mm NMR tubes. For PGI assays, assay mixtures contained 100 μl 50 mM TEP, 100 μl D_2O, 100 μl cell extract (protein concentrations ranging from 0.3 to 1.5 mg.ml^{-1}), G6P as substrate for the forward reaction or F6P for the reverse reaction (starting concentrations ranged from 3 mM to 40 mM) and 10 mM MgCl.

PFK assay mixtures contained 100 μl 50 mM TEP, 100 μl D_2O, 100 μl cell extract (protein concentrations ranging from 0.3 to 1.5 mg.ml^{-1}), F6P (starting concentration: 3 mM to 40 mM), MgATP (starting concentration: 2 mM to 10 mM) and 10 mM MgCl.

The metabolite concentrations used in the enzyme assays do not necessarily represent physiological concentrations. The reason for this is that starting concentrations on either side of the K_m value of the enzyme of interest are needed for a complete characterisation.

All assays had a final volume of 1 ml. The missing volume was made up with 100 mM PIPES or MES-KOH (working pH determined which buffer was used).

3.4 NMR for determination of enzyme kinetic parameters

^{31}P NMR was preformed on a Varian 400 MHz spectrometer, at 25°C and a frequency of 161.89 MHz. Arrays of NMR spectra were collected with a Fourier induction decay (FID) repetition time of 2.4 min, with 12 transients per FID, using a pulse angle of 90° and a relaxation delay of 12 s per transient, with proton decoupling without Nuclear Overhauser Enhancement (NOE). T1 relaxation experiments indicated that all metabolites in solution should be fully relaxed after a 6 s relaxation delay. A fully-relaxed spectrum (52 s) was collected following each enzyme assay so that if there are species present that are not fully-relaxed they can be calibrated [22].

Cell cultures were grown in minimal medium at pH 7.2 (prohibits the pH-inhibition of glycolysis and acts as buffer against acidic fermentation product formation) for *E. coli* and *S. cerevisiae* and harvested by centrifugation [22]. Extracts were prepared by glass bead extraction at the appropriate pH.

Samples were prepared for assay by including all substrates, products and allosteric modifiers of interest at the concentrations of interest in an NMR tube. All necessary co-factors are also included in the NMR tube and constitute such molecules as salts, metal ions and chelating agents. Initially no cell extract was added to the NMR tube.

The NMR instrument was carefully shimmed and tuned to prepare for the collection of NMR spectra. Cell extract was added to the tube to start the reaction. A series of NMR spectra were collected. Magnetisation vectors were calibrated according to fully relaxed magnetisation vectors for each species present in the collected NMR spectra.

The NMR spectra were then processed to determine the area under the peaks, which were identified by spiking with pure standards of the compound of interest. A value of 5 Hz was used for apodisation of FIDs, followed by subjecting the FIDs to Fourier transformation as implemented in NumPy to generate arrayed spectra in the frequency domain. The NMRPy package allowed for automatic phase correction, and the complete arrayed spectra were phase corrected in this manner. Automatic phase correction using the Levenberg-Marquadt algorithm minimises the total absolute area

underneath the peaks of the NMR spectra. A mixture of zero and first order phase correction was used for this investigation.

An added internal standard (triethyl phosphate for ^{31}P-NMR spectra) was used as a normalisation factor, facilitating the conversion of NMR peak areas to concentrations of the intermediates in question.

The data from a number of NMR runs, typically between 5 to 15 runs per enzyme, for the first two enzymes of glycolysis, are then globally fitted to a representative kinetic model. We made use of the well-known reversible Michaelis-Menten [34] equation and the generic reversible Hill equation, a generic systems biology rate equation [37–40].

The enzyme kinetic parameters for *E. coli* and *S. cerevisiae* extracts were first established at pH 7 to determine baseline values. The analysis was then repeated at pH 5.5 and pH 7.5, values reflecting the intracellular pH of starving and exponentially growing *E. coli* and *S. cerevisiae* cells respectively [24–26]. The PGI reaction can be studied separately from the PFK reaction by excluding ATP, this is due to the fact that the PGI reaction does not rely on any co-factors. However, the PFK catalysed reaction cannot be investigated in isolation. PGI will always be coupled with the PFK reaction in a crude cell free extract, as addition of its substrate F6P, also acts as a substrate for the revers reaction of PGI. Thus, as long as F6P is present, both PGI and PFK will be active. To study the PFK catalysed reaction we needed to adopt a modular approach and focused the investigation on a PGI-PFK coupled model, as the two reactions are inseparable in a cell-free extract.

3.5 Kinetic modelling

In this investigation, ODEs were used to model the upper glycolytic reactions. PySCeS, a software module developed in our research group, can be used for the construction, analysis and utilisation of models [82]. However NMRPy (the software module used for the analysis of NMR data) is written in Python 3.5, whereas PySCeS is still written in Python 2.7. The use of ODEs simplifies the interaction between different steps of the numerical analysis as it facilitates the use of Python 3.5 for all computation. The models constructed are used to simulate a number of processes required of living cells and have methods attached to perform tasks required in computational systems biology, such as time course simulation, which will be used extensively in this investigation.

The rate equation for the PGI catalysed reaction was as follows (concentrations are denoted in lowercase, whereas parameters are given in uppercase):

$$v_{PGI} = \frac{V_f\, e_t(\frac{g6p}{K_{m,G6P}})\,(1 - \frac{\frac{f6p}{g6p}}{K_{eq}})}{1 + (\frac{g6p}{K_{m,G6P}} + \frac{f6p}{K_{m,F6P}})} \tag{3.1}$$

The rate equation for the PFK catalysed reaction was as follows:

$$v_{PFK} = \frac{V_f\, e_t\, \alpha\, \beta\, (1 - \frac{\Gamma}{k_{eq}})\,(\alpha + \pi)^{h-1}\,(\beta + \rho)^{h-1}}{(\frac{1+\sigma^h}{1+\mu^{4h}\,\sigma^h}) + (\frac{1+\mu^{2h}\,\sigma^h}{1+\mu^{4h}\,\sigma^h})\,[(\alpha + \pi)^h + (\beta + \rho)^h] + (\alpha + \pi)^h\,(\beta + \rho)^h} \tag{3.2}$$

The model for PFK is parameterised with half-saturation constants [22,40], where:

$$\begin{aligned}
\alpha &= \frac{f6p}{K_{m,F6P}} \\
\beta &= \frac{fbp}{K_{m,FBP}} \\
\pi &= \frac{mgatp}{K_{m,MgATP}} \\
\rho &= \frac{mgadp}{K_{m,MgADP}} \\
\sigma &= \frac{mgatp}{K_{i,MgATP}} \\
\Gamma &= \frac{fbp \times mgadp}{f6p \times mgatp}
\end{aligned}$$

e_t denotes the total protein concentration. This is used as a normalisation factor for the collation of all datasets. This is due to the nature of long-term projects that utilise many datasets, collected on different days and using varying cell extracts, as new extract is made on each day of testing.
The Hill coefficient is given as h. Hill coefficients (h) were obtained from literature [22, 23].

The parameter μ gives an indication of the effect of an allosteric modifier on the enzyme (MgATP in the case of PFK). If μ=1 the modifier has no effect, μ >1 indicates activation and μ <1 inhibition.
Setting μ and σ to zero reverts the equation to the generic Hill rate equation with no allosteric modifiers [40].

As can be seen in equation (3.2), ATP, ADP and Mg^{2+} are present in kinetic assays done for this study. These compounds form complexes with each other in solution and need to be taken into account when modelling upper glycolysis. The rate equations for ATP, ADP and Mg^{2+} complex formation were as below:

$$V_{f1,ATP} = mg \times atp_{free} \tag{3.3}$$
$$V_{r1,ATP} = \frac{mgatp}{K_{eq,ATP}} \tag{3.4}$$
$$V_{f2,ATP} = mg \times mgatp \tag{3.5}$$
$$V_{r2,ATP} = \frac{mg2atp}{K_{eq2,ATP}} \tag{3.6}$$
$$V_{f1,ADP} = k \times mg \times adp_{free} \tag{3.7}$$
$$V_{r1,ADP} = k \times \frac{mgadp}{K_{eq,ADP}} \tag{3.8}$$
$$V_{f2,ADP} = k \times mg \times mgadp \tag{3.9}$$
$$V_{r2,ADP} = k \times \frac{mg2adp}{K_{eq2,ADP}} \tag{3.10}$$

Where, k is 1 and mg represents the concentration of Mg^{2+}.

Equations for the adenylate kinase reactions was also incorporated into the model as we performed the experiments with cell-free extracts, where these reactions would be active and play a role in ATP and ADP concentrations. The equations use to model the adenylate kinase reactions are as follows:

$$v_{ATP} = k_{f,ATP} \times mgatp - k_{r,ATP} \times mgadp \times P_i \tag{3.11}$$

$$v_{ADP} = k_{f,ADP} \times mgadp - k_{r,ADP} \times P_i \tag{3.12}$$

$$v_{ADP} = mgadp - \frac{P_i}{K_{eq,ADP}} \tag{3.13}$$

Where P_i was free phosphate concentration, k_f was the forward rate constant, k_r was the reverse reverse constant and K_{eq} was the equilibrium constant.

All software used is written in Python and is open source, involving no licence fees. This enables the easy integration of various programmes and algorithms as well as the transfer of data between experiment and model, providing an interface supporting the work flow requisite for the project.

3.6 Data analysis and fitting

The Python programming language was employed for all computational and numerical analyses. For the construction of an e-lab book the Jupyter notebook (previously IPython) system for interactive computing was used. Jupyter allows for the representation of code, results and experimental annotations in a single notebook format and is easy to use and access as it runs as a server on your local computer [83].

NMRPy, a custom designed open source software developed by Eicher *et al.* was used for the processing of NMR spectra [22]. This was accomplished through the fitting of Lorentzian functions to NMR spectra peaks to determine the area under the peaks. This was then normalised to the internal TEP standard and converted to figures depicting concentration versus time, named progress-curves.

The Nelder-Mead simplex algorithm was used to estimate initial parameters by fitting of kinetic rate equations to progress-curves. Python packages were used to fit model simulations to data to obtain parameter values. The numerical aspect of the computation was done using the NumPy package and fitting was done using the `scipy.optimize` package. Matplotlib was used to generate all plots presented in this thesis.

3.7 Identifiability analysis

Identifiability analysis also implements the Nelder-Mead algorithm to generate profile-likelihood figures for each fitted parameter. The identifiability of all parameters can be tested in this manner. The `scipy.optimize` package was used for identifiability analysis, and profile-likelihoods were generated using matplotlib.

Profile-likelihoods were generated by varying the parameter of interest whilst keeping all the other fitted parameters fixed and refitting. It is an iterative approach and was repeated several times to produce a range of SSR/SSR_0 (a ratio of newly calculated sum-squared residuals for each variation of the parameter against the original fit's

sum-squared residuals) values that, when graphically represented, give an indication of parameter identifiability. Confidence intervals for each parameter were calculated to assess the identifiability of parameters. Confidence intervals were calculated using the percent point function, the inverse of a cumulative distribution function. This function was used to calculate the α quantiles of the χ^2 distribution. The values obtained for the α quantiles corresponded to the 95% confidence interval bounds. This is the form of confidence interval as calculated by Raue *et al.*, which was briefly discussed in Section 2.2.4, namely the finite sample confidence interval method [63]. If the confidence intervals are infinite, or the valley of the profile-likelihood is perfectly flat, parameters are said to be non-identifiable. The standard deviation between datasets is taken into account for the identifiability analysis [63–65].

All scripts used for the analysis of the data are presented in the appendix.

Chapter 4

Results

As outlined in Section 1.2, the aim of this work was to determine the effects of pH changes on the kinetic parameters of the upper glycolytic enzymes PGI and PFK in *E. coli* and *S. cerevisiae*. The rationale was to determine whether pH changes need to be incorporated into kinetic models for systems biology, as intracellular pH has been shown to vary under some conditions [17–20, 25, 44].

This chapter provides a summary of the results obtained. Experiments were conducted under standardised conditions as close to the physiological, *in vivo* situation as possible [22], with the only variation between experiments being the pH of the assay mixture.

First, we show the fitting of kinetic models to experimental NMR time courses in order to determine the kinetic parameters (as explained in Section 3.6). Next, profile-likelihoods were generated for each of the parameters, as described in Section 3.7, to investigate the identifiability of these parameters and obtain a confidence interval for their values.

Due to the extent of data, only one example (of all model parameters) for each enzyme investigated will be shown in detail. To summarise, all parameters that could be determined are presented in tables and their dependence on pH is illustrated using graphs (Section 4.3). Kinetics and parameters that were fitted for are as stated in Section 3.5.

4.1 Parameter estimation

The PGI catalysed reaction was assayed in isolation, this can be accomplished in cell-free extracts because the reaction does not require any additional co-factors. Addition of G6P or F6P will thus allow the reaction to proceed (in the forward or reverse reaction, respectively), but adjacent reaction will not occur due to lack of co-factors. The situation for PFK, however, is different as addition of its substrates F6P and ATP to a cell-free extract will immediately also cause the PGI reaction to concomitantly proceed in the reverse direction; PFK therefore cannot be studied in isolation.

Thus, a modular approach was implemented and the PGI-PFK coupled reaction module was assayed. For parameter estimation, the kinetic parameters for PGI were determined first. These values were then entered into the PGI-PFK combined model, which allowed the estimation of the remaining PFK kinetic parameters by fitting this model to all PGI and PGI-PFK datasets.

4.1.1 PGI

The upper glycolytic enzymes of *E. coli* were characterised first, for experimental protocol optimisation. To establish baseline measurements the enzymes were initially assayed at neutral pH (pH 7), followed by assays conducted at pH 5.5 and pH 8, corresponding to starving and exponentially growing *E. coli* cells respectively.

A representative example of a ^{31}P-NMR time course of the PGI-PFK coupled reaction module can be seen in Figure 4.1.

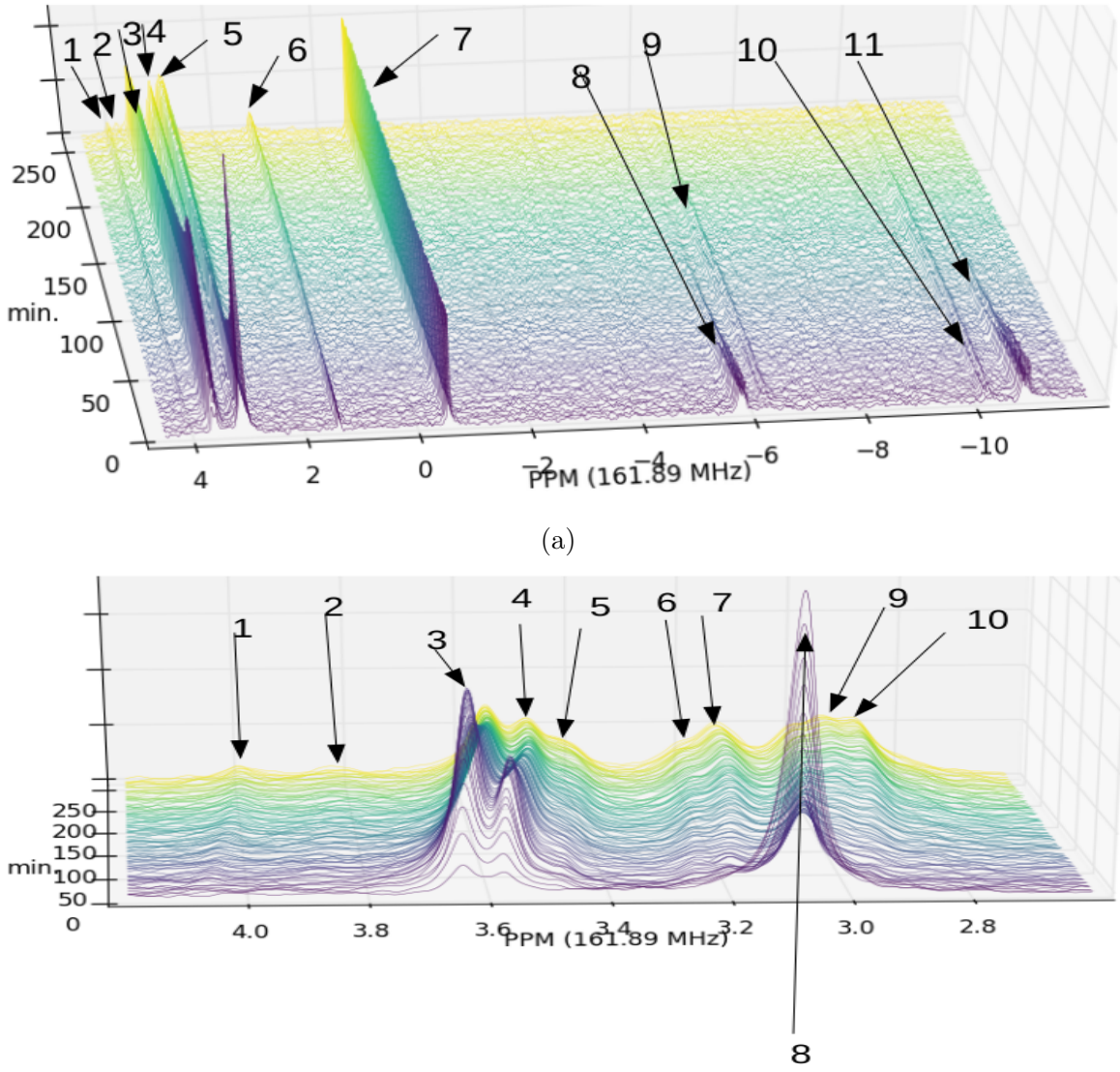
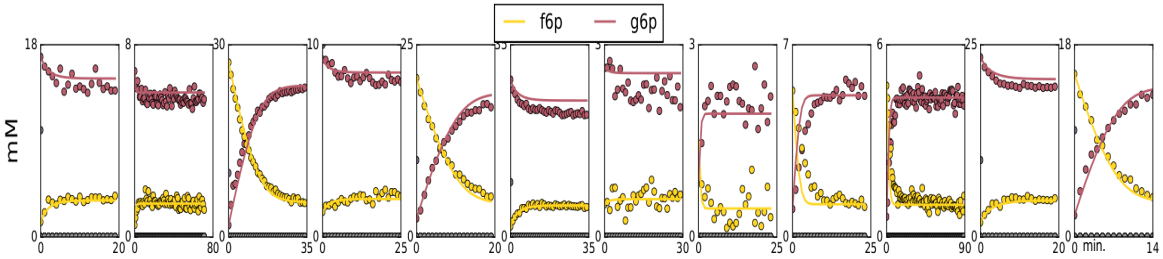


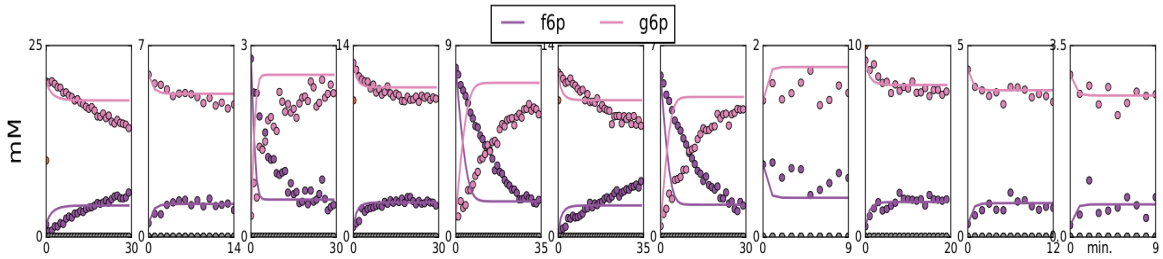
Figure 4.1: (a.) An example of a ^{31}P-NMR time course of the PGI-PFK coupled reaction module. Transients are presented in array format and individual transients are 2.4 minutes apart. Peaks are indicated by numbers representative of the species present. FBP: 1, 2, 4; G6P: 3; F6P: 5; phosphate: 6; TEP: 7; ATP: 8, 9; ADP: 10, 11. (b.) An expanded view of the sugar phosphate area (4.2 to 2.7 ppm). FBP-α: 1, 2, 10; FBP-β: 6, 7, 9; G6P-β: 3; G6P-α: 4, 5; F6P: 8.

E. coli

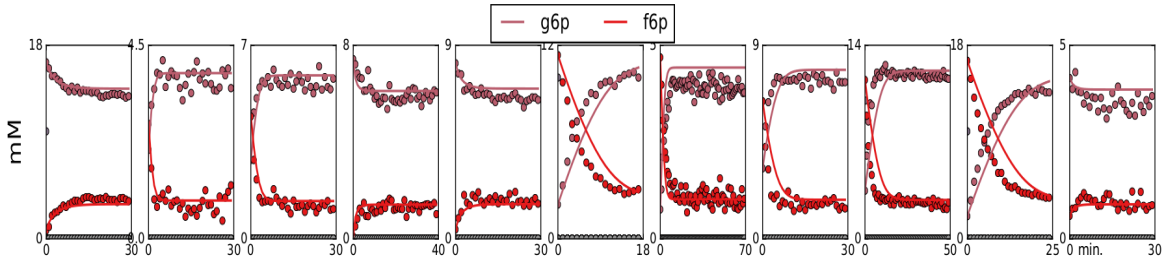
PGI catalyses the reaction from G6P to F6P. The enzyme was characterised in both directions, by starting with varying concentrations of either substrate (G6P) or product (F6P). Figure 4.2 represents the NMR progress-curves and model fits of PGI assays performed at pH 7, pH 5.5 and pH 8.



(a) pH 7.



(b) pH 5.5.

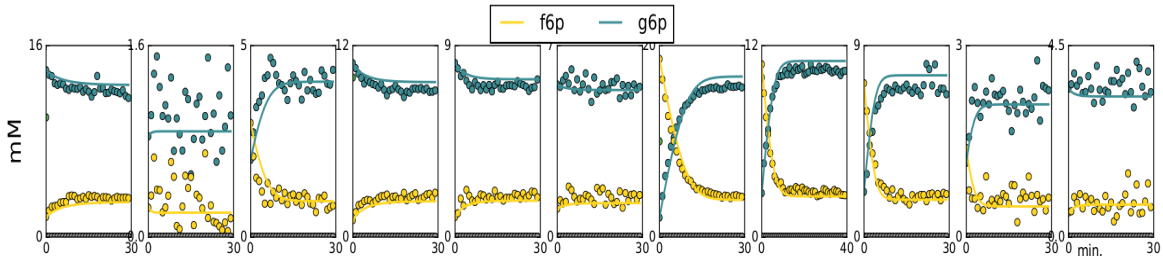


(c) pH 8.

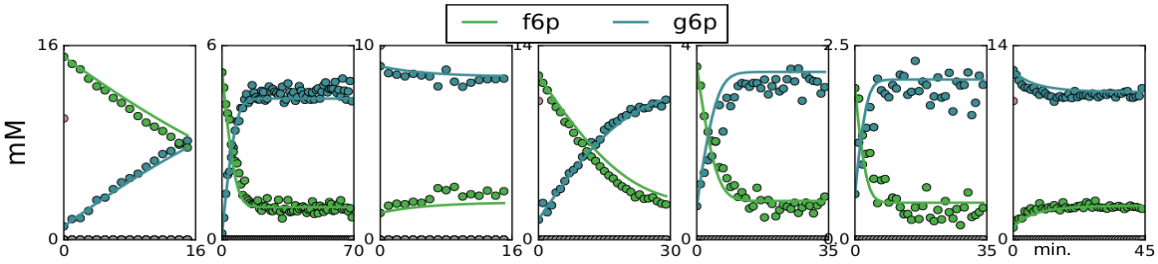
Figure 4.2: NMR progress-curves of concentration versus time for the PGI catalysed reaction in *E. coli* studied at (a) pH 7, (b) pH 5.5 and (c) pH 8. Dots are experimental data points and lines are model simulations with the fitted parameters. Initial concentrations of G6P and F6P were varied from 3 mM to 25 mM. Results were obtained by performing independent fits at each of the pH values studied, fitting to all relevant datasets simultaneously.

S. cerevisiae

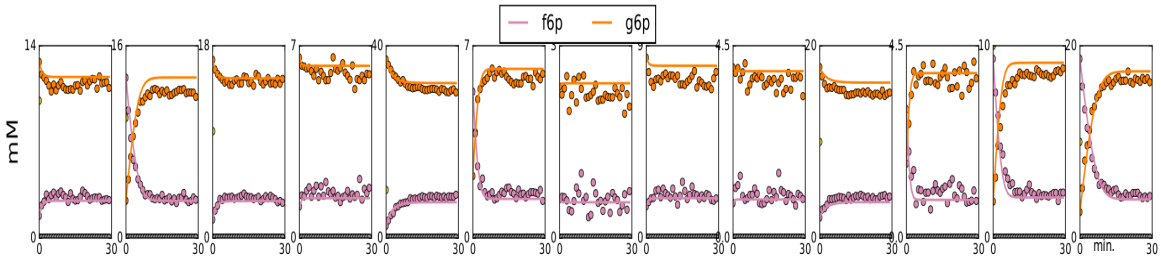
The experimental protocol, optimised in *E. coli*, was applied to characterise the PGI enzyme in *S. cerevisiae*. The results are displayed in Figure 4.3.



(a) pH 7.



(b) pH 5.5.



(c) pH 8.

Figure 4.3: NMR progress-curves for the PGI catalysed in reaction *S. cerevisiae*. Plot attributes are as in Figure 4.2. Initial concentrations were varied from 3 mM to 40 mM.

4.1.2 PFK

PFK catalyses the reaction from F6P to F-1,6-BP. This reaction utilises ATP as phosphate donor and produces ADP as product. The reaction is also allosterically inhibited by ATP. ATP, ADP and AMP exist as a moiety conserved cycle. AMP could not be quantified from the NMR spectra.

E. coli

Figures 4.4 to 4.6 show NMR progress-curves and fitted model simulations for the PGI-PFK reaction module in *E. coli*. Enzymes assays, as outlined in Section 3.3, were performed by incubating cell-free extract along with the substrates for PFK, F6P and G6P. NMR spectroscopy was subsequently used to follow the bio-transformations effected by the enzymes present in the PGI-PFK module.

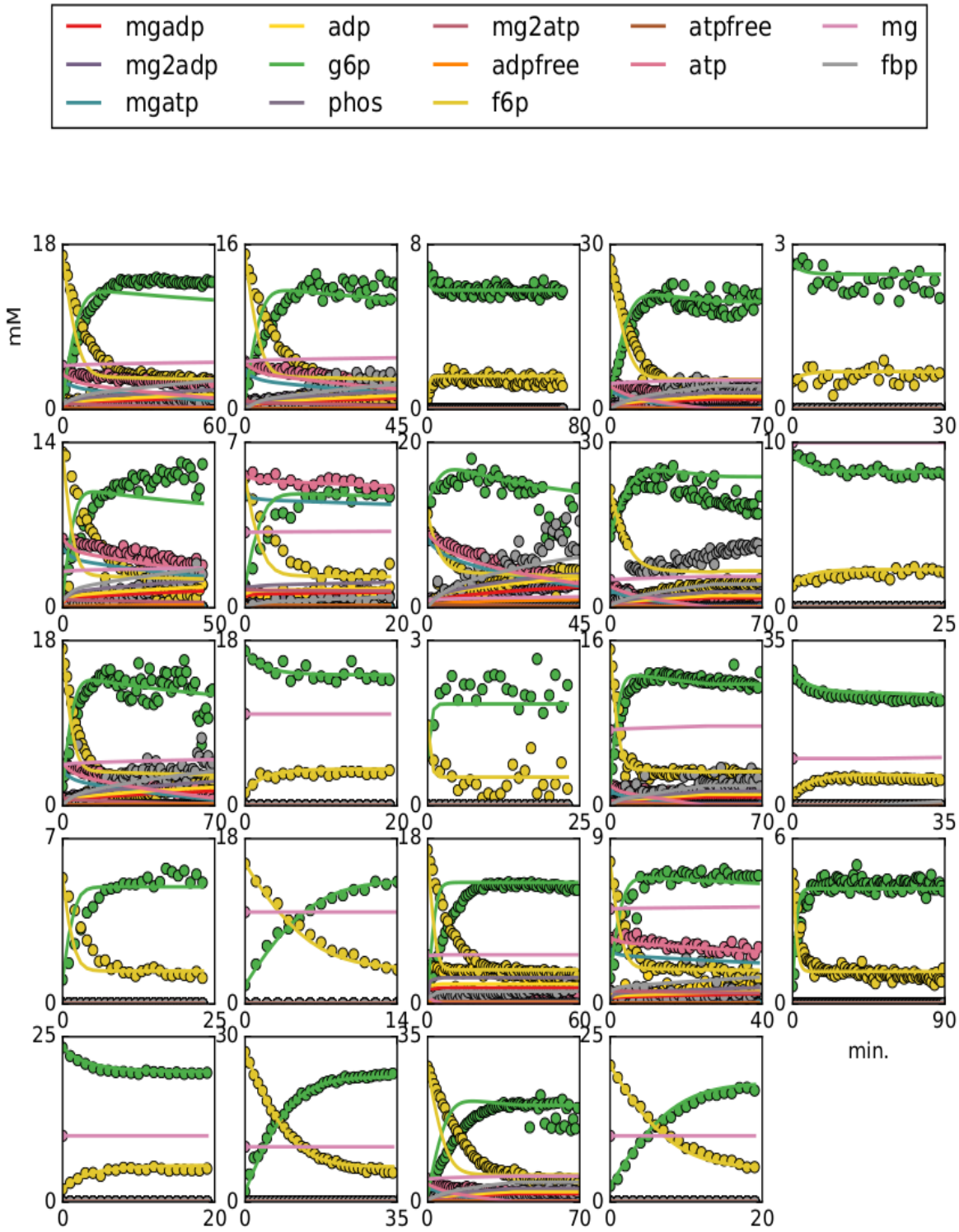


Figure 4.4: NMR progress-curves of concentration versus time for the PGI-PFK reaction module from *E. coli* studied at pH 7. Dots are experimental data points and lines are model simulations with the fitted parameters. Initial concentrations of F6P were varied from 3 mM to 40 mM and ATP from 5 mM to 10 mM. The final parameters were obtained by global fitting to all datasets simultaneously.

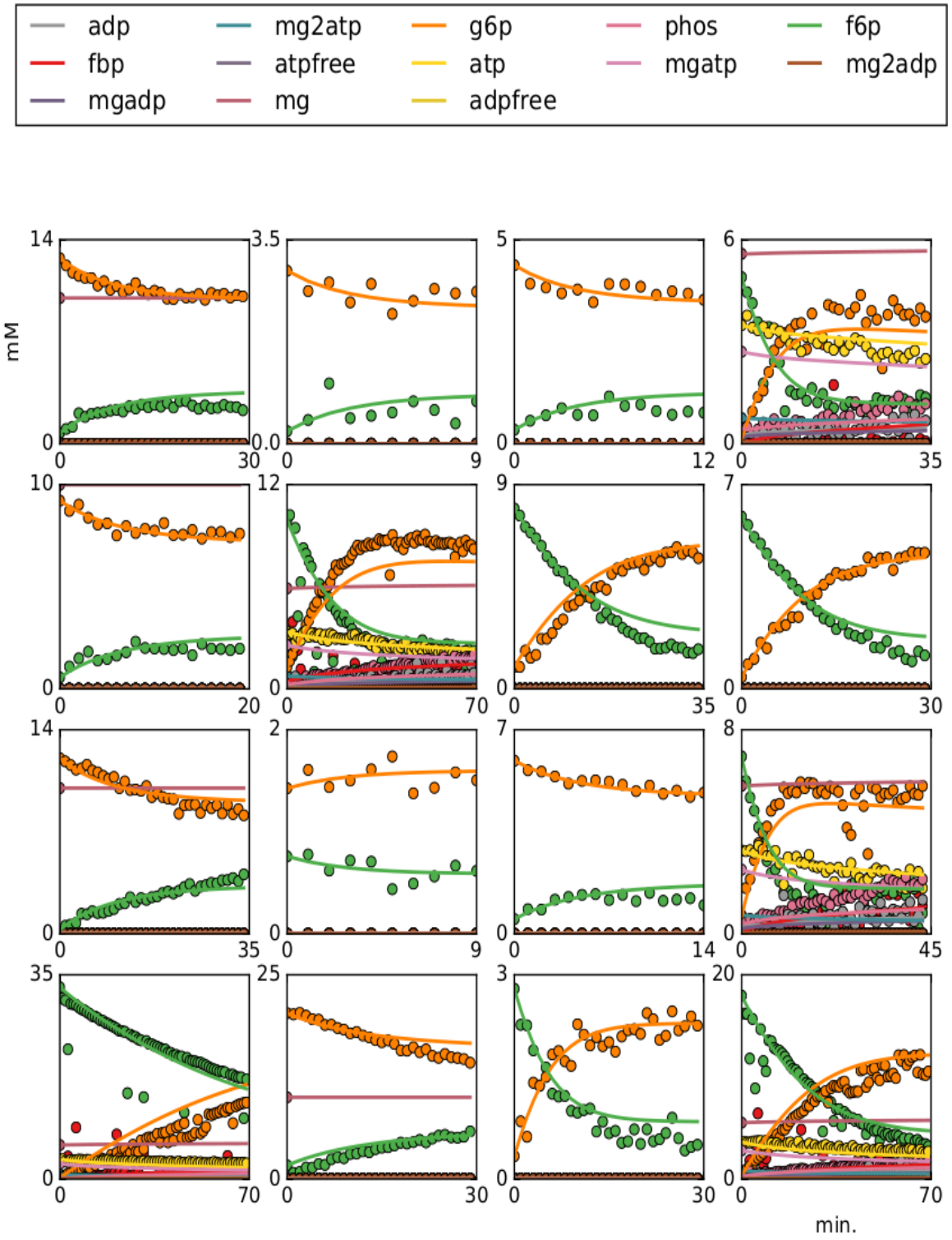


Figure 4.5: The PGI-PFK reaction module from *E. coli* studied at pH 5.5. Figure attributes are as in Figure 4.4. Initial concentrations of F6P were varied from 3 mM to 25 mM and ATP from 5 mM to 10 mM.

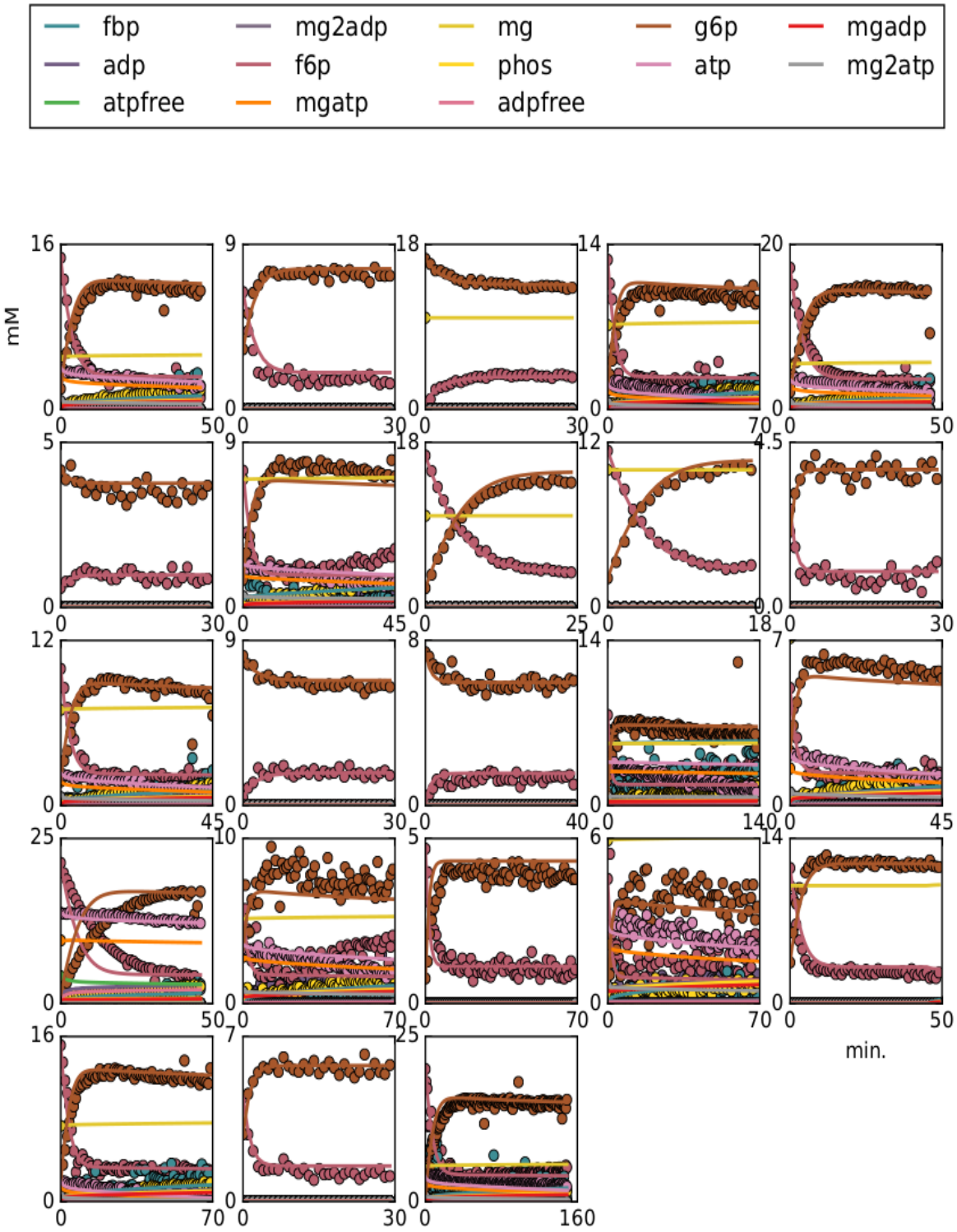


Figure 4.6: The PGI-PFK reaction module from *E. coli* studied at pH 8. Figure attributes are as in Figure 4.4. Initial concentrations of F6P were varied from 3 mM to 25 mM and ATP from 5 mM to 10 mM.

S. cerevisiae

Figures 4.7 to 4.9 show NMR progress-curves and fitted model simulations for the PGI-PFK reaction module in *S. cerevisiae*.

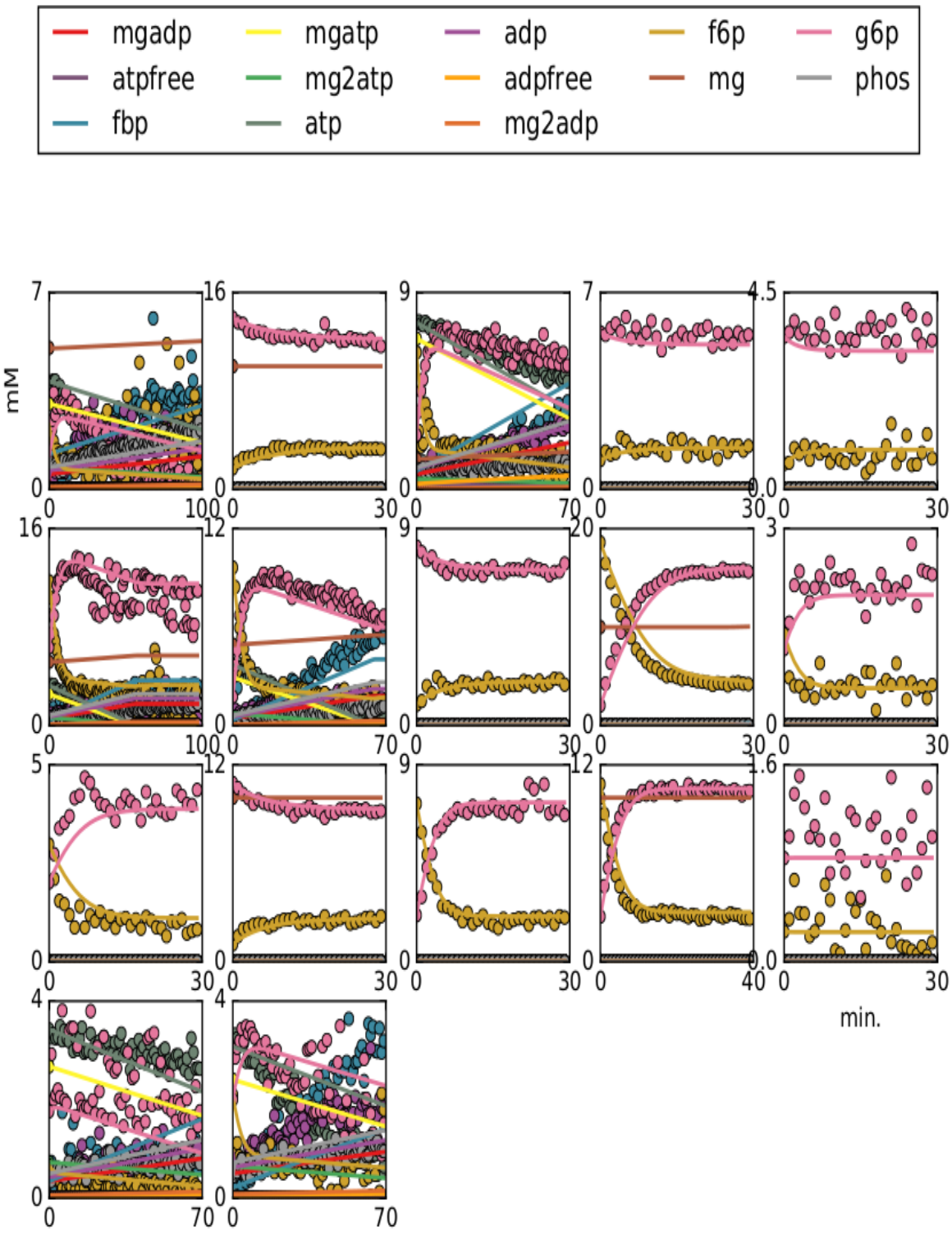


Figure 4.7: The PGI-PFK reaction module from *S. cerevisiae* studied at pH 7. Figure attributes are as in Figure 4.4. Initial concentrations of F6P were varied from 3 mM to 20 mM and ATP from 5 mM to 10 mM.

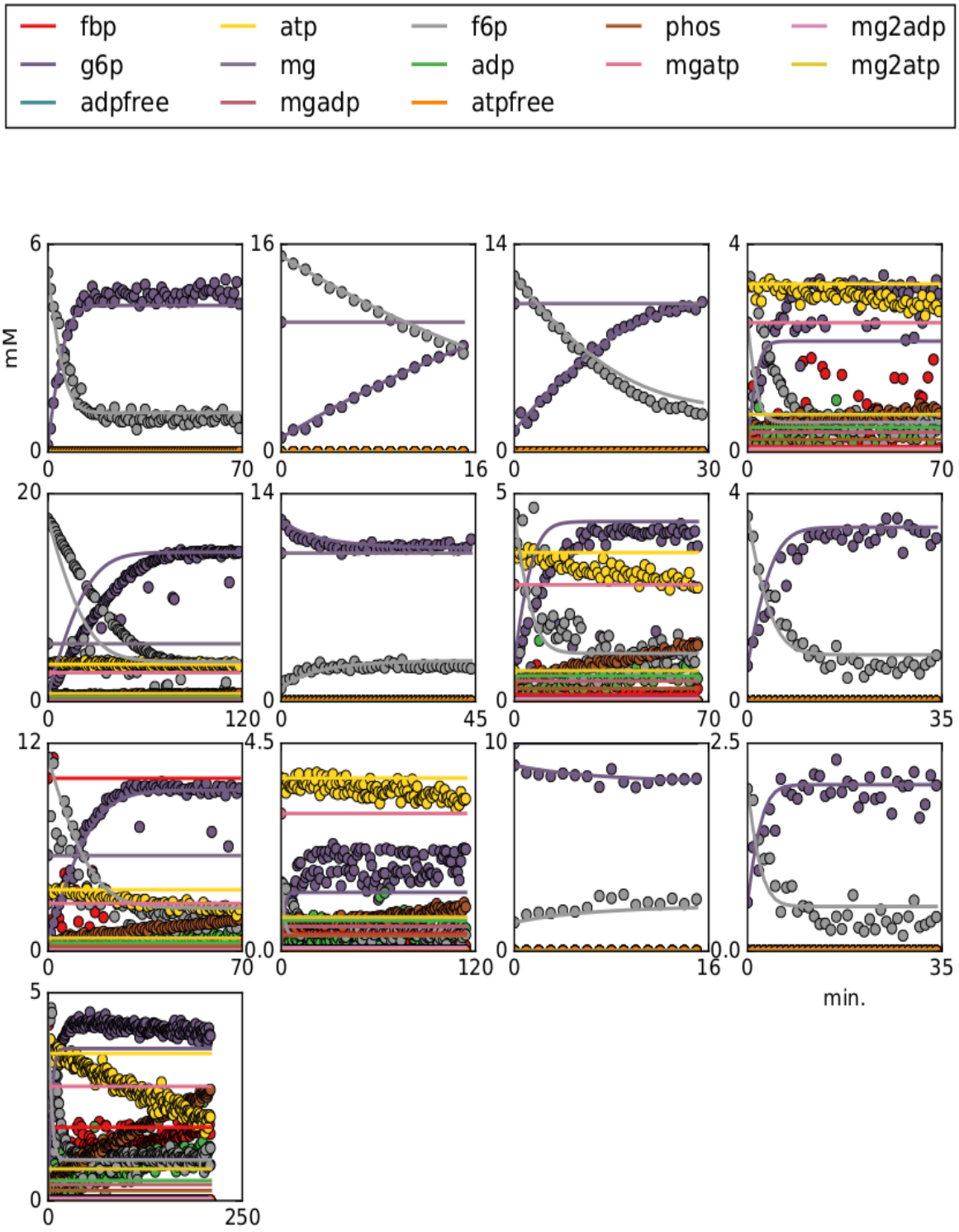


Figure 4.8: The PGI-PFK reaction module *S. cerevisiae* studied at pH 5.5. Figure attributes are as in Figure 4.4. Initial concentrations of F6P were varied from 3 mM to 20 mM and ATP from 5 mM to 10 mM.

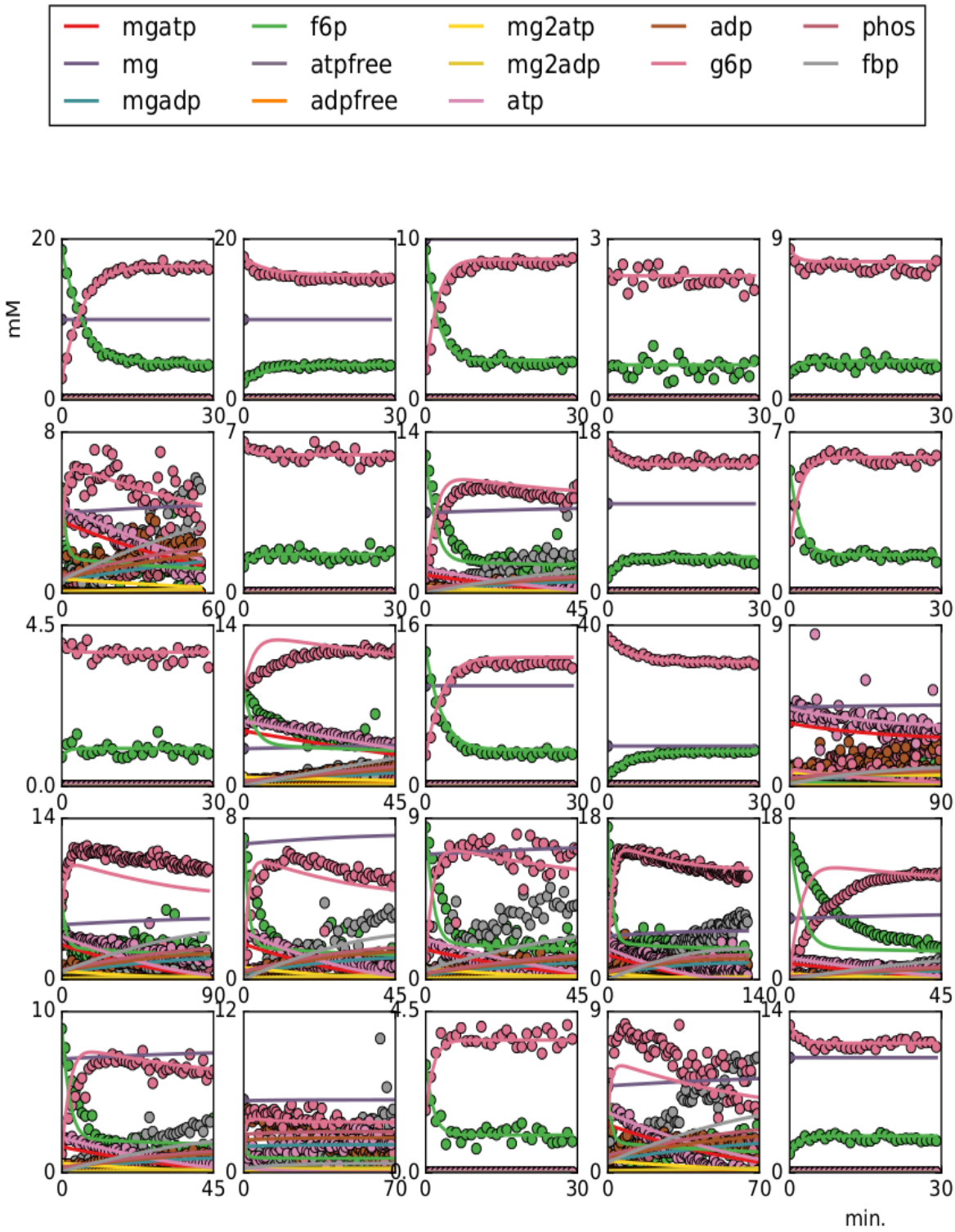


Figure 4.9: The PGI-PFK reaction module *S. cerevisiae* studied at pH 8. Figure attributes and initial concentrations are as in Figure 4.4.

4.2 Parameter identifiability

Identifiability analysis is important as mathematical approaches are used in the estimation of parameters. In a model of a biological system, reactions are abstracted

as mathematical equations. These equations might contain correlations between parameters that might not be easily observable, and may lead to incorrect parameter estimation. Furthermore, multiple solutions could exist for the system of equations. Thus we needed to ensure that the parameters that are estimated are unique, and hence, accurate.

Identifiability analysis was accomplished through use of the Nelder-Mead simplex algorithm to generate profile-likelihoods for each parameter individually. Each fitted parameter was varied in turn, both up and down from its fitted value. All the other parameters were then re-fitted at each iteration. At each variation of the parameter being tested a new fit was done and the sum of the squared residuals (SSR) was calculated. This SSR was then divided by the SSR of the original fit (SSR_0), and each variation in the tested parameter is weighted in this manner. This allowed for plotting these weighted values, resulting in a profile-likelihood distribution for each parameter. Confidence intervals were calculated by determining a certain threshold value, based on the α quantile of the χ^2 distribution as discussed in Section 3.1.6. Any SSR/SSR_0 values under this threshold is considered as falling within the confidence interval upper and lower bounds. Here we used 95% confidence intervals, meaning that 95% of the time the fitted parameter would be expected to be found within the confidence region. Fully identifiable parameters are indicated by profile-likelihoods with finite confidence intervals and definite global minima. Practically identifiable parameters are indicated as having finite confidence intervals and are dependent on the quality and amount of data. Practical non-identifiability can be remedied by collecting more, high quality data. Structural non-identifiability is depicted as profile-likelihoods displaying perfectly flat valleys and arise due to structural redundancies in the model itself or functional relationships between model parameters. To resolve structural non-identifiability the model needs to be re-evaluated and compared with other possible models that might eliminate redundancies and functional relationships between parameters of the model. For more detail on non-identifiability see Section 2.2.3

E. coli

A total of 102 parameters could be estimated using the data fitting process as detailed in Section 3.7. For the sake of not overwhelming the reader with figures, only the profile-likelihoods for PGI at pH 7 and PFK at pH 5.5 in *E. coli* will be shown as representative examples of profile-likelihood distributions. All results obtained from the identifiability analysis will be summarised in Sections 4.3.

4.2.1 PGI

Figure 4.10 depicts the profile-likelihood distributions of the kinetic parameters of PGI in *E. coli* at pH 7. As can be seen from Figure 4.10 all parameters were uniquely identifiable, both structurally and practically. This can be seen from the finite confidence region as well as the definite minima evident at the estimated parameter value.

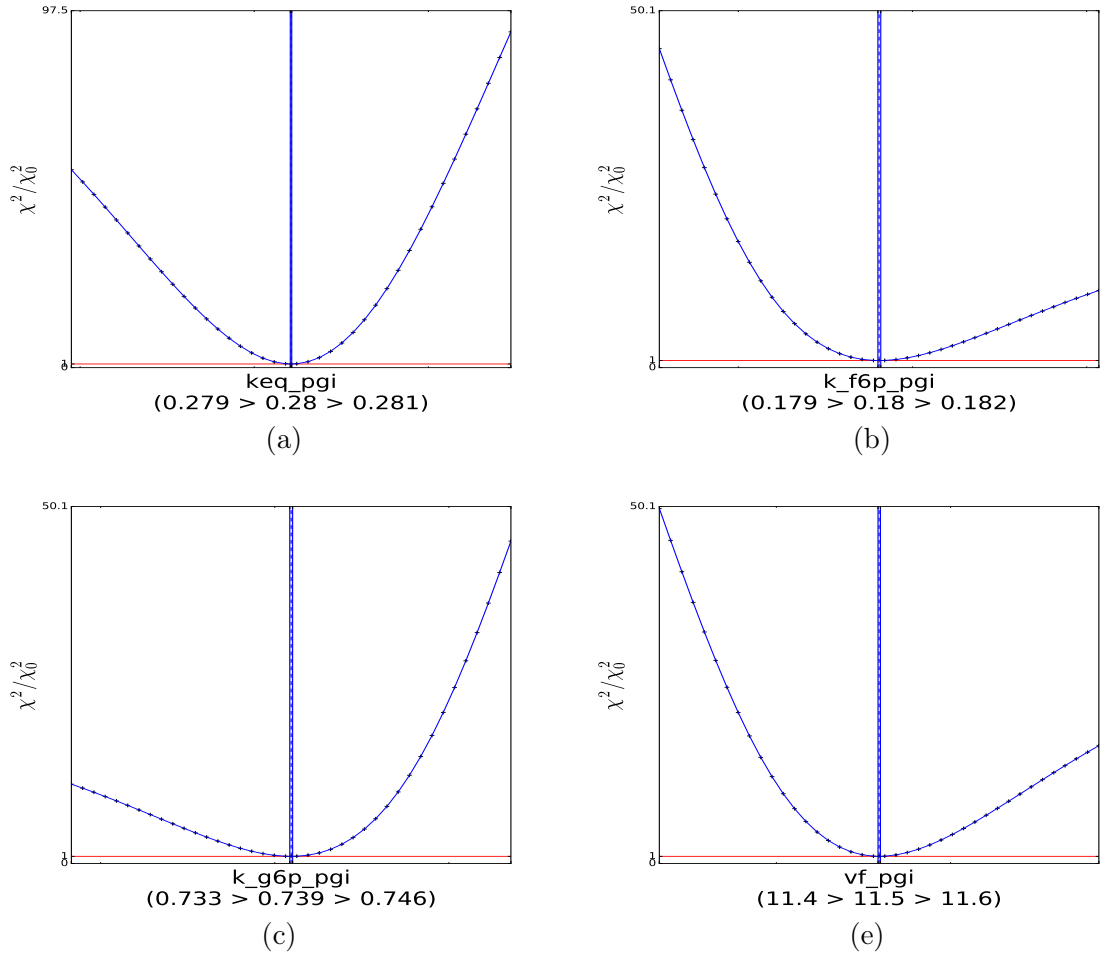


Figure 4.10: Profile-likelihood distributions of the kinetic parameters of PGI in *E. coli* at pH 7. Blue crosses (+) indicate the SSR/SSR_0 values. The blue line joining them is an interpolating spline indicating the curvature of the valley of the likelihood distributions. The horizontal red line depicts the threshold value for the 95% confidence intervals and the vertical dashed blue line marks the originally fitted parameter. The vertical solid blue lines visualise the upper and lower bounds of the 95% confidence intervals. On the y-axis is SSR/SSR_0, and on the x-axis is shown the value of the fitted parameter, as well as the values of the upper and lower bounds of confidence intervals. The profile likelihood distribution for (a) K_{eq}, (b) $K_{m,F6P}$ (mM), (c) $K_{m,G6P}$ (mM) and (d) V_f (μ mol/min/mg protein).

4.2.2 PFK

The results of the identifiability analysis for the kinetic parameters of PFK are presented in Figure 4.11. This includes the parameters taking into account the binding between ATP, ADP and Mg^{2+}. For the entire set of equations representing the PFK reaction module, 13 parameters for each microorganism could be determined from data fitting.

As can be seen in figure 4.10, not all parameters were identifiable. In this case it was an issue of structural, rather than practical identifiability, as the addition of more experimentally obtained datasets had no effect on the profile-likelihoods of non-identifiable parameters. To address this, the model will have to be adjusted. This topic will be discussed further in Chapter 5.

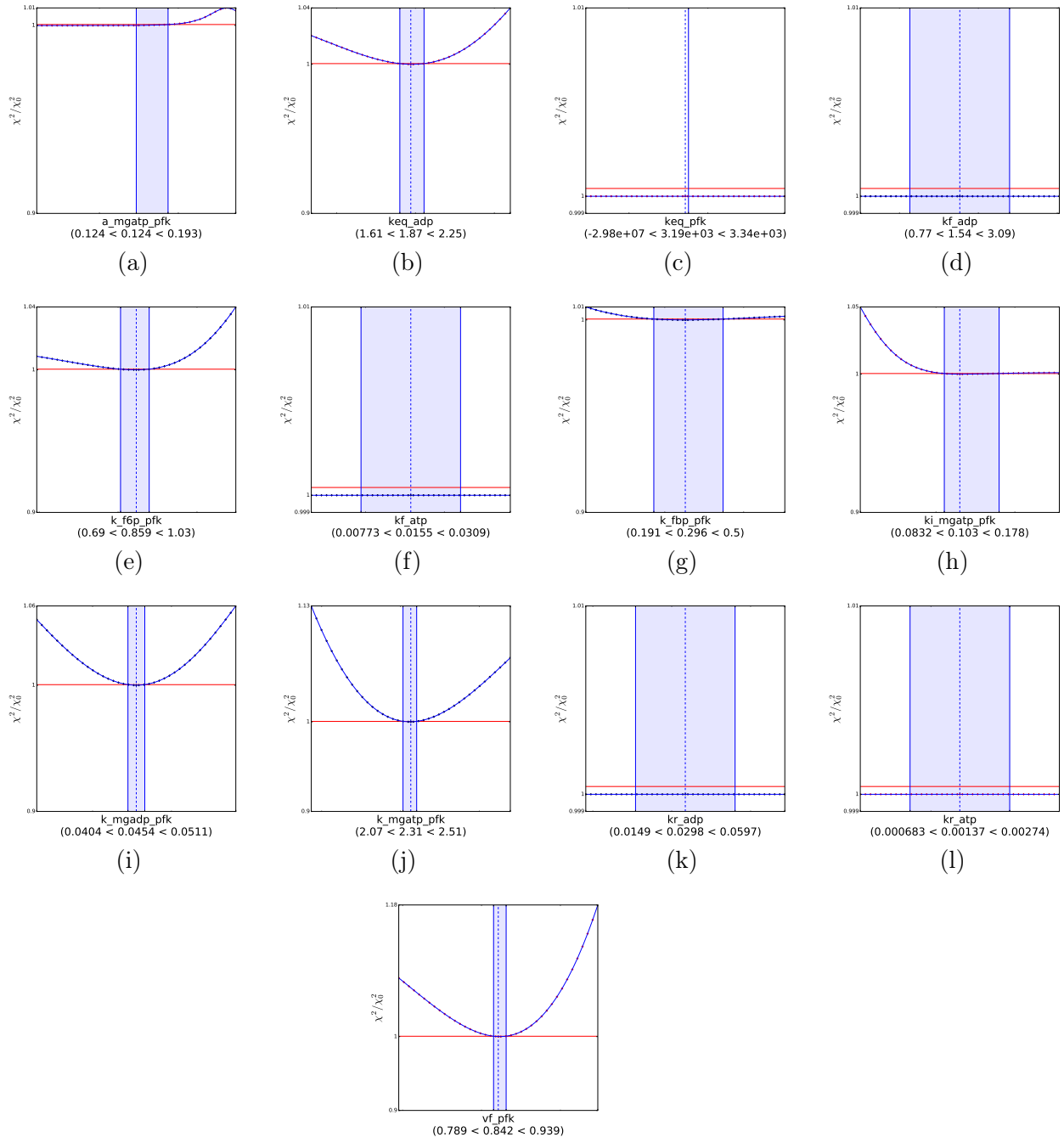


Figure 4.11: Profile-likelihood distributions of the kinetic parameters of PFK in *E. coli* at pH 5.5. Figure attributes are as in Figure 4.10. The profile-likelihoods are arranged as follows: (a) μ_{MgATP}, (b) $K_{eq,ADP}$, (c) $K_{eq,PFK}$, (d) $k_{f,ADP}$, (e) $K_{m,F6P}$, (f) $k_{f,ATP}$, (g) $K_{m,FBP}$, (h) $K_{i,MgATP}$, (i) $K_{m,MgADP}$, (j) $K_m, MgATP$, (k) $k_{r,ADP}$, (l) $k_{r,ATP}$(mM), (m) V_f. These parameters are as in the equations in Section 3.5.

4.3 Summary of final parameters

In this section the final results for all parameters, under all pH conditions investigated, will be presented in a series of tables (obtained by global fitting of models to all datasets

simultaneously). It is important to note that fitting of models was performed on all datasets at a given pH individually. Parameters presented here are for the equations making up the model as outlined in Section 3.3. Where available, literature values are also reported. Literature values were obtained from the BRENDA enzyme database. V_f values were not included as in the current study we worked in lysates, whereas literature V_f values were determined with purified enzymes.

4.3.1 *E. coli*

PGI

Tables 4.1 to 4.3 show the parameter values obtained for the PGI catalysed reaction in *E. coli*.

pH 5.5						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	5.18	10.4	2.6	Yes	Yes	
K_{eq}	0.367	0.37	0.364	Yes	Yes	
$K_{m,G6P}$(mM)	2.78	2.87	2.71	Yes	Yes	
$K_{m,F6P}$(mM)	1.18	1.21	1.16	Yes	Yes	

Table 4.1: Parameters and 95% confidence intervals for the PGI catalysed reaction in *E. coli* at pH 5.5.

pH 7						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	11.5	11.5	11.4	Yes	Yes	
K_{eq}	0.28	0.281	0.279	Yes	Yes	0.084 [22, 23]
$K_{m,G6P}$	0.739	0.746	0.733	Yes	Yes	0.28 [84], 0.084 [85]
$K_{m,F6P}$(mM)	0.18	0.182	0.179	Yes	Yes	0.147 [84], 0.2 [86]

Table 4.2: Parameters and 95% confidence intervals for the PGI catalysed reaction in *E. coli* at pH 7.

pH 8						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	5.06	10.1	2.52	Yes	Yes	
K_{eq}	0.262	0.264	0.259	Yes	Yes	
$K_{m,G6P}$(mM)	4.79	4.48	4.2	Yes	Yes	
$K_{m,F6P}$(mM)	1.77	1.85	1.7	Yes	Yes	

Table 4.3: Parameters and 95% confidence intervals for the PGI catalysed reaction in *E. coli* at pH 8.

PFK

Tables 4.4 to 4.6 show the parameter values obtained for the PFK catalysed reaction in *E. coli*.

pH 5.5						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	0.124	0.124	0.193	Yes	Yes	
$K_{eq}ADP$	1.87	2.25	1.61	Yes	Yes	
$K_{eq}PFK$	3.19e+03	3.34e+03	-2.98e+07	No	No	
$K_{m,F6P}$(mM)	0.859	1.03	0.69	Yes	Yes	
$k_{f,ADP}$(mM.s^{-}1)	1.54	3.09	0.77	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.0155	0.0309	0.00773	No	No	
$K_{m,FBP}$(mM)	0.296	0.5	0.191	Yes	Yes	
$K_{i,MgATP}$(mM)	0.103	0.17	0.0832	Yes	Yes	
$K_{m,MgADP}$(mM)	0.0454	0.0511	0.0404	Yes	Yes	
$K_{m,MgATP}$(mM)	2.31	2.51	2.07	Yes	Yes	
$k_{r,ADP}$(.s^{-}1)	0.0298	0.0598	0.0149	No	No	
$k_{r,ATP}$(.s^{-}1)	0.00137	0.00274	0.000683	No	No	
V_f (μ mol/min/mg protein)	0.842	0.939	0.789	Yes	Yes	

Table 4.4: Parameters and 95% confidence intervals for the PFK catalysed reaction in *E. coli* at pH 5.5.

pH 7						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	0.00478	0.00478	0.00481	Yes	Yes	0.3797 [22, 23]
$K_{eq}ADP$	1.64	1.72	1.57	Yes	Yes	
$K_{eq}PFK$	636	636	338	No	No	
$K_{m,F6P}$	0.000148	0.000154	0.000146	Yes	Yes	0.4174 [22, 23]
$k_{f,ADP}$(mM.s^{-}1)	0.0771	0.154	0.0386	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.00266	0.00533	0.00133	No	No	
$K_{m,FBP}$(mM)	5.49	75.4	2.44	Yes	Yes	
$K_{i,MgATP}$(mM)	1.46e-09	2.93e-09	7.32e-10	Yes	Yes	
$K_{m,MgADP}$(mM)	2.83	99.3	2.83	Yes	Yes	
$K_{m,MgATP}$(mM)	0.000226	0.000244	0.000224	Yes	Yes	0.5444 [22, 23], 0.1 [87], 0.21 [88]
$k_{r,ADP}$(.s^{-}1)	0.00139	0.00278	0.000694	No	No	
$k_{r,ATP}$(.s^{-}1)	1.57e-06	3.13e-06	7.83e-07	No	No	
V_f (μ mol/min/mg protein)	6.82	7.07	6.57	Yes	Yes	

Table 4.5: Parameters and 95% confidence intervals for the PFK catalysed reaction in *E. coli* at pH 7.

pH 8						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	5.27	10.4	5.27	Yes	Yes	
$K_{eq}ADP$	1.18	1.62	0.643	Yes	Yes	
$K_{eq}PFK$	1.34e+04	1.98e+04	6.82e+03	No	No	
$K_{m,F6P}$(mM)	0.000182	0.000223	0.000155	Yes	Yes	0.007 [89]
$k_{f,ADP}$(mM.s^{-}1)	0.0462	0.0925	0.0231	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.0403	0.0806	0.0201	No	No	
$K_{m,FBP}$(mM)	3.64e-07	7.67e-07	1.82e-07	Yes	Yes	
$K_{i,MgATP}$(mM)	1.8	4.02	0.608	Yes	Yes	
$K_{m,MgADP}$(mM)	10.5	13.7	5.32	Yes	Yes	
$K_{m,MgATP}$(mM)	0.000169	3.88	8.57e-05	Yes	Yes	0.008 [89], 0.11 [90], 0.2 [91]
$k_{r,ADP}$(.s^{-}1)	0.00687	0.0137	0.00343	No	No	
$k_{r,ATP}$(.s^{-}1)	3.01e-05	6.02e-05	1.5e-05	No	No	
V_f (μ mol/min/mg protein)	8.87	10.8	8.86	Yes	Yes	

Table 4.6: Parameters and 95% confidence intervals for the PFK catalysed reaction in *E. coli* at pH 8.

4.3.2 *S. cerevisiae*

PGI

Tables 4.7 to 4.9 show the parameter values obtained for the PFK catalysed reaction in *S. cerevisiae*.

pH 5.5						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	5.41	10.8	2.69	Yes	Yes	
K_{eq}	0.286	0.288	0.283	Yes	Yes	
$K_{m,G6P}$(mM)	0.821	0.888	0.762	Yes	Yes	
$K_{m,F6P}$(mM)	0.18	0.192	0.166	Yes	Yes	

Table 4.7: Parameters and 95% confidence intervals for the PGI catalysed reaction in *S.cerevisiae* at pH 5.5.

pH 7						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	5.29	10.6	2.65	Yes	Yes	
K_{eq}	0.25	0.258	0.243	Yes	Yes	0.31 [92]
$K_{m,G6P}$(mM)	1.52	3.61	1.21	Yes	Yes	0.3 - 1.5 [93], 0.167 [94], 1.4 [95]
$K_{m,F6P}$(mM)	0.12	0.158	0.0827	Yes	Yes	0.11 - 0.23 [93], 0.3 [95]

Table 4.8: Parameters and 95% confidence intervals for the PGI catalysed reaction in *S.cerevisiae* at pH 7.

pH 8						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
V_f (μ mol/min/mg protein)	12.2	12.4	12	Yes	Yes	
K_{eq}	0.283	0.284	0.281	Yes	Yes	
$K_{m,G6P}$(mM)	5.67	5.78	5.55	Yes	Yes	
$K_{m,F6P}$(mM)	2.78	2.86	2.71	Yes	Yes	

Table 4.9: Parameters and 95% confidence intervals for the PGI catalysed reaction in *S.cerevisiae* at pH 8.

PFK

Tables 4.10 to 4.12 show the parameter values obtained for the PFK catalysed reaction in *S. cerevisiae*.

pH 5.5						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	1.49	2.99	-139	Yes	No	
$K_{eq}ADP$	0.635	0.768	0.517	Yes	Yes	
$K_{eq}PFK$	2.77e+03	5.55e+03	1.39e+03	No	No	
$K_{m,F6P}$(mM)	0.44	0.879	0.22	Yes	Yes	
$k_{f,ADP}$(mM.s^{-}1)	0.00363	0.00727	0.00182	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.000687	0.00137	0.000344	No	No	
$K_{m,FBP}$(mM)	6.19	12.4	3.09	Yes	Yes	
$K_{i,MgATP}$(mM)	0.413	100	0.206	Yes	Yes	
$K_{m,MgADP}$(mM)	1.56	3.13	0.782	Yes	Yes	
$K_{m,MgATP}$(mM)	0.0377	26.5	0.0334	Yes	Yes	
$k_{r,ADP}$(.s^{-}1)	0.00204	0.00407	0.00102	No	No	
$k_{r,ATP}$(.s^{-}1)	0.00109	0.00217	0.000543	No	No	
V_f (μ mol/min/mg protein)	1.99e-09	3.97e-09	9.93e-10	Yes	Yes	

Table 4.10: Parameters and 95% confidence intervals for the PFK catalysed reaction in *S.cerevisiae* at pH 5.5.

pH7						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	0.648	15.9	-18.6	Yes	No	
$K_{eq}ADP$	1.47	2.11	1.31	Yes	Yes	
$K_{eq}PFK$	3.56e+03	7.11e+03	1.07e+03	No	No	
$K_{m,F6P}$(mM)	1.25e-05	0.00245	1.25e-05	Yes	Yes	0.1 [92]
$k_{f,ADP}$(mM.s^{-}1)	0.000411	0.000822	0.000205	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.0394	0.0789	0.0197	No	No	
$K_{m,FBP}$(mM)	5.69	11.4	0.0277	Yes	Yes	0.111 [92]
$K_{i,MgATP}$(mM)	1.4	2.79	0.057	Yes	Yes	
$K_{m,MgADP}$(mM)	4.29	15.2	0.0368	Yes	Yes	
$K_{m,MgATP}$(mM)	5.86e-05	0.271	3.25e-05	Yes	Yes	0.65 [92]
$k_{r,ADP}$(.s^{-}1)	0.00123	0.00246	0.000615	No	No	
$k_{r,ATP}$(.s^{-}1)	0.0087	0.0174	0.00435	No	No	
V_f (μ mol/min/mg protein)	0.148	0.295	0.0738	Yes	Yes	

Table 4.11: Parameters and 95% confidence intervals for the PFK catalysed reaction in *S.cerevisiae* at pH 7.

pH 8						
Parameter (units)	Fitted parameter value	Upper confidence limit	Lower confidence limit	Structural identifiability	Practical identifiability	Literature value
μ_{MgATP}	1.32	114	1.31	Yes	Yes	
$K_{eq}ADP$	1.47	2.1	1.21	Yes	Yes	
$K_{eq}PFK$	9.61e+03	1.92e+04	280	No	No	
$K_{m,F6P}$(mM)	0.711	0.737	0.483	Yes	Yes	
$k_{f,ADP}$(mM.s^{-}1)	6.33e-06	1.27e-05	3.17e-06	No	No	
$k_{f,ATP}$(mM.s^{-}1)	0.0187	0.0374	0.00936	No	No	
$K_{m,FBP}$(mM)	4.04	6.55	3.37	Yes	Yes	
$K_{i,MgATP}$(mM)	0.261	0.577	0.261	Yes	Yes	
$K_{m,MgADP}$(mM)	0.00497	0.0075	0.00467	Yes	Yes	
$K_{m,MgATP}$(mM)	0.00143	0.00151	0.000979	Yes	Yes	
$k_{r,ADP}$(.s^{-}1)	0.162	0.325	0.0812	No	No	
$k_{r,ATP}$(.s^{-}1)	0.0139	0.0277	0.00693	No	No	
V_f (μ mol/min/mg protein)	0.345	0.691	0.173	Yes	Yes	

Table 4.12: Parameters and 95% confidence intervals for the PFK catalysed reaction in *S.cerevisiae* at pH 8.

We now present the final results obtained for all parameters in graphical form, illustrating the relationship between individual parameters at the different pH values investigated.

The parameter versus pH data for PGI will first be presented, followed by the data for PFK.

From the plots of parameter versus pH displayed in this section, it is evident that pH plays a role where kinetic parameters are concerned. Parameter values varied as pH varied and in some cases clear trends could be seen in parameter values as pH changed. In most cases these changes in parameter values were not consistent across the microorganisms and enzymes studied, however, some consistencies could be identified. The consistencies, where observed, existed as a general trend, rather than in the values themselves.

Figures 4.12 and 4.13 present an example of this for the PGI catalysed reaction in terms of $k_{m,G6P}$. For both microorganisms these values changed in the same way when parameter values at pH 5.5 and 8 (the two conditions evaluated) are compared to the values at baseline (pH 7). In both cases the dissociation constant for substrate increased at pH 5.5 and pH 8 from the value obtained at pH 7, meaning the affinity of the enzyme for its substrates was decreased at the conditions investigated. PFK for both organisms showed the same decrease in affinity for its substrate at pH 5.5 and pH 8, as seen in Figures 4.14(b) and 4.15(b). The inverse was seen in both microorganism for FBP, a product of PFK, where the $K_{m,FBP}$ was highest at pH 7 (Figures 4.14(c) and 4.15(c)).

For the PFK catalysed reaction a general upward trend could be observed for the forward rate, V_f, in both microorganisms (Figures 4.14 and 4.15). The lowest rate was

observed at pH 5.5, followed by a steep increase in V_f at pH 7 and the highest rate reached at pH 8.

In *E. coli* both dissociation constants of the PGI catalysed reaction, $K_{m,G6P}$ and $K_{m,F6P}$, were lowest at pH 7, meaning the enzyme had the highest affinity for both its substrate and product at pH 7. The V_f increased as pH increased, along with the same trend in PFK, this resulted in an increased rate for the PGI-PFK system as pH increased from 5.5 to 8 (Figures 4.12 and 4.14). In *S. cerevisiae* V_f was at its highest at pH 7 (Figure 4.14). Taken with the values for V_f in *S. cerevisiae* the highest rate for the PGI-PFK system was at pH 8 (Tables 4.6 and 4.12).

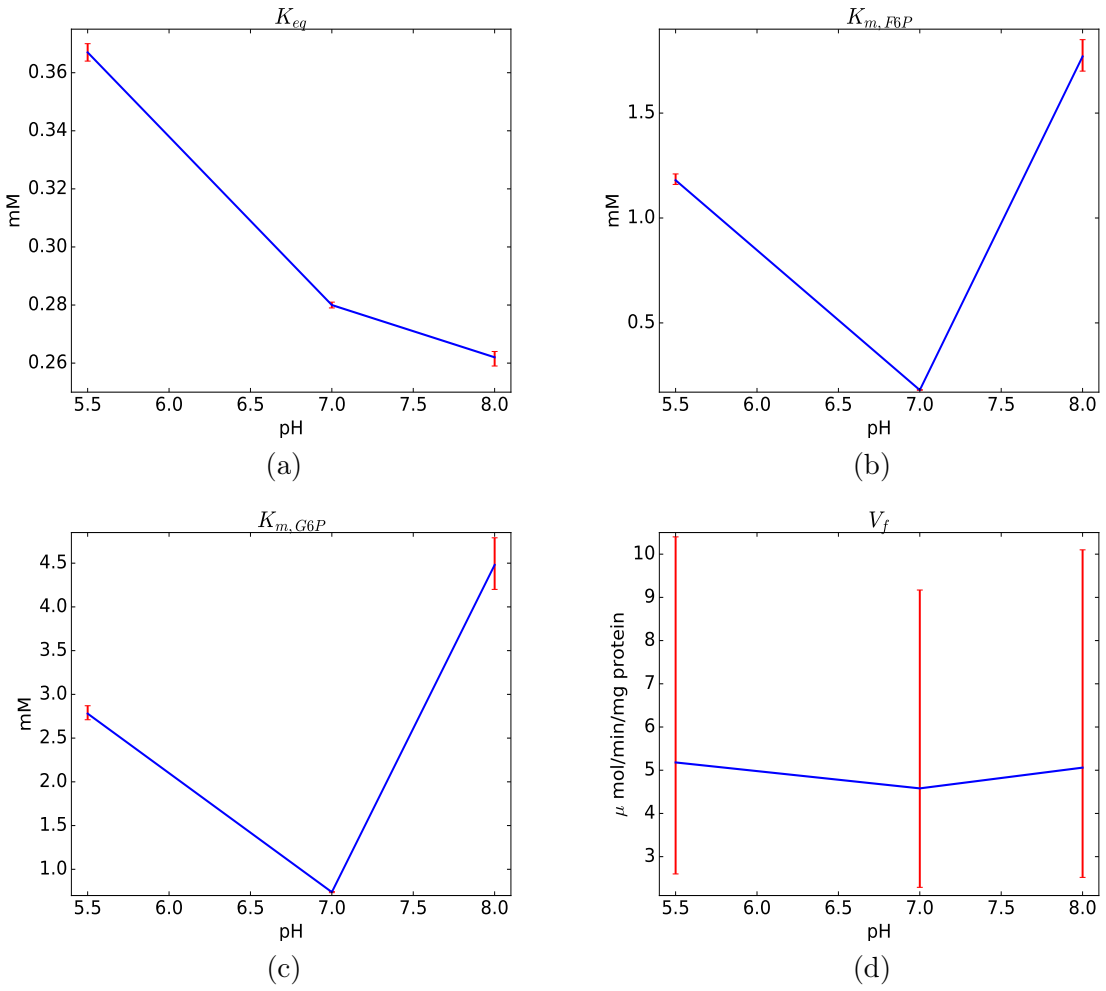


Figure 4.12: Parameters plotted against pH for the PGI catalysed reaction in *E. coli*. The red bars indicate 95% confidence intervals. The blue line illustrates the extent of change in the parameters.

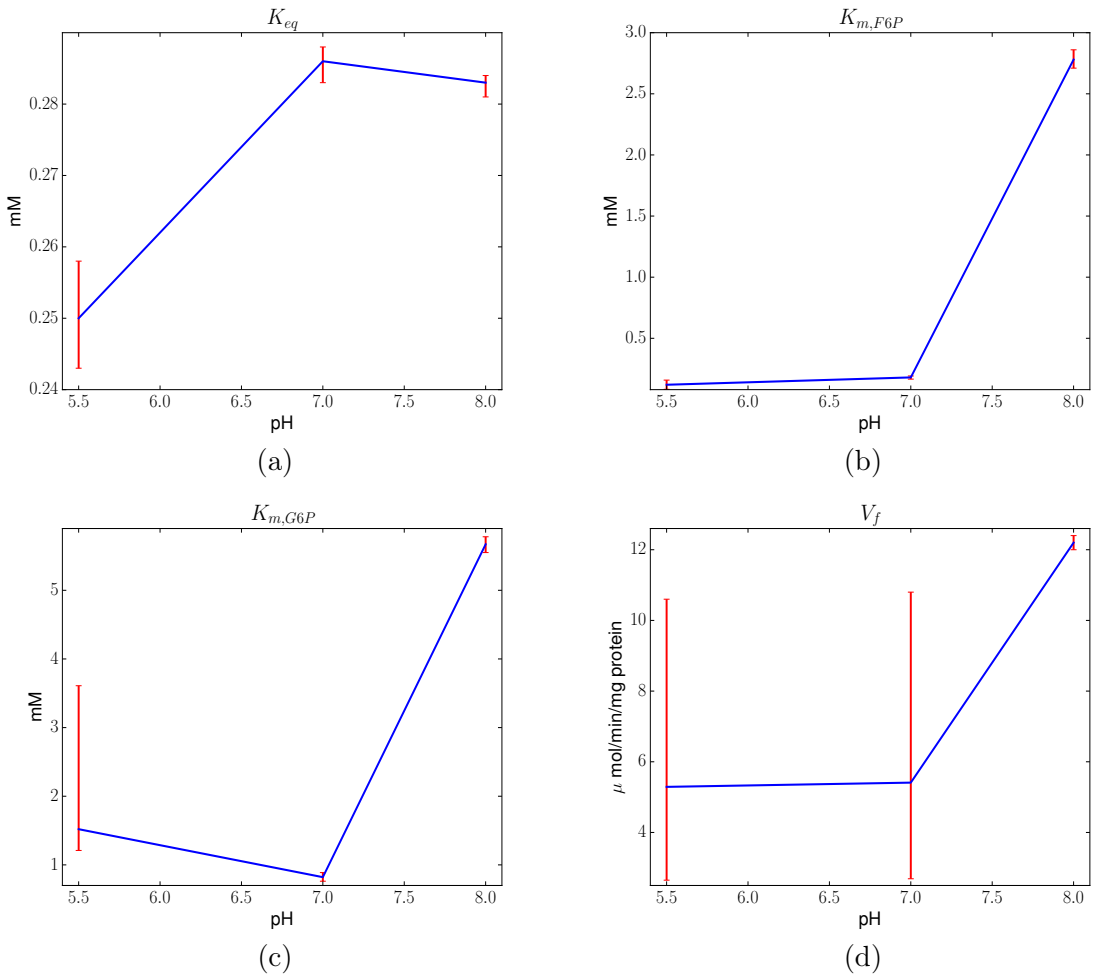


Figure 4.13: Parameters plotted against pH for the PGI catalysed reaction in *S. cerevisiae*. Plot attributes are as in Figure 4.12.

However, many changes in parameters did not follow the same trends across microorganism or enzyme.

In Figures 4.12 and 4.13, when looking at the dissociation constant for the product of PGI, F6P, no such clear trend was observed for the microorganisms. In *E. coli* the $K_{m,F6P}$ changed in the same manner as did $K_{m,G6P}$, whereas in *S. cerevisiae* the $K_{m,F6P}$ value increased as pH increased. This was mirrored in *E. coli* for the $K_{m,MgADP}$ value for PFK. In the case of PFK in *S. cerevisiae*, however, both $K_{m,MgADP}$ and $K_{m,MgATP}$ were highest at pH 7, whereas the inverse was true for $K_{m,MgATP}$ in *E. coli* (Figures 4.14 and 4.15).

As can be seen in Figures 4.12 and 4.13, the K_{eq} for the PGI catalysed reaction, although assuming different values at the varying pH values, did not change significantly relative to the other PGI parameters. Tables 4.1 to 4.3 and 4.7 to 4.9 present the parameter values for the PGI catalysed reaction, from which it can be seen that the K_{eq}, for all pH values tested, was between 0.25 and 0.37. The K_{eq} stayed relatively constant across pH. The K_{eq} for the PFK catalysed reaction, as seen in Figures 4.14 and 4.15, was a non-identifiable parameter, and conclusions about this parameter could not be made.

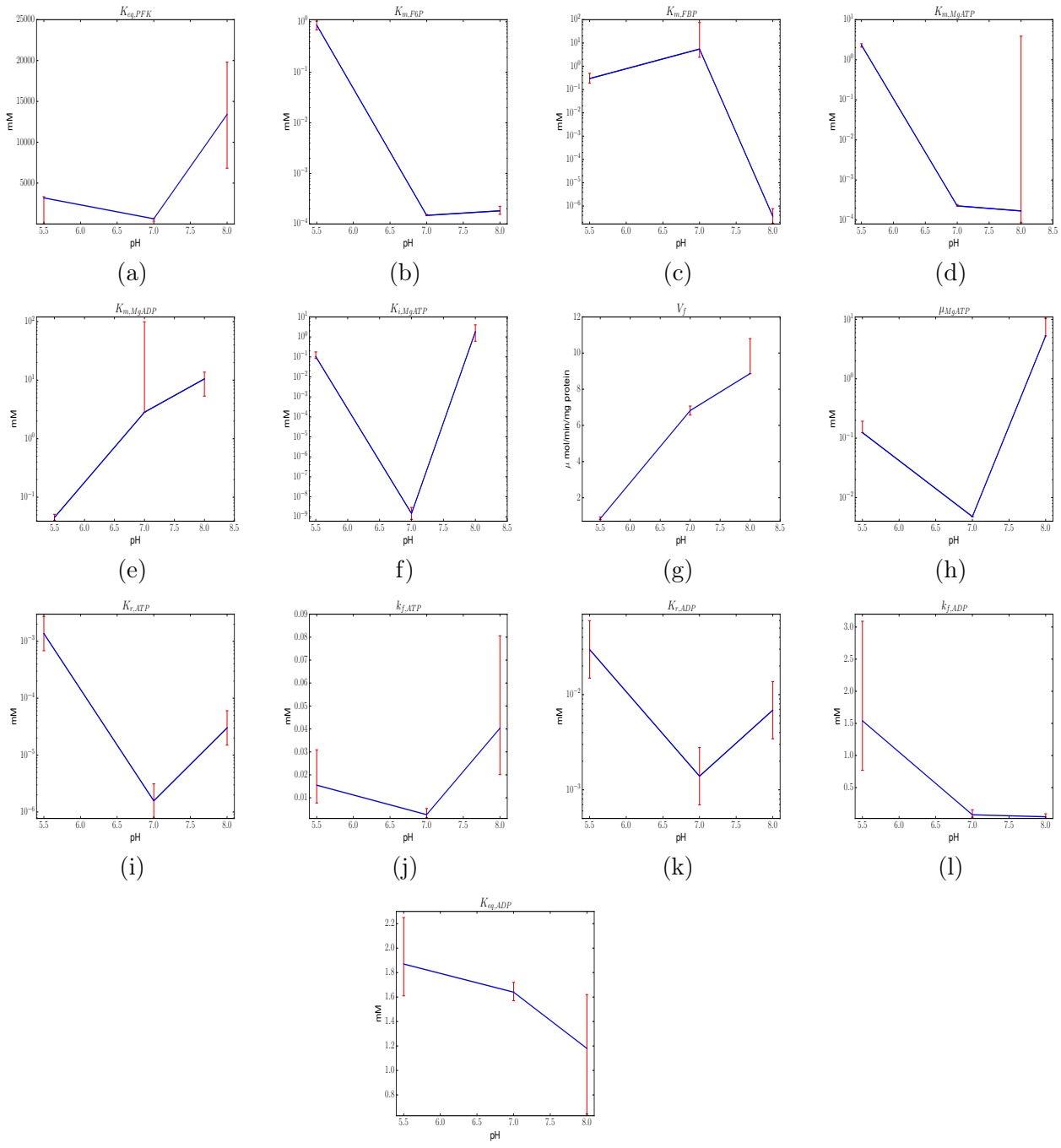


Figure 4.14: Parameters plotted against pH for the PFK catalysed reaction in *E. coli*. Plot attributes are as in Figure 4.12. Where parameter changes or error bars are large a logarithmic scale(base 10) was used on the y-axis.

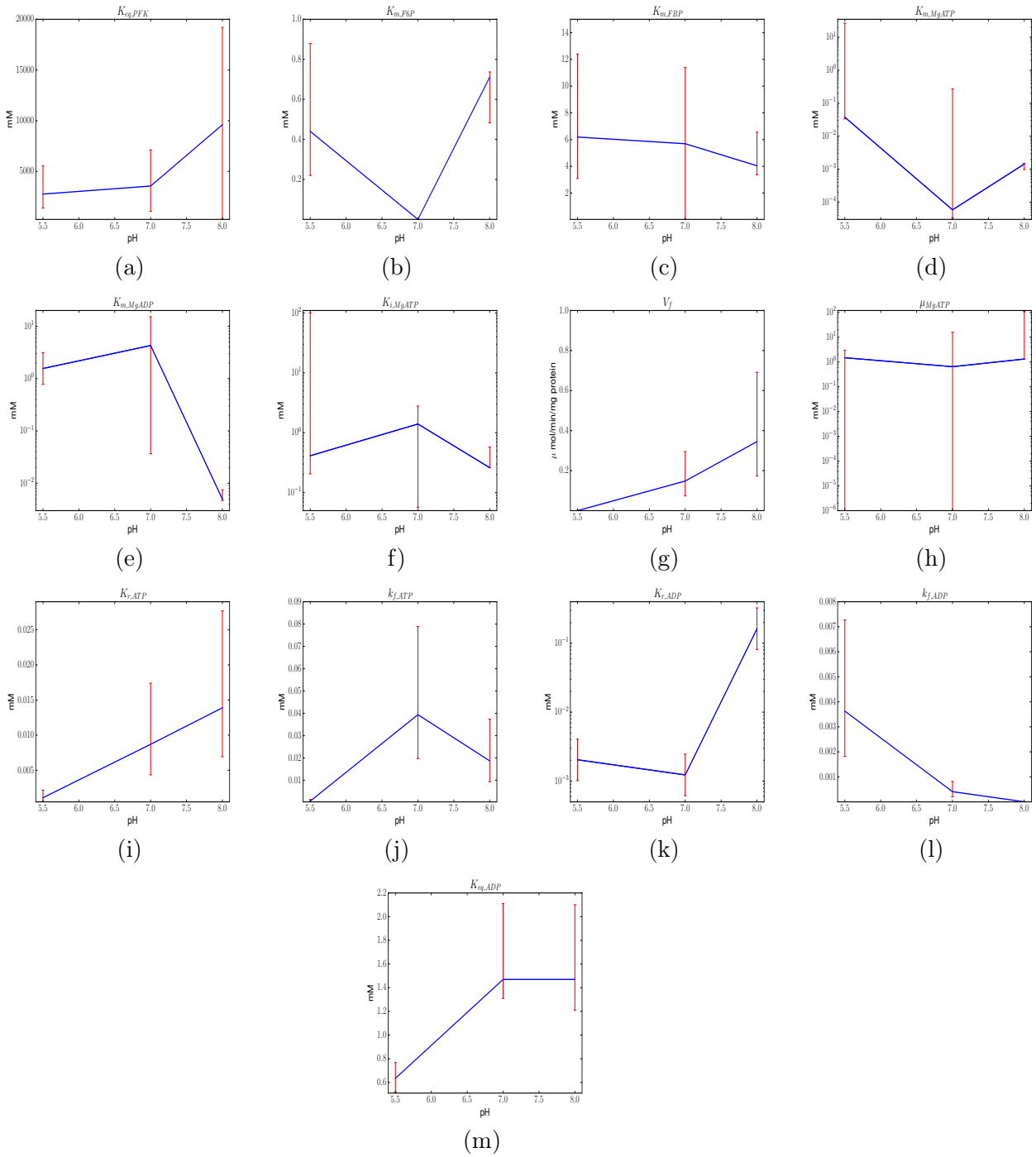


Figure 4.15: Parameters plotted against pH for the PFK catalysed reaction in *S. cerevisiae*. Plot attributes are as in Figure 4.12.

In the current study, Figures 4.14 and 4.15 show that PFK in *E. coli* followed an opposite trend to *S. cerevisiae*, in that μ_{MgATP}, a parameter indicating the extent of inhibition or activation, was higher than one at pH 8 and showed inhibition at pH 5. Evident from Figures 4.14 and 4.15, the same trend did not hold for PFK in *S.cerevisiae*, where the values for μ_{MgATP} were larger than one at both pH 5.5 and pH 8, PFK only being inhibited at pH 7 (μ larger than one indicates activation and smaller than one inhibition [40]). In *E. coli*, as can be seen by the $K_{i,MgATP}$ value in Table 4.4, there is practically no inhibition of PFK by MgATP at pH 7.

Some interesting observations could be made from the ATP, ADP and Mg^{2+} binding. The $K_{eq,ADP}$ value remained almost constant for both microorganisms at all pH values studied and lay between 1.18 and 1.64, the exception being at pH 5.5 in the *S. cerevisiae* experiments, where it was 0.648 (Tables 4.4 to 4.6 and 4.10 to 4.12). This discrepancy might be due to the fact that pH 5.5 is the optimal pH for fermentation [113] and thus ethanol formation and other enzymatic reactions , utilising ATP, might be more active in the cell free extract at this pH. This might be likely as no serine proteases or any protein inhibitors were used during extraction in this investigation, as the rate of the enzymes under investigation was lowered with the addition of protein inhibitors (data not shown). Further, in *E. coli*, all the rate constants except $k_{f,ADP}$ were at their lowest at pH 7. The value for $k_{f,ADP}$ decreased as pH increased (Figure 4.14). In the *S. cerevisiae* experiments the same trend held for $k_{f,ADP}$. The value for $k_{r,ADP}$ was at its lowest at pH 7, where $k_{f,ATP}$ was at its highest at pH 7 and $k_{f,ATP}$ increased as pH increased (Figures 4.14 and 4.15).

The observations made above will be discussed in the light of the existing literature in Chapter 5.

Chapter 5

Discussion

This chapter is a general discussion of the results presented in Chapter 4. The results will be discussed in light of the aims and objectives as stated in Section 1.3. This will be followed by some suggestions for future work in this direction of research.

This investigation forms part of a larger body of work, attempting to characterise and model the glycolytic pathway to study the central carbon metabolism of a range of microorganisms, with the aim of building on and standardising systems biology investigations. The main aim of the current research project was to determine the effects of pH change on the upper glycolytic enzymes of *E. coli* and *S. cerevisiae*. The investigation also shed some light on the topic of experimental design for systems biology and the nature of parameter estimation and identifiability of obtained parameters.

5.1 The effects of pH on kinetic parameters

The effects of pH on the kinetics of enzymes has long been a field of interest in biology and much work has been done to better understand these effects [96,97]. Furthermore, the isomerisation of glucose to fructose and the effect of pH on this interconversion has also been extensively studied [96–100]. The variation of pH and the subsequent effect on enzyme kinetics is similar to the effects of inhibitors and activators. pH effects can give insight relating to enzyme mechanisms of action as well as the presence of ionisable amino acid side chains. This in turn, could provide information on the amino acids present in the active sites of enzymes [97].

It has been shown that the transformation of glucose to fructose follows acid-base reaction dynamics and occurs as a Lobry de Bruyn-Alberdan van Eckenstein transformation via the formation of an enediol intermediate. This is a well known reaction mechanism for the conversion of ketoses to aldoses [98].

The interconversion between glucose and fructose is catalysed, in the absence of enzyme, by both acidic and alkaline aqueous solutions. The yield is relatively low in basic solution, due to the instability of monosaccharides in basic solutions [99]. In the presence of enzyme, the reaction mechanism is mixed between acid and base catalysed transformation mechanisms [98]. In the base catalysed reaction proton transfer occurs between monosaccharides and the solvent, whereas for the acid catalysed reaction proton transfer occurs intramolecularly via hydride shift. For the conversion of glucose to fructose a proton is removed from the C-2 carbon and replaced on the C-1 carbon. The same is true for the transfer of the O-2 oxygen proton, which is replaced at the O-1 oxygen position. The reverse occurs for the transformation of fructose to glucose.

PGI has been shown to use a mixture of both intramolecular and solvent transfer mechanisms of protons. The proposed mechanism is a multi-step reaction, involving catalytically active amino acids in the enzyme active site and can occur in two ways. Either by proton transfer or an intramolecular hydride shift [99, 100]. Dyson *et al.* investigated the effects of pH change on PGI to elucidate the reaction mechanism of this enzyme [96]. They identified two ionisable groups, with pK values of 6.75 and 9.3, as participating in the conversion of G6P to F6P. According to the pK values obtained the researchers identified histidine and lysine as the amino acids responsible for aldose-ketose isomerisation. They proposed the following steps for this conversion. Lysine is responsible for the binding of the substrate, where the ϵ-amino group of lysine catalyses the opening of the hexose ring through its nitrogen. It accomplishes this by protonation and subsequent binding of this proton to the oxygen atom on the hexose ring structure. The histidine residue, with a non-protonated nitrogen atom, acts as the base catalyst for this reaction [96].

It was shown that metal centres in the enzyme are responsible for the stabilisation of the open-chain form of the sugars and facilitated the isomerisation by intramolecular hydride shift [99]. Kovalesky *et al.* have shown that hydrolysed metal species in xylose isomerase also play a role in protonation and deprotonation sequences with water and glucose [101]. This led researchers to conclude that the kinetics change due to ionisable groups and not because of conformational changes brought about by pH change [96].

This is in contrast to PFK, where Trevedi *et al.* postulated that pH change may affect kinetics by antagonising conformational changes brought about by MgATP binding in frog muscle. It was also suggested that pH changes might affect the binding of MgATP to the enzyme active site [102].

Moreover, it was shown that PFK in boar spermatazoa had a decreased affinity for its substrate (F6P) at low pH and that PFK activity increased as pH did, with activator and inhibitor effects disappearing at pH values above pH 8. This led the authors to conclude that H^+ acts as an allosteric modifier for PFK. It is important to note that the activity of PFK depends on the concentration of F6P [102], thus it would make sense that PFK activity would decline as its affinity for F6P declines.

PFK is a main regulatory enzyme in glycolysis in many cell and tissue types, and is often called the rate limiting, or first committed step in glycolysis. This is due to the PFK catalysed reaction being highly exergonic, meaning that the product and substrate concentrations are maintained far removed from equilibrium. The strong inhibitory effects of ATP and H^+, at physiological concentrations, are thought to be responsible for this condition [103].

PFK displays an ordered bi-bi reaction mechanism, with MgATP and F6P as substrates, ATP as regulator and ADP and F1,6-BP as products [104–107]. F6P is the first substrate to bind, and F1,6-BP the last product to be released.

Trevedi *et al.* showed that PFK in frog muscle is highly sensitive to pH changes at physiological conditions [102] and it has been established that ATP inhibition of PFK increased as pH decreased [103, 108]. They also reported that a decrease in affinity for F6P coincided with pH decrease [102, 103]. This is corroborated in the 5th edition of Biochemistry [109]. A link between pH change and a change in the inhibition of PFK by ATP had previously been reported, where a decrease in pH was linked to a decline in inhibition of PFK by ATP in the ischemic rat brain [108].

PFK exists in two conformational states, the R and the T state, that are in equilibrium in solution. ATP can bind to both the active and the allosteric sites in both

conformational states. However, ATP preferentially binds to the T state allosteric site. This causes a shift in the equilibrium from the R state to the T state, which has a lower affinity for F6P [110]. From this it can be concluded that, as a decrease in pH causes a similar decline in affinity for F6P, that pH plays a role in conformational changes of PFK.

According to Hellenga *et al.*, the proposed amino acids in the active site of PFK are aspartic acid (127), an acidic amino acid, and arginine (171), a basic amino acid [111]. Changes in pH will effect the protonation of these amino acids, and may affect the affinity of the enzyme for its substrates, especially as the substrates have phosphate groups that are also subject to protonation and deprotonation due to pH changes.

In the light of the above, some interesting conclusions could be made from the parameter changes observed in the current investigation.

For PGI, where the interactions between amino acids and the compounds involved in the reaction are well understood, certain relationships between parameters and pH change can be understood. For example, a decreased affinity of the enzyme for G6P can be observed in Figures 4.12(c) and 4.13(c). At pH 5.5 this might be due to the protonation of the substrate, preventing Lysine from protonating the oxygen molecule, thus preventing the first step, the opening of the hexose ring structure. At pH 8, the inverse could be true, where a deprotonation of Lysine, might prevent it from protonating the hexose oxygen, subsequently preventing the first step in the reaction. This decreased affinity for G6P was seen in both microorganisms investigated. The same trend held for F6P in *E. coli*, but not for *S. cerevisiae* (Figures 4.12(b) and 4.13(b)).

In *E. coli* an increase in V_f could also be seen at both pH 5.5 and pH 8. This could be due to the fact that, as stated above, the isomerisation between G6P and F6P has been shown to be catalysed by both acidic and alkaline aqueous solutions. It might also be due to the decreased affinity for substrate, as this would lead to an initial lowering of the rate of conversion, which subsequently would lead to a accumulation of substrate, leading to an eventual rise in the forward rate of the reaction, as postulated by Trevedi *et al.* in [102].

As discussed in Chapter 4, the K_{eq} for the PGI catalysed reaction, remained relatively constant and varied within a very narrow range. This is most likely due to a balance achieved in the amount of phosphate, and subsequently protonation, on both sides of the reaction as both G6P and F6P have one phosphate group each. Thus the K_{eq} remained unaffected by pH. The small changes observed could be explained by a difference, albeit small, in the pK_a values of the compounds involved (G6P and F6P), which may result in a slight difference in their protonation states at different pH values. Observations about the K_{eq} for PFK cannot be made, as the parameter was not uniquely identifiable.

For PFK, in both microorganisms, similar trends as reported in the literature could be observed. The activity of PFK, as seen by the changes in V_f in Figures 4.14(g) and 4.15(g), increased as pH increased. This, as stated above, might be due to conformational changes in the enzyme elicited by pH changes and MgATP binding. Similarly, as reported in the literature and above, a decrease in the affinity for F6P with a decrease in pH could be seen for PFK (Figures 4.14(b) and 4.15(b)). However, a decrease in affinity for F6P was also seen with an increase in pH. It is thus concluded that affinity for its substrate, F6P, is also affected by the protonation state of the substrate rather than solely by a conformational change in the enzyme. Although a

decrease in inhibition by MgATP with an increase in pH could be seen, no increase in inhibition was seen with a decrease in pH. This is clear for *E. coli* from Figure 4.14(h) whereas, for *S. cerevisiae*, the confidence interval bounds were too large to make such a conclusion.

In *E. coli*, a decrease in the affinity of the allosteric site for MgATP can be seen at both pH 8 and pH 5.5, agreeing with reports from Trevedi *et al.* [102] that a change in pH affected the affinity of PFK for MgATP in the regulatory site. This might be due to the protonation state of the phosphate groups of MgATP, or due to a conformational change in the allosteric site, brought about by pH changes.

Additional research, including intermediate pH values, will have to be done to elucidate clearer trends of parameter versus pH.

5.2 Standard conditions for enzyme characterisations

With the experimental method outlined in this thesis all 102 parameters, making up the PGI-PFK model at all three pH values investigated, could be obtained from fitting, although not all parameters were uniquely identifiable. This allowed the analysis of the effect of pH changes on the kinetic parameters of the upper glycolytic enzymes. A discussion of the identifiability analysis follows in Section 5.3.

In Chapter 4 we pointed out some trends in parameter values with regards to pH. Evident from these trends, observed in Figures 11 to 14, pH had a definite effect on the kinetic parameters of the enzymes investigated. From this it follows that classical enzyme assays, where enzymes were tested each at their optimal pH, are not suitable for model building for systems biology, where the *in vivo* situation is the point of interest. In the *in vivo* situation, insofar as reactions take place in the same cellular compartment, conditions prevalent in many biological pathways, they take place at the same pH, often removed from the enzymes' optimal pH. As concerns systems biology investigations, this physiological situation needs to be reflected in the experimental design for the results to be useful for model building, where the model needs to be able to explain and make predictions valid for *in vivo* conditions. It is to this end that various researchers have developed *in vivo*-like assay media for systems biology investigations [12, 16]. Efforts towards the standardisation of enzyme kinetics for systems biology investigations is the main aim of STRENDA [114].

5.3 Identifiability analysis

The results of the identifiability analysis are presented in Sections 4.3 and 4.4. From this analysis it is clear that certain parameters were identifiable and some were non-identifiable.

As stated in Section 1.5.3, there are two forms of non-identifiability. Structural non-identifiability arises from the model structure, hence the name, and can only be resolved by adjusting the model. It is illustrated by a perfectly flat valley in the parameter space as can be seen in Figure 4.11(c), (d), (f), (k) and (l). Practical non-identifiability occurs when data are insufficient or of inferior quality and can be remedied by the collection of more, high quality data. Practical non-identifiability is

indicated by an infinite confidence interval region [63]. Non-identifiability can however, be a useful tool. Practical non-identifiability can help in experimental design and structurally non-identifiable parameters indicate functional relationships or correlations in data.

There were 30 parameters in total that could not be structurally identified. The non-identifiable parameters follow: the K_{eq} for PFK, a parameter indicating the relationship between products and substrates of the reaction at equilibrium, the intercorrelation between substrates and products may have caused the structural non-identifiability of this parameter; the forward as well as reverse rate constant parameters for the adenylate kinase reactions, these equations are separate from the PFK rate equation, as seen in Section 3.1.4 and are simple mass action equations. The non-identifiability of the rate constant parameters for the adenylate kinase reactions could be solved by incorporating a more detailed rate equation for these reactions into the current model. Model evaluation however, fell outside of the scope of this investigation.

There were 2 parameters that were practically non-identifiable: μ_{MgATP} for PFK in *S. cerevisiae* at pH 5.5 and pH 7. Practical non-identifiability can be resolved by obtaining more high-quality data.

The exclusion of the separate adenylate kinase reactions was attempted, but led to non-identifiabilities in certain kinetic parameters of the enzymes of the PGI-PFK model, the main focus of this investigation. These reactions are relevant however, as the adenylate kinase reactions are active in a cell-free extract and will directly influence the ATP and ADP metabolite pools and their respective concentrations. This will effect the amount of free ATP and ADP available in solution at all times [22].

The red vertical bars in Figures 4.12 to 4.15 indicate the 95% confidence intervals (CIs). The CI upper and lower bounds demarcate the barriers of the 95% confidence region. The size of the confidence region indicates how accurately a parameter can be determined and, as can be seen in Figure 4.11, the non-identifiable parameters tend to have wide confidence regions. For some parameters, the lower CI bound had a negative value. This is due to the fact that the calculation of the confidence intervals is a purely numerical approach and does not take biological situations into account, where parameter values cannot assume negative values. These values are indicated in the Figures 4.12 to 4.15 as zero, the true values obtained from the identifiability analysis are presented in Tables 4.10 and 4.11. As these parameters are structurally identifiable this is a matter of practical non-identifiability and can be resolved by the addition of high-quality data to the existing data sets.

5.4 Future work

As stated in Section 5.3 there are 5 structurally non-identifiable parameters in the PGI-PFK model and 32 non-identifiable parameters overall. A model comparison with other possible models can be done, followed by an iterative approach of evaluating and re-evaluating various possible models until structural non-identifiabilties can be resolved. A benefit of this approach is that structural non-identifiability can be determined *a priori*, that is, before experimentation [63].

The first step to furthering the work of this thesis would be a re-evaluation of the PGI-PFK model to resolve structural non-identifiabilities. Furthermore, conditionally

expressed enzymes can be included in future studies; this will be especially useful for cases like PFK in *S. cerevisiae* where there are two isozymes present in a cell free extract, PFK-1 and PFK-2 [115]. This will ensure that one is investigating the enzyme of interest and that isozymes are not also playing a role in the observed results.

Another consideration is the isotope effect that might be caused by the added D_2O. The isotope effect can be briefly described as a change in the rate of an enzymatic reaction due to a change in the mass of atoms involved in the reaction. A concentration of 10 % D_2O would lead to 10 % of the hydrogen ions in solution being deuterium ions and may affect the catalytic activity of enzymes where proton transfer or exchange occurs [116,117]. To address this, NMR experiments could be performed using capillary NMR tubes, with the reference D_2O inside the capillary, thereby avoiding the isotope effect.

To gain further insight into the relationship between pH and the kinetic parameters of the upper glycolytic enzymes the intermediate pH values, between those investigated here would have to be done, as well as higher and lower pH values than pH 8 and pH 5.5. In this manner plots can be generated that depict the relationship between pH and the parameter and trends can be identified. Where clear relationships are evident, these trends might yield useful information regarding the underlying principles governing the pH-dependence of the parameter. Moreover, mathematical functions describing the pH-dependence can be fitted by interpolation.

Furthermore, a detailed study on all the enzymes of glycolysis, in both these microorganisms at varying pH, may facilitate the construction of pH sensitive computational models. This can be accomplished by applying the methodology presented here on each of the other enzymes in the glycolytic pathway at varying pH values. This will help with the construction of a model of the full glycolytic pathway at the pH values discussed here, as well as intermediate values, assisting the elucidation of mathematical trends, which could then be exploited to build a pH sensitive model for glycolysis in the microorganisms studied in this investigation. The model could also be further augmented by including pH (as H^+ concentration) as a variable in the equations used for modelling.

With such a model pathway flux in response to pH change may be studied, as well as other effects pH changes might elicit at the pathway level that might not be evident at enzyme level. Further, questions can be answered concerning the glucose signal, a sudden decrease in intracellular glucose concentration leading to a decrease in pH.

Section 2.1.1 mentions a link between nutrient availability, pH and growth. With a pH sensitive model of the appropriate microorganisms, these links can be better investigated and understood. A pH sensitive model would also facilitate the formation of novel hypotheses and the design of new experiments,such as finding the optimal pH for the growth of these microorganisms and possibly reducing their growing time, subsequently saving money and labour time in industry.

Bibliography

[1] Rietman, E.; Karp, R. and Tuszynski, J. (2011). Review and application of group theory to molecular systems biology, Theor. Biol. Med. Model. 8 : 21.

[2] Ma'ayan, A. (2008). Network integration and graph analysis in mammalian molecular systems biology, Systems Biology, IET 2 : 206-221.

[3] Latterich, M. (2005). Molecular systems biology at a crossroads: to know less about more, or to know more about less?, Proteome Science 3 : 8.

[4] Thompson, J. and Poch, O. (2006). Multiple Sequence Alignment as a Workbench for Molecular Systems Biology, Current Bioinformatics 1 : 95-104.

[5] Swainston, N.; Golebiewski, M.; Messiha, H. L.; Malys, N.; Kania, R.; Kengne, S.; Krebs, O.; Mir, S.; Sauer-Danzwith, H.; Smallbone, K.; Weidemann, A.; Wittig, U.; Kell, D. B.; Mendes, P.; Muller, W.; Paton, N. W. and Rojas, I. (2010). Enzyme kinetics informatics: from instrument to browser, FEBS Journal 277 : 3769-3779.

[6] Tummler, K.; Lubitz, T.; Schelker, M. and Klipp, E. (2014). New types of experimental data shape the use of enzyme kinetics for dynamic network modeling, FEBS Journal 281 : 549-571.

[7] Yu, C.; Boutté, A.; Yu, X.; Dutta, B.; Feala, J. D.; Schmid, K.; Dave, J.; Tawa, G. J.; Wallqvist, A. and Reifman, J. (2015). A systems biology strategy to identify molecular mechanisms of action and protein indicators of traumatic brain injury, Journal of Neuroscience Research 93 : 199-214.

[8] Sewald, N. and Puhle, A. (2007). Editorial: Molecular systems biology, J. Biotechnol. 129 : 171-172.

[9] Durmus, S.; Cakir, T.; Ozgur, A. and Guthke, R. (2015). A Review on Computational Systems Biology of Pathogen-Host Interactions, Frontiers in Microbiology 6 : 235.

[10] Murabito, E.; Colombo, R.; Wu, C.; Verma, M.; Rehman, S.; Snoep, J.; Peng, S.-L.; Guan, N.; Liao, X. and Westerhoff, H. (2015). SupraBiology 2014: Promoting UK-China collaboration on Systems Biology and High Performance Computing, Quantitative Biology 3 : 46-53.

[11] Shih, A. J.; Purvis, J. and Radhakrishnan, R. (2008). Molecular systems biology of ErbB1 signaling: bridging the gap through multiscale modeling and high-performance computing, Mol Biosyst. 4 : 1151-1159.

[12] van Eunen, K.; Bouwman, J.; Daran-Lapujade, P.; Postmus, J.; Canelas, A. B.; Mensonides, F. I. C.; Orij, R.; Tuzun, I.; van den Brink, J.; Smits, G. J.; van Gulik, W. M.; Brul, S.; Heijnen, J. J.; de Winde, J. H.; Teixeira de Mattos, M. J.; Kettner, C.; Nielsen, J.; Westerhoff, H. V. and Bakker, B. M. (2010). Measuring enzyme activities under standardized in vivo-like conditions for systems biology, FEBS Journal 277 : 749-760.

[13] Adamczyk, M.; van Eunen, K.; Bakker, B. M. and Westerhoff, H. V. (2011). Chapter thirteen - Enzyme Kinetics for Systems Biology: When, Why and How . In: Daniel Jameson, M. V. and Westerhoff, H. V. (Ed.), Methods in Systems Biology, FEBS Lett. 587(17) : 2832-2841.

[14] Mendes, P.; Messiha, H.; Malys, N. and Hoops, S. (2009). Chapter 22 Enzyme Kinetics and Computational Modeling for Systems Biology . In: Brand, M. L. J. L. (Ed.), Methods in Enzymology, Methods Enzymol. 467 : 583-99.

[15] Scopes, R. K. (2001). In: (Ed.), Enzyme Activity and Assays, John Wiley and Sons, Ltd.

[16] García-Contreras, R.; Vos, P.; Westerhoff, H. V. and Boogerd, F. C. (2012). Why in vivo may not equal in vitro? New effectors revealed by measurement of enzymatic activities under the same in vivo-like assay conditions, FEBS Journal 279 : 4145-4159.

[17] Orij, R.; Urbanus, M. L.; Vizeacoumar, F. J.; Giaever, G.; Boone, C.; Nislow, C.; Brul, S. and Smits, G. J. (2012). Genome-wide analysis of intracellular pH reveals quantitative control of cell division rate by pH c in Saccharomyces cerevisiae, Genome Biology 13(9) : 80.

[18] Rubenstein, E. M. and Schmidt, M. C. (2010). The glucose signal and metabolic p[H +]lux, The EMBO Journal 29 : 2473-2474.

[19] Dechant, R.; Binda, M.; Lee, S. S.; Pelet, S.; Winderickx, J. and Peter, M. (2010). Cytosolic pH is a second messenger for glucose and regulates the PKA pathway through V-ATPase, The EMBO Journal 29 : 2515-2526.

[20] Orij, R.; Postmus, J.; Ter-Beek, A.; Brul, S. and Smits, G. J. (2009). In vivo measurement of cytosolic and mitochondrial pH using a pH-sensitive GFP derivative in Saccharomyces cerevisiae reveals a relation between intracellular pH and growth, Microbiology 155 : 268-278.

[21] Young, B. P.; Shin, J. J. H.; Orij, R.; Chao, J. T.; Li, S. C.; Guan, X. L.; Khong, A.; Jan, E.; Wenk, M. R.; William; Prinz; Smits, G. J. and Loewen, C. J. R. (2010). Phosphatidic Acid Is a pH Biosensor That Links Membrane Biogenesis to Metabolism, Science 329 : 1085-1088.

[22] Eicher, J. J. (2013). Understanding Glycolysis in Escherichia coli : a Systems Approach using Nuclear Magnetic Resonance Spectroscopy, Stellenbosch University. PhD Thesis.

[23] Eicher, J. J.; Snoep, J. L. and Rohwer, J. M. (2012). Determining Enzyme Kinetics for Systems Biology with Nuclear Magnetic Resonance Spectroscopy, Metabolites 2 : 818-843.

[24] Small, P.; Blankenhorn, D.; Welty, D.; Zinser, E. and Slonczewski, J. L. (1994). Acid and base resistance in Escherichia coli and Shigella flexneri: role of rpoS and growth pH., Journal of Bacteriology 176 : 1729-1737.

[25] Diez-Gonzalez, F. and Russell, J. B. (1997). The ability of Escherichia coli O157:H7 to decrease its intracellular pH and resist the toxicity of acetic acid, Microbiology 143 : 1175-1180.

[26] Moreau P. L. (2007). The lysine decarboxylase CadA protects Escherichia coli starved of phosphate against fermentation acids, J Bacteriol 189(6) : 2249-2261.

[27] Schuster, P. (2008). Modeling in biological chemistry. From biochemical kinetics to systems biology, Monatshefte für Chemie - Chemical Monthly 139 : 427-446.

[28] Härdin, H. M.; Zagaris, A.; Krab, K. and Westerhoff, H. V. (2009). Simplified yet highly accurate enzyme kinetics for cases of low substrate concentrations, FEBS Journal 276 : 5491-5506.

[29] Westerhoff, H.; Kolodkin, A.; Conradie, R.; Wilkinson, S.; Bruggeman, F.; Krab, K.; van Schuppen, J.; Hardin, H.; Bakker, B.; Moné, M.; Rybakova, K. N.; Eijken, M.; van Leeuwen, H. and Snoep, J. (2009). Systems biology towards life in silico: mathematics of the control of living cells, Journal of Mathematical Biology 58 : 7-34.

[30] Karr, J.; Sanghvi, J.; Macklin, D.; Gutschow, M.; Jacobs, J.; Jr., B. B.; Assad-Garcia, N.; Glass, J. and Covert, M. (2012). A Whole-Cell Computational Model Predicts Phenotype from Genotype , Cell 150 : 389-401.

[31] Laubenbacher, R. and Jarrah, A. S. (2009). Chapter 7 - Algebraic Models of Biochemical Networks . In: Brand, M. L. J. L. (Ed.), Methods in Enzymology, Academic Press 467 : 163-196.

[32] Shahzad, K. and Loor, J. J. Application of Top-Down and Bottom-up Systems Approaches in Ruminant Physiology and Metabolism Current Genomics, 2010, 13, 379-394.

[33] Oyarzún, D. (2011). Optimal control of metabolic networks with saturable enzyme kinetics, IET Systems Biology 5(9) : 110-119.

[34] Michaelis, L. and Menten, M. (1913). Die Kinetik der Invertinwirkung, Biochem. Z. 49 : 333.

[35] Michaelis, L.; Menten, M. L.; Johnson, K. A. and Goody, R. S. (2011). The original Michaelis constant: translation of the 1913 Michaelis-Menten paper., Biochemistry (Mosc.) 50 : 8264.

[36] Goddard, J. P. and Reymond, J. L. (2004). Enzyme assays for high-throughput screening , Current Opinion in Biotechnology 15 : 314-322.

[37] Hofmeyr, J. and Cornish-Bowden, A. (1997). The reversible Hill equation: how to incorporate cooperative enzymes into metabolic models, Bioinformatics/computer Applications in The Biosciences 13 : 377-385.

[38] Westermark, P. O.; Hellgren-Kotaleski, J. and Lansner, A. (2004). Derivation of a reversible Hill equation with modifiers affecting catalytic properties, WSEAS Transactions on Biology and Medicine 1 : 91-98.

[39] Rohwer, J. M.; Hanekom, A. J. and Hofmeyr, J.-H. S. (2006). A Universal Rate Equation for Systems Biology, Beilstein-Institut, ESCEC .

[40] Hanekom, A. J. (2006). Generic kinetic equations for modelling multisubstrate reactions in computational systems biology. Masters thesis.

[41] Goddard, J. P. and Reymond, J. L. (2004). Recent advances in enzyme assays, TRENDS in Biotechnology 22 : 363-370.

[42] Behzadi, A.; Hatleskog, R. and Ruoff, P. (1999). Hysteretic enzyme adaptation to environmental pH: change in storage pH of alkaline phosphatase leads to a pH-optimum in the opposite direction to the applied change, Biophysical Chemistry 77 : 99-109.

[43] Yang, K. Y.; Haynes, C. A.; Spatzal, T.; Rees, D. C. and Howard, J. B. (2014). Turnover-Dependent Inactivation of the Nitrogenase MoFe-Protein at High pH, Biochemistry 53 : 333-343.

[44] Orij, R.; Brul, S. and Smits, G. J. (2011). Intracellular pH is a tightly controlled signal in yeast , Biochimica et Biophysica Acta (BBA) - General Subjects 1810 : 933-944.

[45] Makhlynets, O. V.; Raymond, E. A. and Korendovych, I. V. (2015). Design of Allosterically Regulated Protein Catalysts, Biochemistry 54 : 1444-1456.

[46] Prasad, M.; Thomas, J. L.; Whittal, R. M. and Bose, H. S. (2012). Mitochondrial 3 -Hydroxysteroid Dehydrogenase Enzyme Activity Requires Reversible pH-dependent Conformational Change at the Intermembrane Space, J. Biol. Chem. 287 : 9534-9546.

[47] de Oliveira, L. C.; da Silva, V. M.; Colussi, F.; Cabral, A. D.; de Oliveira Neto, M.; Squina, F. M. and Garcia, W. (2015). Conformational Changes in a Hyperthermostable Glycoside Hydrolase: Enzymatic Activity Is a Consequence of the Loop Dynamics and Protonation Balance, PLoS ONE 10 : e0118225.

[48] Salmond, C. V.; Kroll, R. G. and Booth, I. R. (1984). The Effect of Food Preservatives on pH Homeostasis in Escherichia coli, Microbiology 130 : 2845-2850.

[49] Petrovska, I.; Nüske, E.; Munder, M. C.; Kulasegaran, G.; Malinovska, L.; Kroschwald, S.; Richter, D.; Fahmy, K.; Gibson, K.; Verbavatz, J. M. and Alberti, S. (2014). Filament formation by metabolic enzymes is a specific adaptation to an advanced state of cellular starvation, eLife 3.

[50] Weber, H. and Brecker, L. (2000). Online NMR for monitoring biocatalysed reactions , Current Opinion in Biotechnology 11 : 572-578.

[51] Jordà, J.; de Jesus, S. S.; Peltier, S.; Ferrer, P. and Albiol, J. (2014). Metabolic flux analysis of recombinant Pichia pastoris growing on different glycerol/methanol mixtures by iterative fitting of NMR-derived 13C-labelling data from proteinogenic amino acids , New Biotechnology 31 : 120-132.

[52] Pelletier, M.; Billingham, L. K.; Ramaswamy, M. and Siegel, R. M. (2014). Chapter Seven - Extracellular Flux Analysis to Monitor Glycolytic Rates and Mitochondrial Oxygen Consumption . In: Galluzzi, L. and Kroemer, G. (Ed.), Conceptual Background and Bioenergetic/Mitochondrial Aspects of Oncometabolism, Methods Enzymol. 542 : 125-49.

[53] Fliniaux, O.; Mesnard, F.; Raynaud, S.; Baltora, S.; Robins, R. J. and Fliniaux, M.A. (2001). Use of heteronuclear multiple bond coherence NMR spectroscopy to monitor nitrogen metabolism in a transformed root culture of Datura stramonium , Comptes Rendus de l'Académie des Sciences - Series IIC - Chemistry 4 : 775-778.

[54] Cass, A. E.; Ribbons, D. W.; Rossiter, J. T. and Williams, S. R. (1987). Biotransformation of aromatic compounds Monitoring fluorinated analogues by NMR , FEBS Letters 220 : 353-357.

[55] Aguayo, J. B.; McLennan, I. J.; Jr, C. G. and Cheng, H.-M. (1988). Dynamic monitoring of corneal carbohydrate metabolism using high-resolution deuterium NMR spectroscopy , Experimental Eye Research 47 : 337-343.

[56] Henry, O.; Kamen, A. and Perrier, M. (2007). Monitoring the physiological state of mammalian cell perfusion processes by on-line estimation of intracellular fluxes , Journal of Process Control 17 : 241-251.

[57] Beauvieux, M. C.; Stephant, A.; Gin, H.; Serhan, N.; Couzigou, P. and Gallis, J.L. (2013). Resveratrol mainly stimulates the glycolytic ATP synthesis flux and not the mitochondrial one: A saturation transfer NMR study in perfused and isolated rat liver , Pharmacological Research 78 : 11-17.

[58] Fan, T. W. M.; Higashi, R. M.; Lane, A. N. and Jardetzky, O. (1986). Combined use of 1H-NMR and GC-MS for metabolite monitoring and in vivo 1H-NMR assignments , Biochimica et Biophysica Acta (BBA) - General Subjects 882 : 154-167.

[59] Zheng, C.; Zhang, S.; Ragg, S.; Raftery, D. and Vitek, O. (2011). Identification and quantification of metabolites in 1H NMR spectra by Bayesian model selection, Bioinformatics 27 : 1637-1644.

[60] Murphy, E. F.; Gilmour, S. G. and Crabbe, M. C. (2002). Effective experimental design: enzyme kinetics in the bioinformatics era , Drug Discovery Today 7 : 187-191. Journal of Bacteriology 189 : 2249-2261.

[61] Rohwer, J. M. (2012). Kinetic modelling of plant metabolic pathways. Journal of Experimental Botany, 63(6) : 2275-2292.

[62] Rohwer J. M. and Botha F. C. (2001). Analysis of sucrose accumulation in the sugar cane culm on the basis of in vitro kinetic data, Biochem J. 358 : 437-45.

[63] Raue A.; Kreutz C.; Maiwald, T.; Bachmann J.; Schilling, M.; Klingmüller, U. and Timmer, J. (2009). Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood, Bioinformatics, 25(15) 2009 : 1923-1929.

[64] Raue, A.; Becker, V.; Klingmüller, U. and Timmer, J. (2010). Identifiability and observability analysis for experimental design in nonlinear dynamical models, Chaos 20 : 045105.

[65] Raue, A.; Karlsson, J.; Saccomani, M. P.; Jirstrand, M. and Timmer, T. (2014). Comparison of approaches for parameter identifiability analysis of biological systems, Bioinformatics 30(10) : 1440-1448.

[66] Berthoumieux, S.; Brilli, M.; Kahn, D.; De Jong, H. and Cinquemani, E. (2012). On the identifiability of metabolic network models, Journal of Mathematical Biology 67(6-7) : 1795-1832.

[67] Kreutz, C. and Timmer J. (2008). Systems biology: experimental design, FEBS Journal 276 : 923-942.

[68] Hines, K. E.; Middendorf, T. R. and Aldrich, R. W. (2014). Determination of parameter identifiability in nonlinear biophysical models: A Bayesian approach, Gen. Physiol. 143(3) : 401-416.

[69] Little, M. P.; Heidenreich, W. F. and Li, G. (2010).Parameter identifiability and redundancy: Theoretical considerations, PLoS ONE 5(1) : e8915.

[70] Cedersund, G. and Roll, J.. (2008). Systems biology: model based evaluation and comparison of potential explanations for given biological data, FEBS Journal 276 : 903-922.

[71] Villaverde, A. F.; Barreiro, A. and Papachristodoulou, A. (2016). Structural identifiability of dynamic systems biology models, PLoS Comput Biol 12(10) : e1005153.

[72] Ashyraliyev, M.; Fomekong-Nanfack, Y.; Kaandorp, J. A and Blom, J. G. (2008). Systems biology: parameter estimation for biochemical models, FEBS Journal 276 : 886-902.

[73] Hooke, R. and Jeeves, T.A. (1961). Direct search solution of numerical and statistical problems, J Assoc Comput Mach 8 : 212-229.

[74] Nelder, J.A. and Mead, R. (1965). A simplex method for function minimization, Comput J 7 : 308-313.

[75] Lagarias, J. C; Reeds, J. A; Wright, M. H. and Wright, P. E. (1998). Convergence properties of the Nelder-Mead simplex method in low dimensions, SIAM J Optim 9 : 112-147.

[76] Bellu, G.; Saccomani M. P.; Audoly, S. and D'Angiò, L. (2007). DAISY: a new software tool to test global identifiability of biological and physiological systems, Comput. Methods Programs Biomed. 88 : 52-61.

[77] Sedoglavic, A. (2002). A probabilistic algorithm to test local algebraic observability in polynomial time, J. Symbolic Comput. 33 : 735-755.

[78] Anguelova, M.; Karlsson, J. and Jirstrand, M. (2012). Minimal output sets for identifability, Math. Biosci. 239 : 139-153.

[79] Kreutz, C.; Raue, A. and Timmer, J. (2012). Likelihood based observability analysis and confidence intervals for predictions of dynamic models, BMC Syst. Biol. 6 : 120.

[80] Portugal-Nunes, D. J.; Pawar, S. S.; Lidén, G. and Gorwa-Grauslund, M. F. (2017). Effect of nitrogen availability on the poly-3-d-hydroxybutyrate accumulation by engineered Saccharomyces cerevisiae, AMB Express. 7 : 35.

[81] Bradford, M. (1976). A rapid and sensitive method for the quantitation of microgram quantities of protein utilizing the principle of protein-dye binding, Anal.Biochem. 72(1) : 248.

[82] Olivier, B. G.; Rohwer, J. M. and Hofmeyr, J.-H. S. (2005). Modelling cellular systems with PySCeS, Bioinformatics 21 : 560-561.

[83] Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; Ivanov, P.; Avila, D.; Abdalla, S.; Willing, C. and the Jupyter Development Team. (2016). Jupyter Notebooks-a publishing format for reproducible computational workflows, Positioning and Power in Academic Publishing: Players, Agents and Agendas 87-90.

[84] Gao, H., Chen, Y.; Leary, J.A. (2005). Kinetic measurements of phosphoglucose isomerase and phosphomannose isomerase by direct analysis of phosphorylated aldose-ketose isomers using tandem mass spectrometry, Int. J. Mass Spectrom. 240 : 291-299.

[85] Desvergnes, S.; Courtiol-Legourd, S.; Daher, R.; Dabrowski, M.; Salmon, L. and Therisod, M. (2012). Synthesis and evaluation of malonate-based inhibitors of phosphosugar-metabolizing enzymes: class II fructose-1,6-bis-phosphate aldolases, type I phosphomannose isomerase, and phosphoglucose isomerase, Bioorg. Med. Chem. 20 : 1511-1520.

[86] Schreyer, R. and Böck, A. (1980). Phosphoglucose isomerase from Escherichia coli K10: purification, properties and formation under aerobic and anaerobic conditions, Arch. Microbiol. 127 : 289-298.

[87] Wang, X. and Kemp, R. G. (2001). Reaction path of phosphofructo-1-kinase is altered by mutagenesis and alternative substrates. Biochemistry 40: 3938-3942.

[88] Wang, X. and Kemp, R.G. (1999). Identification of residues of Escherichia coli phosphofructokinase that contribute to nucleotide binding and specificity, Biochemistry 38 : 4313-4318.

[89] Rivas-Pardo, J.; Caniuguir, A.; Wilson, C.; Babul, J. and Guixe, V.,Divalent metal cation requirements of phosphofructokinase-2 from E. coli. Evidence for a high affinity binding site for Mn2+. (2011). Arch. Biochem. Biophys. 505 : 60-66.

[90] Zheng, R .L. and Kemp, R .G. (1992). The mechanism of ATP inhibition of wild type and mutant phosphofructo-1-kinase from Escherichia coli, J. Biol. Chem. 267 : 23640-23645.

[91] Zheng, R .L. and Kemp, R .G. (1995). Phosphofructo-1-kinase: role of charge neutralization in the active site, Biochem. Biophys. Res. Commun. 214 : 765-770.

[92] Teusink, B.; Passarge, J.; Reijenga, C. A.; Esgalhado, E.; van der Weijden, C. C.; Schepper, M.; Walsh, M.C.; Bakker, B. M.; van Dam, K.; Westerhoff, H. V. and Snoep, J. L. (2000). Can yeast glycolysis be understood in terms of in vitro kinetics of the constituent enzymes?, Testing biochemistry. Eur J Biochem. 267(17) : 5313-5329.

[93] Noltmann, E.A. (1972). Aldose-ketose isomerases, The Enzymes, 3rd Ed. (Boyer, P. D. , ed.) 6 : 271-354.

[94] Marchand, M.; Kooystra, U.; Wierenga, R. K.; Lambeir, A. M.; van Beeumen, J.; Opperdoes, F. R. and Michels, P. A. M. (1989). Glucosephosphate isomerase from Trypanosoma brucei. Cloning and characterization of the gene and analysis of the enzyme, Eur. J. Biochem. 184 : 455-464.

[95] Wurster, B. and Schneider, F. (1970). Kinetik der Glucosephosphate-Isomerase (EC 5.3.1.9) aus Hefe und ihre Anwendung auf Flußberechnungen durch die Ga ¨rungkette anaeroben Hefezelle, Hoppe-Seyler Z. Physiol. Chem. 351 : 961-966.

[96] Dyson, J. E. D. and Noltmann, E. A. (1967). The effect of pH and temperature on the kinetic parameters of phosphoglucose isomerase. Participation of Histidine and Lysine in a proposed duel function mechanism, The Journal of Biological Chemistry 243 : 1401-1414.

[97] Tipton, K. F. and Dixon, H. B. F. (1979). Effects of pH on Enzymes, Methods Enzymol. 63 : 183-234.

[98] Topper, Y. J. and Stetten, D. (1950). The alkali catalyzed isomerization of glucose into fructose and mannose, The Journal of Biological Chemistry 189 : 191-202.

[99] Roman-Leshkov, Y.; Moliner, M.; Labinger, J. A and Davis, M. E. (2010). Mechanism of Glucose Isomerization Using a Solid Lewis Acid Catalyst in Water, Angewandte Chemie International edition 49(47) : 8954-8957.

[100] Gaily, B. H; Elhassan, B. M; Abasaeed, A. E and Al-Shrhan, M. (2010). Isomerization and Kinetics of Glucose into Fructose, IJET-IJENS 10(3) : 1-5.

[101] Kovalevsky, A. Y.; Hanson, L.; Fisher, V; Mustyakimov, M.; Mason,V; Forsyth, V. T.; Blakeley, M. P.; Keen, D. A.; Wagner, T.; Carrell, H. L.; Katz, A. K.; Glusker, J. P. and Langan, P. (2010). Metal Ion Roles and the Movement of Hydrogen during Reaction Catalyzed by D-Xylose Isomerase: A Joint X-Ray and Neutron Diffraction Study, Structure 18 : 688.

[102] Trivedi, B. and Danforth, W. H. (1966). Effect of pH on the Kinetics of Frog Muscle Phosphofructokinase, The Journal Of Biological Chemistry 241(17) : 4110-4114.

[103] Kamp, G.; Schmidt, H.; Stypa, H.; Feiden, S.; Mahling, C. and Wegener, G. (2007). Regulatory properties of 6-phosphofructokinase and control of glycolysis in boar spermatozoa, Reproduction 133 : 29-40.

[104] Campos, G.; Guixe, V. Q. and Babul, J. (1983). Kinetic Mechanism of Phosphofructokinase-2 from Escherichia coli: a mutant enzyme with a different mechanism, The Journal Of Biological Chemistry 259(10) : 6147-6152.

[105] Surendranathan, K. K.; Iyer, M. G. and Nair, P. M. Mechanism of action of a dimeric phosphofructokinase from banana: role of magnesium on its kinetics and regulation. (1992). Plant science 81(1) : 29-36.

[106] Kuby, S. A. A Study of Enzymes 2: 446.

[107] Etiemble, J.; Picat, C. and Boivin, P. (1977). Reaction mechanism of erythrocyte phosphofructokinase, Biochimie 59(8-9) : 673-678.

[108] Lajtha, A. Alterations of chemical equilibrium in the nervous system. (2013). pp 283.

[109] Berg, J. M; Tymoczko, J. L and Stryer, L. (2002). Biochemistry, 5th edition, Section 16.2.

[110] Schirmer, T. and Evans, P. R. (1990). Structural basis of the allosteric behaviour of phosphofructokinase, Nature, 343(6254) : 140-145.

[111] Hellinga, H. W. and Evans, P. R. (1987). Mutations in the active site of Escherichia coli phosphofructokinase, Nature 327 : 437-439.

[112] Rosa, M. F. and Correia, I. S. (1996). Intracellular acidification does not account for inhibition of Saccharomyces cerevisiae growth in the presence of ethanol, FEMS Micribiology letters, 135 : 271-274.

[113] Narendranath, N. V. and Power, R. (2005). Relationship between pH and Medium Dissolved Solids in Terms of Growth and Metabolism of Lactobacilli and Saccharomyces cerevisiae during Ethanol Production, Appl Environ Microbiol. 71(5) : 2239-2243.

[114] Tipton, K. F.; Armstrong, R. N.; Bakker, B. M.; Bairoch, A. M.; Cornish-Bowden, A.; Halling, P. J.; Hofmeyr, J.; Leyh, T. S.; Kettner, C.; Raushel, F. M.; Rohwer, J. M.; Schomburg, D. and Steinbeck, C. (2014). Standards for Reporting Enzyme Data: The STRENDA Consortium: What it aims to do and why it should be helpful, Perspectives in Science 1(1-6) : 131-137. https://archive-ouverte.unige.ch/unige:39293.

[115] Heinisch, J. J.: Boles, E. and Timpel C. (1996). A Yeast Phosphofructokinase Insensitive to the Allosteric Activator Fructose 2,6-Bisphosphate. The Journal of Biological Chemistry. 271 : 15928-15933.

[116] Cleland, W. W. (2003). The Use of Isotope Effects to Determine Enzyme Mechanisms. The Journal of Biological Chemistry 278 : 51975-51984.

[117] Richards, J. H. (1970). 6 Kinetic Isotope Effects in Enzymic Reactions. The Enzymes, 2 : 321-333.

Appendix A

NMRPy scripts

A.1 PGI

```
import nmrpy
import pylab
import pickle



fnm = 'yeast_160611_20g6p_pgi_ph7_10%lys.fid'
p = nmrpy.data_objects.FidArray.from_path(fnm)
p.emhz_fids(lb=1) #apodisation
p.zf_fids()  #zero-filling
p.ft_fids()  #fourier-transforming
p.phase_correct_fids()


#p.peakpicker_traces()


peaks = [ 3.72453339,  3.66959839,  3.16907956, -0.54513639]
ranges = [[ 4.  ,  3.43],
       [ 3.4 ,  2.92],
       [-0.5 , -0.6 ]]

for fid in p.get_fids():
    fid.peaks = peaks
    fid.ranges = ranges

p.real_fids()
p.norm_fids()
p.deconv_fids()

p.save_to_file(filename=fnm+'.nmrpy')
p = nmrpy.data_objects.FidArray.from_path(fid_path=fnm+'.nmrpy')

ints = p.deconvoluted_integrals.transpose()

```

```
intdict = {
'g6p': ints[0]+ints[1],
'f6p': ints[2],
'tep': ints[3],
}

intdict['tep'] *= 1.2
intdict = {k:5.0*v/intdict['tep'].mean() for k, v in intdict.
    items()}

fig = pylab.figure()
ax = fig.add_subplot(111)
for k,v in intdict.items():
    ax.plot(p.t, v, label=k)
ax.legend()
fig.show()

fig.savefig('plot_%s.pdf' % fnm,format='pdf')

intdict.pop('tep')
intdict['sundry'] = {'rt':p.t[0], 'tp':1.038*0.1, 'mg':10.0}
with open('nmr_%s.npy'%fnm,'wb') as f:
    pickle.dump(intdict, f)

pickle.dump(intdict, open('data_%s.t' % fnm, 'wb'))
```

A.2 PFK

```
import nmrpy
import numpy
import pylab
import pickle



fnm = 'yeast_160531_20f6p_5atp_pfk_ph7_10%lys.fid'
p = nmrpy.data_objects.FidArray.from_path(fnm)
p.emhz_fids(lb=5) #apodisation
p.zf_fids()  #zero-filling
p.ft_fids()  #fourier-transforming
p.phase_correct_fids()

p.save_to_file(filename=fnm+'.nmrpy')
p = nmrpy.data_objects.FidArray.from_path(fid_path=fnm+'.nmrpy')

peaks = [ 4.06940307,  3.8984942 ,  3.64823478,  3.57804007,
    3.48953369,
        3.25758594,  3.20875484,  3.08362513,  2.98291097,
            2.95239153,
        1.40810782, -0.55124028, -5.72123357, -5.81584383,
            -6.19123296,
```

```
        -6.29194711]

ranges = [[ 4.13,  2.71],
        [ 1.5 ,  1.14],
        [-0.5 , -0.61],
        [-5.57, -5.94],
        [-6.13, -6.37]]

for fid in p.get_fids():
    fid.peaks = peaks
    fid.ranges = ranges

p.real_fids()
p.norm_fids()
p.deconv_fids()

p.save_to_file(filename=fnm+'.nmrpy')
p = nmrpy.data_objects.FidArray.from_path(fid_path=fnm+'.nmrpy')

index = [
    [[0,99], {'g6p': [2,3,4], 'f6p':[7], 'fbp':[0,1,5,6,8,9], '
       phos':[10], 'tep':[11], 'adp':[14,15], 'atp':[12,13]}],
    ]

names = ['g6p', 'fbp', 'f6p', 'phos', 'tep', 'adp', 'atp']
ints_d = p.deconvoluted_integrals
ints = []
for i in index:
    if len(i[0]) == 1:
        j = i[0][0]
        int_d = {}
        for k in names:
            if k in i[1]:
                int_d[k] = sum(ints_d[j][numpy.array(i[1][k])])
            else:
                int_d[k] = 0.0
        ints.append(int_d)
    if len(i[0]) == 2:
        for j in range(i[0][0], i[0][1]):
            int_d = {}
            for k in names:
                if k in i[1]:
                    int_d[k] = sum(ints_d[j][numpy.array(i[1][k
                       ])])
                else:
                    int_d[k] = 0.0
            ints.append(int_d)

intdict = {i: numpy.array([j[i] for j in ints]) for i in names}

intdict['tep'] *= 1.2
```

```
70 intdict['fbp'] *= 0.5
71 intdict = {k:5.0*v/intdict['tep'].mean() for k, v in intdict.
       items()}
72
73
74 intdict['f6p'][0] = intdict['f6p'][0]+intdict['fbp'][0]
75 intdict['fbp'][0] = 0
76
77 intdict['f6p'][5] = numpy.mean([intdict['f6p'][4], intdict['f6p'
       ][6]])
78 intdict['fbp'][5] = numpy.mean([intdict['fbp'][4], intdict['fbp'
       ][6]])
79
80 fig = pylab.figure()
81 ax = fig.add_subplot(111)
82 for k,v in intdict.items():
83     ax.plot(v, label=k)
84 ax.legend()
85 fig.show()
86 fig.savefig('plot_%s.pdf' % fnm,format='pdf')
87 intdict.pop('tep')
88 intdict['sundry'] = {'rt':p.t[0], 'tp':0.925*0.1, 'mg':10.0}
89 with open('nmr_%s.npy'%fnm,'wb') as f:
90     pickle.dump(intdict, f)
91
92 pickle.dump(intdict, open('data_%s.t' % fnm, 'wb'))
```

Appendix B

Data compilation scripts

B.1 PGI

```
import sys
import os
import scipy as sp, numpy as np
#from pylab import *
from scipy.integrate import odeint
from glob import glob
import pickle


k_precision = 1e-3

mod_dir = os.getcwd()+'/'

data_filenames = sorted(glob('dump/*.npy'))
data = {i:pickle.load(open(i,'rb')) for i in data_filenames}

sundry_dict = {}
for k, v in data.items():
    print(v)
    sundry_dict[k] = v.pop('sundry')

"""
Define species and create experimental data object filling in
   empty species
"""

species_list = ['adp',
         'adpfree',
         'atp',
         'atpfree',
         'f6p',
         'fbp',
         'g6p',
         'mg',
         'mg2adp',
```

```
35          'mg2atp',
36          'mgadp',
37          'mgatp',
38          #'pep',
39          'phos']
40
41 species_list.sort()
42 species_list = np.array(species_list)
43
44 specnames = []
45
46 def remove_negatives(dic):
47     for k,v in dic.items():
48         v[np.where(v<=0)[0]] = k_precision
49
50 def create_fulldict(dic, all_species_names):
51     dic_len = len(list(dic.values())[0])
52     for species in all_species_names:
53         if species not in dic:
54             dic[species] = k_precision*np.ones(dic_len)
55     return dic
56
57 #quick 'n dirty linear approximation of the initial
     concentrations
58 def f_lin(y):
59     i = 4 #number of points to fit
60     y = y[:i]
61     A = np.array([np.arange(i),np.ones(i)])
62     p = np.linalg.lstsq(A.T,y)[0]
63     return p[1]
64
65 def get_inits(dic):
66     inits = {k:f_lin(v) for k,v in dic.items()}
67     return inits
68
69 def set_inits(dic):
70     inits = get_inits(dic)
71     for k in inits:
72         if inits[k] <= 0:
73            inits[k] = k_precision
74         dic[k][0] = inits[k]
75
76
77 #Mg equilibration
78 #================
79 #This section involves using a small model of the Mg binding
     reactions to
80 #simulate the equilibration of Mg binding. That is, known
     initial concentrations
81 #of all the Mg-binding compounds (and Mg) are used, and the
     model is allowed to
```

```
82 #run to equilibrium. The equilibrium values are used as the true
       initial
83 #concentrations of the full model simulation.
84
85 def f(y, t):
86     mg,atpfree,mgatp,mg2atp,adpfree,mgadp,mg2adp = y
87
88     vf1atp = mg*atpfree
89     vr1atp = mgatp/19.05
90     vf2atp = mg*mgatp
91     vr2atp = mg2atp/0.05
92
93     k = 1
94     vf1adp = k*mg*adpfree
95     vr1adp = k*mgadp/2.0
96     vf2adp = k*mg*mgadp
97     vr2adp = k*mg2adp/0.0186
98
99     datpfree    = vr1atp - vf1atp
100    dmgatp      = vf1atp - vr1atp - vf2atp + vr2atp
101    dmg2atp     = vf2atp - vr2atp
102    dadpfree    = vr1adp - vf1adp
103    dmgadp      = vf1adp - vr1adp - vf2adp + vr2adp
104    dmg2adp     = vf2adp - vr2adp
105    dmg         = vr1atp - vf1atp + vr1adp - vf1adp + vr2atp -
           vf2atp + vr2adp - vf2adp
106
107    return [dmg,datpfree,dmgatp,dmg2atp,dadpfree,dmgadp,dmg2adp]
108
109
110 def getEquilibriumAXP(mg,atpfree,adpfree):
111    mgatp = 0.
112    mg2atp = 0.
113    mgadp = 0.
114    mg2adp = 0.
115    y0 = [mg,atpfree,mgatp,mg2atp,adpfree,mgadp,mg2adp] # inits
116    t  = np.mgrid[0:20:100j]
117    soln = sp.integrate.odeint(f, y0, t)
118    return dict(list(zip(['mg','atpfree','mgatp','mg2atp','
           adpfree','mgadp','mg2adp'],soln[-1])))
119
120 def setEquilibriumAXP(dic,mg):
121    adp, atp = dic['adp'][0], dic['atp'][0]
122    soln = getEquilibriumAXP(mg, atp, adp)
123    for k,v in soln.items():
124        dic[k][0] = v
125
126 for k,v in data.items():
127    remove_negatives(v)
128    create_fulldict(v, species_list)
129    set_inits(v)
```

```
130     setEquilibriumAXP(v, sundry_dict[k]['mg'])
131
132
133 pickle.dump(data, open("./d_data.t", "wb"))
134 pickle.dump(sundry_dict, open("./d_sundries.t", "wb"))
```

B.2 Data compilation and ATP,ADP and $Mg^{2}+$ complex formation model

B.2.1 PFK

```
1  import sys
2  import os
3  import scipy as sp, numpy as np
4  #from pylab import *
5  from scipy.integrate import odeint
6  from glob import glob
7  import pickle
8
9
10 k_precision = 1e-3
11
12 mod_dir = os.getcwd()+'/'
13
14 data_filenames = sorted(glob('dump/*.npy'))
15 data = {i:pickle.load(open(i,'rb')) for i in data_filenames}
16
17 sundry_dict = {}
18 for k, v in data.items():
19     print(v)
20     sundry_dict[k] = v.pop('sundry')
21
22 """
23 Define species and create experimental data object filling in
      empty species
24 """
25
26 species_list = ['adp',
27          'adpfree',
28          'atp',
29          'atpfree',
30          'f6p',
31          'fbp',
32          'g6p',
33          'mg',
34          'mg2adp',
35          'mg2atp',
36          'mgadp',
37          'mgatp',
38          #'pep',
39          'phos']
```

```
40
41 species_list.sort()
42 species_list = np.array(species_list)
43
44 specnames = []
45
46 def remove_negatives(dic):
47     for k,v in dic.items():
48         v[np.where(v<=0)[0]] = k_precision
49
50 def create_fulldict(dic, all_species_names):
51     dic_len = len(list(dic.values())[0])
52     for species in all_species_names:
53         if species not in dic:
54             dic[species] = k_precision*np.ones(dic_len)
55     return dic
56
57 #quick 'n dirty linear approximation of the initial
      concentrations
58 def f_lin(y):
59     i = 4 #number of points to fit
60     y = y[:i]
61     A = np.array([np.arange(i),np.ones(i)])
62     p = np.linalg.lstsq(A.T,y)[0]
63     return p[1]
64
65 def get_inits(dic):
66     inits = {k:f_lin(v) for k,v in dic.items()}
67     return inits
68
69 def set_inits(dic):
70     inits = get_inits(dic)
71     for k in inits:
72         if inits[k] <= 0:
73             inits[k] = k_precision
74         dic[k][0] = inits[k]
75
76
77 #Mg equilibration
78 #================
79 #This section involves using a small model of the Mg binding
      reactions to
80 #simulate the equilibration of Mg binding. That is, known
      initial concentrations
81 #of all the Mg-binding compounds (and Mg) are used, and the
      model is allowed to
82 #run to equilibrium. The equilibrium values are used as the true
       initial
83 #concentrations of the full model simulation.
84
85 def f(y, t):
```

```
86      mg,atpfree,mgatp,mg2atp,adpfree,mgadp,mg2adp = y
87
88      vf1atp = mg*atpfree
89      vr1atp = mgatp/19.05
90      vf2atp = mg*mgatp
91      vr2atp = mg2atp/0.05
92
93      k = 1
94      vf1adp = k*mg*adpfree
95      vr1adp = k*mgadp/2.0
96      vf2adp = k*mg*mgadp
97      vr2adp = k*mg2adp/0.0186
98
99      datpfree    = vr1atp - vf1atp
100     dmgatp      = vf1atp - vr1atp - vf2atp + vr2atp
101     dmg2atp     = vf2atp - vr2atp
102     dadpfree    = vr1adp - vf1adp
103     dmgadp      = vf1adp - vr1adp - vf2adp + vr2adp
104     dmg2adp     = vf2adp - vr2adp
105     dmg         = vr1atp - vf1atp + vr1adp - vf1adp + vr2atp -
           vf2atp + vr2adp - vf2adp
106
107     return [dmg,datpfree,dmgatp,dmg2atp,dadpfree,dmgadp,dmg2adp]
108
109
110 def getEquilibriumAXP(mg,atpfree,adpfree):
111     mgatp = 0.
112     mg2atp = 0.
113     mgadp = 0.
114     mg2adp = 0.
115     y0 = [mg,atpfree,mgatp,mg2atp,adpfree,mgadp,mg2adp] # inits
116     t  = np.mgrid[0:20:100j]
117     soln = sp.integrate.odeint(f, y0, t)
118     return dict(list(zip(['mg','atpfree','mgatp','mg2atp','
           adpfree','mgadp','mg2adp'],soln[-1])))
119
120 def setEquilibriumAXP(dic,mg):
121     adp, atp = dic['adp'][0], dic['atp'][0]
122     soln = getEquilibriumAXP(mg, atp, adp)
123     for k,v in soln.items():
124         dic[k][0] = v
125
126 for k,v in data.items():
127     remove_negatives(v)
128     create_fulldict(v, species_list)
129     set_inits(v)
130     setEquilibriumAXP(v, sundry_dict[k]['mg'])
131
132
133 pickle.dump(data, open("./d_data.t", "wb"))
134 pickle.dump(sundry_dict, open("./d_sundries.t", "wb"))
```

Appendix C

Fitting scripts

C.1 PGI model

```
1
2
3 class parameter_set(object):
4     def __init__(self, dic):
5             for i in dic:
6                     setattr(self,i,dic[i])
7
8 class model:
9     # solve the system dy/dt = f(y, t)
10    @staticmethod
11    def f(y, t, pardic):
12            g6p,f6p = y
13            p = parameter_set(pardic)
14            p.vf_pgi =p.keq_pgi*p.k_g6p_pgi*p.vr_pgi/p.k_f6p_pgi
15            vpgi = p.vf_pgi*p.e_t*(g6p/p.k_g6p_pgi)*(1-(f6p/g6p)
                  /p.keq_pgi)/(1+((g6p/p.k_g6p_pgi)+(f6p/p.
                  k_f6p_pgi)))
16
17            f_g6p = -vpgi
18            f_f6p = vpgi
19            return [f_g6p,f_f6p]
20
21    def __init__(self):
22        self.parameter_dict = {
23
24                    "k_f6p_pgi"    : [0.18      , 1    , 20 ],
25                    "k_g6p_pgi"    : [1.52      , 1    , 20 ],
26                    "keq_pgi"      : [0.231      , 0.238    , 1
                        ],
27                    "vr_pgi"       : [4.46     , 4.46       , 4.46
                        ],
28                    "vf_pgi"       : [5.07      , 5.07       , 9.7
                        ],
29                    "e_t"          : [0.0011      , 1e-1    , 2
                        ],
```

```
                    }

        self.fitted_params = [
                     'k_f6p_pgi',
                     'k_g6p_pgi',
                     'keq_pgi',
                     'vr_pgi',
                     'vf_pgi',
                    ]

        self.par_init_lims = {}
        for i in self.fitted_params:
            self.par_init_lims[i] = [self.parameter_dict[i][1],
               self.parameter_dict[i][2]]
        self.parameters = {}
        for i in self.parameter_dict:
            self.parameters[i] = self.parameter_dict[i][0]

        #order in which the ode function returns rates
        self.species = ['g6p','f6p']
        self.crit = ['g6p','f6p']

if __name__=='__main__':
    print('importing model')
    m = model()
```

C.2 PFK model

```


class parameter_set(object):
    def __init__(self, dic):
            for i in dic:
                    setattr(self,i,dic[i])

class model:

    # solve the system dy/dt = f(y, t)
    @staticmethod
    def f(y, t, pardic):
            atpfree,mgatp,mg,adpfree,mgadp,g6p,f6p,fbp,phos,
               mg2atp,mg2adp = y
            p = parameter_set(pardic)
            #uni-uni Hill
            vpgi = p.vf_pgi*p.e_t*(g6p/p.k_g6p_pgi)*(1-(f6p/g6p)
               /p.keq_pgi)/(1+((g6p/p.k_g6p_pgi)+(f6p/p.
               k_f6p_pgi)))
            #bi-bi Hill with atp allosteric
            vpfk = (p.vf_pfk*p.e_t*((((f6p/p.k_f6p_pfk)*(mgatp/p
               .k_mgatp_pfk))*(((f6p/p.k_f6p_pfk)+(fbp/p.
               k_fbp_pfk))**(p.h_pfk-1))*(((mgatp/p.k_mgatp_pfk)
```

```
                +(mgadp/p.k_mgadp_pfk))**(p.h_pfk-1))*(1-((fbp*
                mgadp)/(f6p*mgatp))/p.keq_pfk))/(((1+((mgatp/p.
                ki_mgatp_pfk)**(p.h_pfk)))/(1+((p.a_mgatp_pfk**(p
                .h_pfk*4))*((mgatp/p.ki_mgatp_pfk)**(p.h_pfk)))))
                +((1+(p.a_mgatp_pfk**(p.h_pfk*2))*((mgatp/p.
                ki_mgatp_pfk)**(p.h_pfk)))/(1+((p.a_mgatp_pfk**(p
                .h_pfk*4))*((mgatp/p.ki_mgatp_pfk)**(p.h_pfk)))))
                *((((f6p/p.k_f6p_pfk)+(fbp/p.k_fbp_pfk))**(p.
                h_pfk))+(((mgatp/p.k_mgatp_pfk)+(mgadp/p.
                k_mgadp_pfk))**(p.h_pfk)))+(((f6p/p.k_f6p_pfk)+(
                fbp/p.k_fbp_pfk))**(p.h_pfk))*(((mgatp/p.
                k_mgatp_pfk)+(mgadp/p.k_mgadp_pfk))**(p.h_pfk))))
                )

            #atp consumption
            vatp = p.kf_atp*mgatp - p.kr_atp*mgadp*phos
            vadp = p.kf_adp*mgadp - p.kr_adp*phos
            vadp = mgadp - phos/p.keq_adp

            f_atpfree = - mg*atpfree + mgatp/19.05
            f_mgatp = mg*atpfree - mgatp/19.05 - vpfk - (mg*
               mgatp - mg2atp/0.05)
            f_mg = - mg*atpfree + mgatp/19.05 - mg*adpfree +
               mgadp/2.0 - (mg*mgatp - mg2atp/0.05) - (mg*mgadp
               - mg2adp/0.0286)
            f_adpfree = - mg*adpfree + mgadp/2.0
            f_mgadp = mg*adpfree - mgadp/2.0 + vpfk - (mg*mgadp
               - mg2adp/0.0286) - vadp
            f_g6p = -vpgi
            f_f6p = vpgi - vpfk
            f_fbp = vpfk
            f_phos = vadp
            f_mg2atp = mg*mgatp - mg2atp/0.05
            f_mg2adp = mg*mgadp - mg2adp/0.0286
            return [f_atpfree,f_mgatp,f_mg,f_adpfree,f_mgadp,
               f_g6p,f_f6p,f_fbp,f_phos,f_mg2atp,f_mg2adp]

    def __init__(self):
        self.parameter_dict = {
                    "a_mgatp_pfk"    : [1.0      , 0.1    , 1.0 ],
                    "h_pfk"          : [2.0      , 1.0    , 4.0 ],
                    "k_f6p_pgi"      : [1.362151     , 0.0    ,
                       1.362151 ],
                    "k_g6p_pgi"      : [1.52     , 0.01    , 20 ],
                    "k_f6p_pfk"      : [0.5      , 0.01    , 20 ],
                    "k_fbp_pfk"      : [3.5      , 0.01    , 20 ],
                    "k_mgadp_pfk"    : [2.0      , 0.01    , 20 ],
                    "k_mgatp_pfk"    : [0.04     , 0.01    , 20 ],
                    "k_pep_pgi"      : [0.26     , 0.01    , 20 ],
                    "ki_mgatp_pfk"     : [0.4      , 0.01    , 20
                       ],
```

```
50                      "keq_pfk"       : [2290.0   , 1      , 1e4 ],
51                      "keq_pgi"       : [0.252    , 1e-2   , 1   ],
52                      "vf_pfk"        : [0.44     , 0.1    , 2   ],
53                      "vf_pgi"        : [0.7    , 0.1    , 10 ],
54                      "mgatp"           : [0.01       , 1e-1    , 10
                            ],
55                      "e_t"           : [0.011      , 1e-1    , 1
                           ],
56                      "kf_adp"        : [2e-3     , 1e-4   , 1e-1  ],
57                      "kr_adp"       : [1e-3     , 1e-4   , 1e-1  ],
58                      "kf_atp"        : [2e-3     , 1e-4   , 1e-1  ],
59                      "kr_atp"       : [1e-3     , 1e-4   , 1e-1  ],
60                      "keq_adp"      : [1       , 1e-2   , 1e2 ],
61                      }
62
63          self.fitted_params = [
64                       'a_mgatp_pfk',
65                       #'h_pfk',
66                       'k_f6p_pfk',
67                       #'k_f6p_pgi',
68                       'k_fbp_pfk',
69                       #'k_g6p_pgi',
70                       'k_mgadp_pfk',
71                       'k_mgatp_pfk',
72                       #'k_pep_pgi',
73                       'ki_mgatp_pfk',
74                       'keq_pfk',
75                       #'keq_pgi',
76                       'vf_pfk',
77                       #'vf_pgi',
78                       'kf_adp',
79                       'kr_adp',
80                       'kf_atp',
81                       'kr_atp',
82                       'keq_adp',
83                      ]
84
85 #         self.par_init_lims  = {i:[self.parameter_dict[i][1],
       self.parameter_dict[i][2]] for i in self.parameter_dict}
86 #         self.parameter_dict = {i:self.parameter_dict[i][0] for
       i in self.parameter_dict}
87          self.par_init_lims = {}
88          for i in self.fitted_params:
89              self.par_init_lims[i] = [self.parameter_dict[i][1],
                   self.parameter_dict[i][2]]
90          self.parameters = {}
91          for i in self.parameter_dict:
92              self.parameters[i] = self.parameter_dict[i][0]
93
94          #order in which the ode function returns rates
95          self.species = ['atpfree','mgatp','mg','adpfree','mgadp'
```

```
            ,'g6p','f6p','fbp','phos','mg2atp','mg2adp']
96          self.crit = ['atp','adp','g6p','f6p','fbp']#,'phos']
97
98
99
100 if __name__=='__main__':
101     print('importing model')
102     m = model()
```

C.3 PGI fitting

```
1  import sys
2  import os
3  import scipy as sp, numpy as np
4  import matplotlib
5  #matplotlib.use('Agg')
6  from multiprocessing import Pool
7  from glob import glob
8  import datetime
9  import pickle
10 import pylab as pl
11 from subprocess import call
12 import lmfit
13 from scipy.optimize import leastsq
14 from time import time
15
16
17
18 def plot_dict_of_dicts(d1,d2,f1=['','o', 2],f2=['-','', 2], fnm=
       None):
19     """
20     This assumes the dictionaries have identical keys.
21     """
22     fig = pl.figure(figsize=[5*len(data),6])
23     axs = [fig.add_subplot(1, len(data), i+1) for i in range(len
          (data))]
24     var = {j for i in [list(v.keys()) for k,v in d1.items()]+[
          list(v.keys()) for k,v in d1.items()] for j in i}
25     cls = dict(list(zip(var,pl.cm.Set1(np.mgrid[0:1:np.complex(
          len(var))]))))
26     plts2 = []
27     for ax,d in zip(axs,d1):
28         for k,v in d1[d].items():
29             ax.plot(v, ls=f1[0], marker=f1[1], lw=f1[2], color=
                  cls[k])
30     #for ax,d in zip(axs,d2):
31         for k,v in d2[d].items():
32             plts2.append(ax.plot(v, ls=f2[0], marker=f2[1], lw=
                  f2[2], color=cls[k]))
33     for ax,d in zip(axs,d1):
34         ax.set_title(d)
```

```
35      for i in axs:
36          box = i.get_position()
37          i.set_position([box.x0, box.y0*0.8, box.width, box.
                height*0.8])
38
39      fig.legend([i[0] for i in plts2], list(d2.values())[0].keys
            (),
40                     loc="upper center",
41                     ncol=5,
42                     )
43      if fnm is not None:
44          fig.savefig(fnm, format='pdf')
45      pl.show()
46
47
48  def f_sim(mod, pardic, inits, t):
49      integration = sp.integrate.odeint(mod.f, inits, t, args=
            tuple([pardic]))
50      return dict(list(zip(mod.species, np.transpose(integration))
            ))
51
52  def fp(pardic, data):
53      result = {}
54      for k,v in data.items():
55          inits = [v[s][0] for s in m.species]
56          #dataset-specific repetition time and total protein
57          t = sundries[k]['rt']*np.arange(len(list(v.values())[0])
                )
58          pardic['e_t'] = sundries[k]['tp']
59          result[k] = f_sim(m, pardic, inits, t)
60      return result
61
62
63  def fp_with_min(pardic, data, mins=None):
64      result = {}
65      for k,v in data.items():
66          inits = [v[s][0] for s in m.species]
67          #dataset-specific repetition-time and total protein
68          t = sundries[k]['rt']*np.arange(len(list(v.values())[0])
                )
69          pardic['e_t'] = sundries[k]['tp']
70          if mins is not None:
71              p = lmfit.Parameters()
72              for i in mins:
73                  p.add(i, value=pardic[i], vary=True, min=0)
74              bf = np.array([pardic[i] for i in mins])
75              mm = lmfit.minimize(f_lin_res, p, args=[pardic,
                    inits, t, v])
76              for i in mm.params:
77                  pardic[i] = mm.params[i].value
78              af = np.array([pardic[i] for i in mins])
```

```
79          result[k] = f_sim(m, pardic, inits, t)
80          #print 'diff:', bf-af, mm.success
81      return result
82
83  def f_lin_res(pars, pardic, inits, t, d):
84      for i in pars:
85          pardic[i] = pars[i].value
86      sim = f_sim(m, pardic, inits, t)
87      sim['atp'] = sum([sim[met] for met in ['atpfree','mgatp','
           mg2atp']],0)
88      sim['adp'] = sum([sim[met] for met in ['adpfree','mgadp','
           mg2adp']],0)
89      crit = ['atp','adp','g6p','f6p','fbp']#,'phos']
90      err = []
91      err += [d[c]-sim[c] for c in crit]
92      #err += [(d[c]-sim[c])/sim[c] for c in crit]
93      err = [j for i in err for j in i]
94      #catch nans and convert them to infs
95      for i in range(len(err)):
96          if np.isnan(err[i]):
97              err[i] == np.inf
98      #print 'during', pars
99      #print sum(np.array(err)**2)
100     return err
101
102
103 def get_res_from_list(p):
104     pardic = m.parameters.copy()
105     for i,j in zip(sorted(m.fitted_params), p):
106         pardic[i] = j
107     err = res(pardic)
108     return err
109
110 def get_res_from_pars(p):
111     pardic = m.parameters.copy()
112     for i in p:
113         pardic[i] = p[i].value
114     err = res(pardic)
115     return err
116
117 def res(pardic):
118     sim = fp(pardic,data)
119     crit = m.crit
120     err = []
121     for k in data:
122         err += [data[k][c]-sim[k][c] for c in crit]
123         #err += [(data[k][c]-sim[k][c])/sim[k][c] for c in crit]
124     err = [j for i in err for j in i]
125
126     #catch nans and convert them to infs
127     for i in range(len(err)):
```

```
128         if np.isnan(err[i]):
129             err[i] == np.inf
130     open('text.h', 'a').write('%f (%i)\n'%(sum(np.array(err)**2)
            ,fit_number))
131     return err #[sum(np.array(err, dtype='f8')**2)]
132
133 #"""
134 #These two functions take a series of parameter values (as
        fractions of the
135 #parameter constraints, 0 -> 1) and a parameter constraint
        dictionary, and output
136 #a parameter dictionary. logconvdic() does this on a logarithmic
         basis so as not
137 #to be biased toward higher parameter values.
138 #"""
139 #
140 def convdic(vn,par_init_lims):
141     a = np.array([par_init_lims[i] for i in par_init_lims]).
            transpose()
142     da = a[0]+vn*(a[1]-a[0])
143     newpars =  dict(list(zip(list(par_init_lims.keys()),da)))
144     pardic = m.parameters.copy()
145     for i in newpars:
146         pardic[i] = newpars[i]
147     return pardic
148
149 def logconvdic(vn, par_init_lims):
150     constraints = np.log(list(par_init_lims.values())).transpose
            ()
151     constraint_diff = vn*(constraints[1]-constraints[0])
152     newpars = dict(list(zip(list(par_init_lims.keys()), np.exp(
            constraints[0]+constraint_diff))))
153     pardic = m.parameters.copy()
154     for i in newpars:
155         pardic[i] = newpars[i]
156     return pardic
157
158
159 #sim_data = fp(logconvdic([0.5]*len(m.fitted_params), m.
        par_init_lims), data)
160
161 mod_dir = os.getcwd()
162
163 data = pickle.load(open('%s/data/d_data.t' % mod_dir,'rb'))
164 sundries = pickle.load(open('%s/data/d_sundries.t' % mod_dir,'rb
        '))
165
166 #original_pardic = '../1_ga/results/latest.t'
167 original_pardic = 'results/latest.t'
168 #fitted_pardic = pickle.load(open(sys.argv[1],'rb'))
169 #fitted_pardic = pickle.load(open(original_pardic,'rb'))
```

```
170
171 sys.path.append(mod_dir+'/data')
172 from d_model_pgi import *
173 m = model()
174
175 #m.parameters.update(fitted_pardic)
176 #m.parameters.update({k:v for k,v in fitted_pardic.iteritems()
       if k in m.fitted_params})
177
178 fitted_pardic = m.parameters.copy()
179 print('fitting:', m.fitted_params)
180
181 lims = False
182 use_leastsq = False
183
184 #os.system('rm %s/text*'%mod_dir)
185 open('text.h', 'w').write('#beginning fit\n')
186
187 pardic = m.parameters.copy()
188 pardic_err = {i:0.0 for i in m.parameters}
189
190 fit_number = 0
191 begin_time = time()
192 if use_leastsq:
193     p = [fitted_pardic[i] for i in sorted(m.fitted_params)]
194     orig_ssr = sum(np.array(get_res_from_list(p))**2)
195     print('orig_ssr', orig_ssr)
196     p, covx, details, msg, scs = leastsq(get_res_from_list, p,
          full_output=True)
197     fitted_pars = dict(list(zip(sorted(m.fitted_params), p)))
198     pardic_err = dict(list(zip(sorted(m.fitted_params), np.sqrt(
          covx.diagonal()))))
199     pardic.update(fitted_pars)
200 else:
201     p = lmfit.Parameters()
202     for i in m.fitted_params:
203         if lims:
204             p.add(i, value=m.parameters[i], min=m.par_init_lims[
                  i][0], max=m.par_init_lims[i][1])
205         else:
206             p.add(i, value=m.parameters[i], min=0)
207     orig_ssr = sum(np.array(get_res_from_pars(p))**2)
208     print('orig_ssr', orig_ssr)
209     for fit_method in ['leastsq', 'nelder', 'nelder']:
210         fit_number += 1
211         mmz = lmfit.minimize(get_res_from_pars,
212                              p,
213                              method=fit_method,
214                              #ftol=1e-12,
215                              #xtol=1e-12,
216                              #maxfev=1000000,
```

```
                            )
        for k,v in mmz.params.items():
            p[k].value = v
        #fit_method = 'leastsq'
    for k,v in mmz.params.items():
        pardic[k] = v.value
        pardic_err[k] = v.stderr
    if hasattr(mmz, 'success'):
        print('Success? %s (%s)' % (mmz.success, mmz.message))

end_time = time()
print('completed in: %.2f min.' % ((end_time-begin_time)/60.0))
print('final:', sum(np.array(res(pardic))**2))


#model simulation dictionary
sim_data = fp(pardic,data)
#for k,v in sim_data.iteritems():
#    v['atp'] = sum([v[met] for met in ['atpfree','mgatp','
    mg2atp']],0)
#    v['adp'] = sum([v[met] for met in ['adpfree','mgadp','
    mg2adp']],0)

if fit_method != 'leastsq':
    pardic_err = {i:0.0 for i in pardic_err}

for k in m.fitted_params:
    print('%s: %f +- %f'%(k, pardic[k], pardic_err[k]))

plot_dict_of_dicts(data, sim_data, fnm='plot.pdf')


time_completion = '_'.join([str(i) for i in datetime.datetime.
    now().timetuple()[:-4]])
pickle.dump(pardic,open("results/%s.t" % time_completion, "wb"))
pickle.dump(data,open("results/latest.data", "wb"))
pickle.dump(pardic,open("results/latest.t", "wb"))
```

C.4 PFK fitting

```
import sys
import os
import scipy as sp, numpy as np
import matplotlib
#matplotlib.use('Agg')
from multiprocessing import Pool
from glob import glob
import datetime
import pickle
import pylab as pl
import pylab
```

```
12 from subprocess import call
13 import lmfit
14 from scipy.optimize import leastsq
15 from time import time
16
17
18
19 def plot_dict_of_dicts(d1,d2,f1=['','o', 2],f2=['-','', 2], fnm=
       None):
20     """
21     This assumes the dictionaries have identical keys.
22     """
23     fig = pl.figure(figsize=[15,15])
24     dim = np.ceil(np.sqrt(len(data)))
25     axs = [fig.add_subplot(dim, dim, i+1) for i in range(len(
           data))]
26     var = {j for i in [list(v.keys()) for k,v in d1.items()]+[
           list(v.keys()) for k,v in d1.items()] for j in i}
27     cls = dict(list(zip(var,pl.cm.Set1(np.mgrid[0:1:np.complex(
           len(var))]))))
28     plts2 = []
29     for ax,d in zip(axs,d1):
30         for k,v in d1[d].items():
31             ax.plot(v, ls=f1[0], marker=f1[1], lw=f1[2], color=
                   cls[k])
32     #for ax,d in zip(axs,d2):
33         for k,v in d2[d].items():
34             plts2.append(ax.plot(v, ls=f2[0], marker=f2[1], lw=
                   f2[2], color=cls[k]))
35     for ax,d in zip(axs,d1):
36         ax.set_title(d)
37     for i in axs:
38         box = i.get_position()
39         i.set_position([box.x0, box.y0*0.8, box.width, box.
               height*0.8])
40
41     fig.legend([i[0] for i in plts2], list(d2.values())[0].keys
           (),
42                loc="upper center",
43                ncol=5,
44                )
45     if fnm is not None:
46         fig.savefig(fnm, format='pdf')
47     pl.show()
48
49
50 def f_sim(mod, pardic, inits, t):
51     integration = sp.integrate.odeint(mod.f, inits, t, args=
           tuple([pardic]))
52     return dict(list(zip(mod.species, np.transpose(integration))
           ))
```

```
53
54 def fp(pardic, data):
55     result = {}
56     for k,v in data.items():
57         inits = [v[s][0] for s in m.species]
58         #dataset-specific repetition time and total protein
59         t = (sundries[k]['rt']*np.arange((len(list(v.values())
               [0]))+1)[1:])
60         pardic['e_t'] = sundries[k]['tp']
61         result[k] = f_sim(m, pardic, inits, t)
62     return result
63
64 def fp2(kp):
65     k,pardic = kp
66     v = data[k]
67     inits = [v[s][0] for s in m.species]
68     #dataset-specific repetition time and total protein
69     t = (sundries[k]['rt']*np.arange((len(list(v.values())[0]))
          +1)[1:])
70     pardic['e_t'] = sundries[k]['tp']
71     result = f_sim(m, pardic, inits, t)
72     return result
73
74 def fp(pardic):
75     proc_pool = Pool(7)
76     #results = proc_pool.map(fp2, [[i,pardic] for i in data.keys
          ()])
77     results = proc_pool.map(fp2, list(zip(list(data.keys()),[
          pardic]*len(list(data.keys())))))
78     proc_pool.close()
79     proc_pool.join()
80     return dict(list(zip(list(data.keys()),results)))
81
82
83 def fp_with_min(pardic, data, mins=None):
84     result = {}
85     for k,v in data.items():
86         inits = [v[s][0] for s in m.species]
87         #dataset-specific repetition-time and total protein
88         t = (sundries[k]['rt']*np.arange((len(list(v.values())
               [0]))+1)[1:])
89         pardic['e_t'] = sundries[k]['tp']
90         if mins is not None:
91             p = lmfit.Parameters()
92             for i in mins:
93                 p.add(i, value=pardic[i], vary=True, min=0)
94             bf = np.array([pardic[i] for i in mins])
95             mm = lmfit.minimize(f_lin_res, p, args=[pardic,
                   inits, t, v])
96             for i in mm.params:
97                 pardic[i] = mm.params[i].value
```

```
98              af = np.array([pardic[i] for i in mins])
99          result[k] = f_sim(m, pardic, inits, t)
100         #print 'diff:', bf-af, mm.success
101     return result
102
103 def f_lin_res(pars, pardic, inits, t, d):
104     for i in pars:
105         pardic[i] = pars[i].value
106     sim = f_sim(m, pardic, inits, t)
107     sim['atp'] = sum([sim[met] for met in ['atpfree','mgatp','
            mg2atp']],0)
108     sim['adp'] = sum([sim[met] for met in ['adpfree','mgadp','
            mg2adp']],0)
109     crit = ['atp','adp','g6p','f6p','fbp']#,'phos']
110     err = []
111     err += [d[c]-sim[c] for c in crit]
112     #err += [(d[c]-sim[c])/sim[c] for c in crit]
113     err = [j for i in err for j in i]
114     #catch nans and convert them to infs
115     for i in range(len(err)):
116         if np.isnan(err[i]):
117             err[i] == np.inf
118     #print 'during', pars
119     #print sum(np.array(err)**2)
120     return err
121
122
123 def get_res_from_list(p):
124     pardic = m.parameters.copy()
125     for i,j in zip(sorted(m.fitted_params), p):
126         pardic[i] = j
127     err = res(pardic)
128     return err
129
130 def get_res_from_pars(p):
131     pardic = m.parameters.copy()
132     for i in p:
133         pardic[i] = p[i].value
134     err = res(pardic)
135     return err
136
137 def res(pardic):
138     #change these lines to fit adp consumption on each iteration
139     sim = fp(pardic)
140     #sim = fp_with_min(pardic,data,mins=['keq_adp'])
141     for k,v in sim.items():
142         v['atp'] = sum([v[met] for met in ['atpfree','mgatp','
                mg2atp']],0)
143         v['adp'] = sum([v[met] for met in ['adpfree','mgadp','
                mg2adp']],0)
144     #crit = ['atp','adp','g6p','f6p','fbp']#,'phos']
```

```
145     crit = m.crit
146     err = []
147     for k in data:
148         err += [data[k][c]-sim[k][c] for c in crit]
149         #err += [(data[k][c]-sim[k][c])/sim[k][c] for c in crit]
150     err = [j for i in err for j in i]
151 
152     #catch nans and convert them to infs
153     for i in range(len(err)):
154         if np.isnan(err[i]):
155             err[i] == np.inf
156     open('text.h', 'a').write('%f (%i)\n'%(sum(np.array(err)**2)
        ,fit_number))
157     return err #[sum(np.array(err, dtype='f8')**2)]
158 
159 #"""
160 #These two functions take a series of parameter values (as
    fractions of the
161 #parameter constraints, 0 -> 1) and a parameter constraint
    dictionary, and output
162 #a parameter dictionary. logconvdic() does this on a logarithmic
     basis so as not
163 #to be biased toward higher parameter values.
164 #"""
165 #
166 def convdic(vn,par_init_lims):
167     a = np.array([par_init_lims[i] for i in par_init_lims]).
        transpose()
168     da = a[0]+vn*(a[1]-a[0])
169     newpars =  dict(list(zip(list(par_init_lims.keys()),da)))
170     pardic = m.parameters.copy()
171     for i in newpars:
172         pardic[i] = newpars[i]
173     return pardic
174 
175 def logconvdic(vn, par_init_lims):
176     constraints = np.log(list(par_init_lims.values())).transpose
        ()
177     constraint_diff = vn*(constraints[1]-constraints[0])
178     newpars = dict(list(zip(list(par_init_lims.keys()), np.exp(
        constraints[0]+constraint_diff))))
179     pardic = m.parameters.copy()
180     for i in newpars:
181         pardic[i] = newpars[i]
182     return pardic
183 
184 
185 #sim_data = fp(logconvdic([0.5]*len(m.fitted_params), m.
    par_init_lims), data)
186 
187 mod_dir = os.getcwd()
```

```
188
189 data = pickle.load(open('%s/data/d_data.t' % mod_dir,'rb'))
190 sundries = pickle.load(open('%s/data/d_sundries.t' % mod_dir,'rb
    '))
191
192 #original_pardic = '../1_ga/results/latest.t'
193 original_pardic = 'results/latest.t'
194 #fitted_pardic = pickle.load(open(sys.argv[1],'rb'))
195 fitted_pardic = pickle.load(open(original_pardic,'rb'))
196
197
198
199 sys.path.append(mod_dir+'/data')
200 from d_model_pgi_pfk_atp_inh import *
201 m = model()
202
203 m.parameters.update(fitted_pardic)
204 m.parameters.update({k:v for k,v in fitted_pardic.items() if k
    in m.fitted_params})
205
206 print('fitting:', m.fitted_params)
207
208 lims = False
209 use_leastsq = False
210
211 #os.system('rm %s/text*'%mod_dir)
212 open('text.h', 'w').write('#beginning fit\n')
213
214 pardic = m.parameters.copy()
215 pardic_err = {i:0.0 for i in m.parameters}
216
217 fit_number = 0
218 begin_time = time()
219 if use_leastsq:
220     p = [fitted_pardic[i] for i in sorted(m.fitted_params)]
221     orig_ssr = sum(np.array(get_res_from_list(p))**2)
222     print('orig_ssr', orig_ssr)
223     p, covx, details, msg, scs = leastsq(get_res_from_list, p,
        full_output=True)
224     fitted_pars = dict(list(zip(sorted(m.fitted_params), p)))
225     pardic_err = dict(list(zip(sorted(m.fitted_params), np.sqrt(
        covx.diagonal()))))
226     pardic.update(fitted_pars)
227 else:
228     p = lmfit.Parameters()
229     for i in m.fitted_params:
230         if lims:
231             p.add(i, value=m.parameters[i], min=m.par_init_lims[
                i][0], max=m.par_init_lims[i][1])
232         else:
233             p.add(i, value=m.parameters[i], min=0)
```

```
234     orig_ssr = sum(np.array(get_res_from_pars(p))**2)
235     ##fit 1
236     #fit_method='nelder'
237     #fit_number += 1
238     #mmz = lmfit.minimize(get_res_from_pars,
239     #                     p,
240     #                     method=fit_method,
241     #                     #ftol=1e-12,
242     #                     #xtol=1e-12,
243     #                     #maxfev=1000000,
244     #                     )
245     #for k,v in mmz.params.items():
246     #    p[k].value = v
247     ##fits until chisqr no longer changes
248     #old_chisqr = orig_ssr
249     #new_chisqr = mmz.chisqr
250     #while np.abs(new_chisqr - old_chisqr) > 1e-4 and fit_number
            <= 1:
251     ##while new_chisqr != old_chisqr:
252     #    print('new: %f, old: %f'%(new_chisqr, old_chisqr))
253     #    fit_number += 1
254     #    mmz = lmfit.minimize(get_res_from_pars,
255     #                         p,
256     #                         method=fit_method,
257     #                         #ftol=1e-12,
258     #                         #xtol=1e-12,
259     #                         #maxfev=1000000,
260     #                         )
261     #    for k,v in mmz.params.items():
262     #        p[k].value = v
263     #    old_chisqr = new_chisqr
264     #    new_chisqr = mmz.chisqr
265     curr_ssr = orig_ssr.copy()
266     for fit_method in ['nelder', 'nelder', 'nelder']:
267         fit_number += 1
268         print('ssr init ({}): {}'.format(fit_number, curr_ssr))
269         mmz = lmfit.minimize(get_res_from_pars,
270                              p,
271                              method=fit_method,
272                              #ftol=1e-12,
273                              #xtol=1e-12,
274                              #maxfev=1000000,
275                              )
276         for k,v in mmz.params.items():
277             p[k].value = v
278         #fit_method = 'leastsq'
279         curr_ssr = mmz.chisqr
280         print('ssr final ({}): {}'.format(fit_number, curr_ssr))
281     for k,v in mmz.params.items():
282         pardic[k] = v.value
283         pardic_err[k] = v.stderr
```

```
284     if hasattr(mmz, 'success'):
285         print('Success? %s (%s)' % (mmz.success, mmz.message))
286 
287 end_time = time()
288 print('completed in: %.2f min.' % ((end_time-begin_time)/60.0))
289 print('final:', sum(np.array(res(pardic))**2))
290 
291 
292 #model simulation dictionary
293 sim_data = fp(pardic)#,data)
294 for k,v in sim_data.items():
295     v['atp'] = sum([v[met] for met in ['atpfree','mgatp','mg2atp
        ']],0)
296     v['adp'] = sum([v[met] for met in ['adpfree','mgadp','mg2adp
        ']],0)
297 
298 if fit_method != 'leastsq':
299     pardic_err = {i:0.0 for i in pardic_err}
300 
301 for k in m.fitted_params:
302     print('%s: %f +- %f'%(k, pardic[k], pardic_err[k]))
303 
304 plot_dict_of_dicts(data, sim_data, fnm='plot.pdf')
305 
306 
307 time_completion = '_'.join([str(i) for i in datetime.datetime.
    now().timetuple()[:-4]])
308 pickle.dump(pardic, open("results/%s.t" % time_completion, "wb")
    )
309 pickle.dump(pardic, open("results/latest.t", "wb"))
310 pickle.dump(data, open("results/latest.data", "wb"))
```

Appendix D

Identifiability analysis scripts

D.1 PGI

D.1.1 Identifiability analysis

```
import sys
import os
import scipy as sp, numpy as np
from scipy.integrate import odeint
import matplotlib
import pylab as pl
from multiprocessing import Pool, cpu_count
from glob import glob
import pickle
import datetime
import lmfit
from time import time

"""

This script successively fits each of the previously fitted
   parameters over a
range of values, and refits the remaining parameters to generate
    profile
likelihoods according to Raue et al. 2009.

base - the base over which to scan parameters
nmb - number of steps over which to scan parameters

"""
base = 2
ormag = 1
nmb = 20*ormag
cpu = 7

def f_sim(mod, pardic, inits, t):
    integration = sp.integrate.odeint(mod.f, inits, t, args=
       tuple([pardic]))
```

```
31      return dict(list(zip(mod.species, np.transpose(integration))
           ))
32
33  def fp(pardic, data):
34      result = {}
35      for k,v in data.items():
36          inits = [v[s][0] for s in m.species]
37          #dataset-specific repetition time and total protein
38          t = sundries[k]['rt']*np.arange(len(list(v.values())[0])
               )
39          pardic['e_t'] = sundries[k]['tp']
40          result[k] = f_sim(m, pardic, inits, t)
41      return result
42
43  def get_res_from_pars(p):
44      pardic = m.parameters.copy()
45      for i in p:
46          pardic[i] = p[i].value
47      err = res(pardic)
48      return err
49
50
51  def res(pardic):
52      sim = fp(pardic,data)
53      crit = m.crit
54      err = []
55      for k in data:
56          err += [data[k][c]-sim[k][c] for c in crit]
57      err = [j for i in err for j in i]
58
59      #catch nans and convert them to infs
60      for i in range(len(err)):
61          if np.isnan(err[i]):
62              err[i] == np.inf
63      return err
64
65  def do_min(d,p_fix):
66      p = lmfit.Parameters()
67      for i in d:
68          p.add(i, value=d[i], min=1e-6)
69      p[p_fix].vary = False
70      mz = lmfit.minimize(get_res_from_pars,p,method=fit_method)
71      return mz.chisqr
72
73  def do_par(pd):
74      begin_time = time()
75      d = {}
76      for i in m.fitted_params:
77          d[i] = m.parameters[i]
78      result = []
79      for i in pd[1:]:
```

```
80          d[pd[0]] = i
81          c2 = do_min(d,pd[0])
82          result.append(c2)
83      end_time = time()
84      run_time = ((end_time-begin_time)/60.0)
85      run_time_total = run_time*len(m.fitted_params)/float(nprocs)
86      return pd+result
87
88  def do_tuple(pd):
89      d = {}
90      for i in m.fitted_params:
91          d[i] = m.parameters[i]
92      d[pd[0]] = pd[1]
93      result = do_min(d,pd[0])
94      return pd+[result]
95
96  def do_n(n):
97      with open('text.h','a') as f:
98          f.write('%i/%i\n'%(n,len(pdic)))
99      return do_tuple(pdic[n])
100
101 #import data
102 mod_dir = os.getcwd()
103 data = pickle.load(open('../2_lsq_lims/results/latest.data','rb'
    ))
104 sundries = pickle.load(open('%s/data/d_sundries.t' % mod_dir,'rb
    '))
105
106 #import previously fitted parameters
107 original_pardic = '../2_lsq_lims/results/latest.t'
108 fitted_pardic = pickle.load(open(original_pardic,'rb'))
109
110 #import model
111 sys.path.append(mod_dir+'/data')
112 from d_model_pgi import *
113 m = model()
114
115 #update model parameters with previously fitted parameters
116 m.parameters.update({k:v for k,v in fitted_pardic.items() if k
    in m.fitted_params})
117
118 #begin fitting
119 print('fitting:', m.fitted_params)
120 open('text.h', 'w').write('#beginning fit\n')
121
122 p = lmfit.Parameters()
123 for i in m.fitted_params:
124     p.add(i, value=m.parameters[i], min=0)
125
126 #get previous chi2
127 orig_chi2 = sum(np.array(get_res_from_pars(p))**2)
```

```
print('orig_chi2', orig_chi2)


#set rng for range of values to assess fixed parameters
rng = np.logspace(-ormag,ormag,nmb, base=base)

#pdic is a list of individual parameter range tests, so that we
    can easily multiprocess
pdic = [l for k in [[[i,j*m.parameters[i]] for j in rng] for i
    in m.fitted_params] for l in k]

#perform fits
begin_time = time()

fit_method = 'nelder'
pool = Pool()
if cpu is not None:
    pool = Pool(cpu)
results = pool.map(do_n, list(range(len(pdic))))

end_time = time()
print('completed in: %.2f min.' % ((end_time-begin_time)/60.0))


pardic = {}
for i in m.fitted_params:
    result_par = np.array([r[1:] for r in results if r[0] == i])
    pardic[i] = list(result_par[:,0])+list(result_par[:,1])

pardic['orig_chi2'] = orig_chi2

time_completion = '_'.join([str(i) for i in datetime.datetime.
    now().timetuple()[:-4]])
pickle.dump(pardic,open("results/%s.t" % time_completion, "wb"))
pickle.dump(pardic,open("results/latest.t", "wb"))

print(time_completion)
with open('text.h','a') as f:
    f.write('finished.\n')
```

D.1.2 Profile likelihood plots

```
import matplotlib
import pylab
import numpy
import pickle
import sys
from scipy.stats import chi2
from scipy.interpolate import UnivariateSpline
import lmfit


```

```
spl_k = 2
spl_s = 1

#ident.t file
original_pardic = '../2_lsq_lims/results/latest.t'
result = 'results/latest.t'

with open('sd.t', 'rb') as f:
    mean_std = float(f.read())

d = pickle.load(open(result, 'rb'))
pardic = pickle.load(open(original_pardic, 'rb'))
orig_chi2 = d.pop('orig_chi2')/mean_std**2


d = {i:numpy.reshape(d[i], [2, -1]).tolist() for i in d}
for k in d:
    d[k][1] = list(numpy.array(d[k][1])/mean_std**2)

chi2_q95 = chi2.ppf(0.95, len(d))

#spline interpolation
spl = {k:UnivariateSpline(v[0], v[1], k=spl_k, s=spl_s) for k, v
    in d.items()}

def y_to_x(s, y, init):
    p1 = lmfit.Parameters()
    p1.add('x', value=0.5*init, max=init, min=-numpy.inf)
    p2 = lmfit.Parameters()
    p2.add('x', value=2.0*init, max=numpy.inf, min=init)
    mz1 = lmfit.minimize(lambda par: s(par['x'].value)-y, p1)
    mz2 = lmfit.minimize(lambda par: s(par['x'].value)-y, p2)
    return [mz1.params['x'].value, mz2.params['x'].value]

ci = {}
for k, v in spl.items():
    ci[k] = y_to_x(spl[k], spl[k](pardic[k])+chi2_q95, pardic[k
       ])


incs = []
for k, v in d.items():
    if True in (v[1] < orig_chi2):
        incs.append(k)

#figure
l = numpy.ceil(numpy.sqrt(len(d)))
k = sorted(d)
y_max = max([j for i in [i[1] for i in list(d.values())] for j
   in i])
y_min = min([j for i in [i[1] for i in list(d.values())] for j
```

```
   in i])
fig = pylab.figure(figsize=[10, 10])
axs = [fig.add_subplot(l, l, i+1)  for i in range(len(d))]
for i in range(len(d)):
    lc = 'k'
    if k[i] in incs:
        lc = 'r'
    x = numpy.array(d[k[i]][0]+[pardic[k[i]]])#/pardic[k[i]]
    ssi = numpy.argsort(x)
    x = x[ssi]
    y = numpy.array(d[k[i]][1]+[orig_chi2])[ssi]
    axs[i].semilogx(x, y, '+', color=lc, mec=lc, basex=10)
    axs[i].semilogx(x, spl[k[i]](x), '-', color='b', mec=lc,
       basex=3)
    axs[i].hlines(orig_chi2+chi2_q95, x[0], x[-1], color='r')
    axs[i].set_xticks([x[0], ci[k[i]][0], ci[k[i]][1], x[-1]])
    axs[i].set_xlim([x[0], x[-1]])
    ylim = axs[i].get_ylim()

    axs[i].vlines(pardic[k[i]], ylim[0], ylim[1], color='b',
       linestyle='dashed')
    axs[i].vlines(ci[k[i]][0], ylim[0], ylim[1], color='b',
       linestyle='solid')
    axs[i].vlines(ci[k[i]][1], ylim[0], ylim[1], color='b',
       linestyle='solid')
    axs[i].fill_betweenx(ylim, ci[k[i]][0], ci[k[i]][1], alpha
       =0.1)

    y = axs[i].get_yticks()
    axs[i].set_yticks([ylim[0], orig_chi2, ylim[1]])
    axs[i].set_yticklabels(['%.3g'%(y/orig_chi2) for y in axs[i
       ].get_yticks()], size=6)
    axs[i].set_xlabel('%s\n(%.3g > %.3g > %.3g)'%(k[i], ci[k[i
       ]][0], pardic[k[i]], ci[k[i]][1]), size=8)

axs[0].set_ylabel(r'$\chi^2/\chi^2_0$', size=15)


fig.subplots_adjust(wspace=0.4, hspace=0.4)
fig.savefig('plot.pdf', format='pdf')
fig.show()
```

D.2 PFK

D.2.1 Identifiability analysis

```

import sys
import os
import scipy as sp, numpy as np
from scipy.integrate import odeint
```

```
6  import matplotlib
7  #matplotlib.use('Agg')
8  import pylab as pl
9  from multiprocessing import Pool, cpu_count
10 from glob import glob
11 import pickle
12 import datetime
13 import lmfit
14 from time import time
15
16
17 """
18
19 This script successively fits each of the previously fitted
       parameters over a
20 range of values, and refits the remaining parameters to generate
       profile
21 likelihoods according to Raue et al. 2009.
22
23 base - the base over which to scan parameters
24 nmb - number of steps over which to scan parameters
25
26 """
27 base = 2
28 ormag = 1
29 nmb = 20*ormag
30 cpu = None
31
32 def f_sim(mod, pardic, inits, t):
33     integration = sp.integrate.odeint(mod.f, inits, t, args=
          tuple([pardic]))
34     return dict(list(zip(mod.species, np.transpose(integration))
          ))
35
36 def fp(pardic, data):
37     result = {}
38     for k,v in data.items():
39         inits = [v[s][0] for s in m.species]
40         #dataset-specific repetition time and total protein
41         t = sundries[k]['rt']*np.arange(len(list(v.values())[0])
              )
42         pardic['e_t'] = sundries[k]['tp']
43         result[k] = f_sim(m, pardic, inits, t)
44     return result
45
46 def get_res_from_pars(p):
47     pardic = m.parameters.copy()
48     for i in p:
49         pardic[i] = p[i].value
50     err = res(pardic)
51     return err
```

```
52
53
54 def res(pardic):
55     sim = fp(pardic,data)
56     for k,v in sim.items():
57         v['atp'] = sum([v[met] for met in ['atpfree','mgatp','
               mg2atp']],0)
58         v['adp'] = sum([v[met] for met in ['adpfree','mgadp','
               mg2adp']],0)
59     crit = m.crit
60     err = []
61     for k in data:
62         err += [data[k][c]-sim[k][c] for c in crit]
63         #err += [(data[k][c]-sim[k][c])/sim[k][c] for c in crit]
64     err = [j for i in err for j in i]
65
66     #catch nans and convert them to infs
67     for i in range(len(err)):
68         if np.isnan(err[i]):
69             err[i] == np.inf
70     #open('text.h', 'ab').write('%f\n'%sum(np.array(err)**2))
71     return err #[sum(np.array(err, dtype='f8')**2)]
72
73 def do_min(d,p_fix):
74     p = lmfit.Parameters()
75     for i in d:
76         p.add(i, value=d[i], min=1e-6)
77     p[p_fix].vary = False
78     mz = lmfit.minimize(get_res_from_pars,p,method=fit_method)
79     #d = {i:p[i].value for i in p}
80     return mz.chisqr
81
82 def do_par(pd):
83     begin_time = time()
84     #d = {i:m.parameters[i] for i in m.fitted_params}
85     #open('text.h','ab').write('%s\n'%pd[0])
86     d = {}
87     for i in m.fitted_params:
88         d[i] = m.parameters[i]
89     result = []
90     for i in pd[1:]:
91         d[pd[0]] = i
92         c2 = do_min(d,pd[0])
93         result.append(c2)
94     end_time = time()
95     run_time = ((end_time-begin_time)/60.0)
96     run_time_total = run_time*len(m.fitted_params)/float(nprocs)
97     #open('text.h','ab').write('%s runtime: %.2f min (total: +-
           %.2f)\n'%(pd[0], run_time, run_time_total))
98     return pd+result
99
```

```
100 def do_tuple(pd):
101     #open('text.h','ab').write('%s\n'%pd[0])
102     d = {}
103     for i in m.fitted_params:
104         d[i] = m.parameters[i]
105     d[pd[0]] = pd[1]
106     result = do_min(d,pd[0])
107     #open('text.h','ab').write('%s completed\n'%(pd[0]))
108     return pd+[result]
109 
110 def do_n(n):
111     with open('text.h','a') as f:
112         f.write('%i/%i\n'%(n,len(pdic)))
113     return do_tuple(pdic[n])
114 
115 #import data
116 mod_dir = os.getcwd()
117 data = pickle.load(open('../2_lsq_lims/results/latest.data','rb'
    ))
118 sundries = pickle.load(open('%s/data/d_sundries.t' % mod_dir,'rb
    '))
119 
120 #import previously fitted parameters
121 original_pardic = '../2_lsq_lims/results/latest.t'
122 fitted_pardic = pickle.load(open(original_pardic,'rb'))
123 
124 #import model
125 sys.path.append(mod_dir+'/data')
126 from d_model_pgi_pfk_atp_inh import *
127 m = model()
128 
129 #update model parameters with previously fitted parameters
130 #m.parameters.update(fitted_pardic)
131 m.parameters.update({k:v for k,v in fitted_pardic.items() if k
    in m.fitted_params})
132 
133 #begin fitting
134 print('fitting:', m.fitted_params)
135 open('text.h', 'w').write('#beginning fit\n')
136 
137 p = lmfit.Parameters()
138 for i in m.fitted_params:
139     p.add(i, value=m.parameters[i], min=0)
140 
141 #get previous chi2
142 orig_chi2 = sum(np.array(get_res_from_pars(p))**2)
143 print('orig_chi2', orig_chi2)
144 
145 
146 #set rng for range of values to assess fixed parameters
147 rng = np.logspace(-ormag,ormag,nmb, base=base)
```

```
148
149 #pdic is a list of individual parameter range tests, so that we
       can easily multiprocess
150 pdic = [l for k in [[[i,j*m.parameters[i]] for j in rng] for i
       in m.fitted_params] for l in k]
151
152 #perform fits
153 begin_time = time()
154
155 fit_method = 'nelder'
156 pool = Pool()
157 if cpu is not None:
158     pool = Pool(cpu)
159 results = pool.map(do_n, list(range(len(pdic))))
160
161 end_time = time()
162 print('completed in: %.2f min.' % ((end_time-begin_time)/60.0))
163
164
165 pardic = {}
166 for i in m.fitted_params:
167     result_par = np.array([r[1:] for r in results if r[0] == i])
168     pardic[i] = list(result_par[:,0])+list(result_par[:,1])
169
170 pardic['orig_chi2'] = orig_chi2
171
172 time_completion = '_'.join([str(i) for i in datetime.datetime.
       now().timetuple()[:-4]])
173 pickle.dump(pardic,open("results/%s.t" % time_completion, "wb"))
174 pickle.dump(pardic,open("results/latest.t", "wb"))
175
176 print(time_completion)
177 with open('text.h','a') as f:
178     f.write('finished.\n')
```

D.2.2 Profile likelihood plots

```
1 import matplotlib
2 #matplotlib.use('Agg')
3 import pylab
4 import numpy
5 import pickle
6 import sys
7 from scipy.stats import chi2
8 from scipy.interpolate import UnivariateSpline
9 import lmfit
10
11
12 spl_k = 1
13 spl_s = 1
14
15 #ident.t file
```

```
original_pardic = '../2_lsq_lims/results/latest.t'
result = 'results/latest.t'

with open('sd.t', 'rb') as f:
    mean_std = float(f.read())

d = pickle.load(open(result, 'rb'))
pardic = pickle.load(open(original_pardic, 'rb'))
orig_chi2 = d.pop('orig_chi2')/mean_std**2


d = {i:numpy.reshape(d[i], [2, -1]).tolist() for i in d}
for k in d:
    d[k][1] = list(numpy.array(d[k][1])/mean_std**2)

chi2_q95 = chi2.ppf(0.95, len(d))

#spline interpolation
spl = {k:UnivariateSpline(v[0], v[1], k=spl_k, s=spl_s) for k, v
    in d.items()}

def y_to_x(s, y, init):
    p1 = lmfit.Parameters()
    p1.add('x', value=0.5*init, max=init, min=-numpy.inf)
    p2 = lmfit.Parameters()
    p2.add('x', value=2.0*init, max=numpy.inf, min=init)
    mz1 = lmfit.minimize(lambda par: s(par['x'].value)-y, p1)
    mz2 = lmfit.minimize(lambda par: s(par['x'].value)-y, p2)
    return [mz1.params['x'].value, mz2.params['x'].value]

ci = {}
for k, v in spl.items():
    ci[k] = y_to_x(spl[k], spl[k](pardic[k])+chi2_q95, pardic[k
       ])

#incs = [i for i in d if d[i][1][0]>orig_chi2 and d[i][1][-1]>
   orig_chi2]
incs = []
for k, v in d.items():
    if True in (v[1] < orig_chi2):
        incs.append(k)

#figure
l = numpy.ceil(numpy.sqrt(len(d)))
k = sorted(d)
y_max = max([j for i in [i[1] for i in list(d.values())] for j
    in i])
y_min = min([j for i in [i[1] for i in list(d.values())] for j
    in i])
fig = pylab.figure(figsize=[10, 10])
axs = [fig.add_subplot(l, l, i+1)  for i in range(len(d))]
```

```
for i in range(len(d)):
    lc = 'k'
    if k[i] in incs:
        lc = 'r'
    x = numpy.array(d[k[i]][0]+[pardic[k[i]]])#/pardic[k[i]]
    ssi = numpy.argsort(x)
    x = x[ssi]
    y = numpy.array(d[k[i]][1]+[orig_chi2])[ssi]
    axs[i].semilogx(x, y, '+', color=lc, mec=lc, basex=10)
    axs[i].semilogx(x, spl[k[i]](x), '-', color='b', mec=lc,
        basex=3)
    #axs[i].loglog(x, y, '-o', color=lc, mec=lc, basex=3, basey
        =10)
    axs[i].hlines(orig_chi2+chi2_q95, x[0], x[-1], color='r')
    axs[i].set_xticks([x[0], ci[k[i]][0], ci[k[i]][1], x[-1]])
    axs[i].set_xlim([x[0], x[-1]])
    #axs[i].set_xticklabels(['%.4g'%xx for xx in x], size=6,
        rotation='vertical')
    #axs[i].set_yticklabels(['%.2f'%yy for yy in axs[i].
        get_yticks()], size=6)
    #axs[i].set_ylim([0.99*orig_chi2, y_max])
    #axs[i].set_ylim([0.9*y_min, y_max])
    #axs[i].set_ylim([0, 2000])
    ylim = [0.95*orig_chi2, 1.15*orig_chi2]
    #axs[i].set_ylim(ylim)
    ylim = axs[i].get_ylim()

    axs[i].vlines(pardic[k[i]], ylim[0], ylim[1], color='b',
        linestyle='dashed')
    axs[i].vlines(ci[k[i]][0], ylim[0], ylim[1], color='b',
        linestyle='solid')
    axs[i].vlines(ci[k[i]][1], ylim[0], ylim[1], color='b',
        linestyle='solid')
    axs[i].fill_betweenx(ylim, ci[k[i]][0], ci[k[i]][1], alpha
        =0.1)

    y = axs[i].get_yticks()
    axs[i].set_yticks([ylim[0], orig_chi2, ylim[1]])
    axs[i].set_yticklabels(['%.3g'%(y/orig_chi2) for y in axs[i
        ].get_yticks()], size=6)
    #axs[i].set_yticks([orig_chi2-0.2*(y[-1]-orig_chi2),
        orig_chi2, y[-1]])
    #axs[i].set_yticklabels(['%.6g'%y for y in axs[i].get_yticks
        ()], size=6)
    #axs[i].set_title(k[i], size=10)
    #axs[i].set_xlabel('%s\n(%.3g-%.3g)'%(k[i].replace('kcat', '
        vmax'), ci[k[i]][0], ci[k[i]][1]), size=8)
    #axs[i].set_xlabel('%s\n(%.3g > %.3g > %.3g)'%(k[i], ci[k[i
        ]][0], pardic[k[i]], ci[k[i]][1]), size=8)
    axs[i].set_xlabel('%s\n(%.3g > %.3g > %.3g)'%(k[i], ci[k[i
        ]][0], pardic[k[i]], ci[k[i]][1]), size=8)
```

```
99
100 axs[0].set_ylabel(r'$\chi^2/\chi^2_0$', size=15)
101 #for i in [0, 7, 14, 21, 28, 35]:
102 #    axs[i].set_ylabel('SSR')
103
104 fig.subplots_adjust(wspace=0.4, hspace=0.4)
105 fig.savefig('plot.pdf', format='pdf')
106 #fig.show()
```

D.3 Standard deviation determination

D.3.1 PGI

```
1  import sys
2  import os
3  import scipy as sp, numpy as np
4  from scipy.integrate import odeint
5  import matplotlib
6  import pylab
7  from multiprocessing import Pool, cpu_count
8  from glob import glob
9  import pickle
10 import datetime
11 import lmfit
12 from time import time
13 from scipy.interpolate import UnivariateSpline
14
15
16 """
17
18 This script estimates the noise in the data by fitting splines
      to the first n
19 data of all datasets and taking the total standard deviation,
      which is saved as
20 'sd.t'. A plot of the fit is saved as 'plot_sd.pdf'.
21
22 """
23 n = 20
24 k = 3
25
26 def f_sim(mod, pardic, inits, t):
27     integration = sp.integrate.odeint(mod.f, inits, t, args=
          tuple([pardic]))
28     return dict(list(zip(mod.species, np.transpose(integration))
          ))
29
30 def fp(pardic, data):
31     result = {}
32     for k,v in data.items():
33         inits = [v[s][0] for s in m.species]
34         #dataset-specific repetition time and total protein
```

```
35          t = np.arange(len(list(v.values())[0]))
36          pardic['et'] = v['et']
37          pardic['g1p'] = v['g1p'][0]
38          result[k] = f_sim(m, pardic, inits, t)
39      return result
40
41  def get_res_from_pars(p):
42      pardic = m.parameters.copy()
43      for i in p:
44          pardic[i] = p[i].value
45      err = res(pardic)
46      return err
47
48  def res(pardic):
49      sim = fp(pardic,data)
50      crit = m.crit
51      err = []
52      for k in data:
53          err += [data[k][c]-sim[k][c] for c in crit]
54      err = [j for i in err for j in i]
55
56      #catch nans and convert them to infs
57      for i in range(len(err)):
58          if np.isnan(err[i]):
59              err[i] == np.inf
60      open('text.h', 'ab').write('%f\n'%sum(np.array(err)**2))
61      return err
62
63  def do_min(d,p_fix):
64      p = lmfit.Parameters()
65      for i in d:
66          p.add(i, value=d[i], min=1e-6)
67      p[p_fix].vary = False
68      mz = lmfit.minimize(get_res_from_pars,p,method=fit_method)
69      return mz.chisqr
70
71  def do_par(pd):
72      begin_time = time()
73      open('text.h','ab').write('%s\n'%pd[0])
74      d = {}
75      for i in m.fitted_params:
76          d[i] = m.parameters[i]
77      result = []
78      for i in pd[1:]:
79          d[pd[0]] = i
80          c2 = do_min(d,pd[0])
81          result.append(c2)
82      end_time = time()
83      run_time = ((end_time-begin_time)/60.0)
84      run_time_total = run_time*len(m.fitted_params)/float(nprocs)
85      open('text.h','ab').write('%s runtime: %.2f min (total: +-
```

```
        %.2f)\n'%(pd[0], run_time, run_time_total))
86      return pd+result
87
88  def do_tuple(pd):
89      open('text.h','ab').write('%s\n'%pd[0])
90      d = {}
91      for i in m.fitted_params:
92          d[i] = m.parameters[i]
93      d[pd[0]] = pd[1]
94      result = do_min(d,pd[0])
95      open('text.h','ab').write('%s completed\n'%(pd[0]))
96      return pd+[result]
97
98  #import model
99  mod_dir = os.getcwd()
100 sys.path.append(mod_dir+'/data')
101 from d_model_pgi import *
102 m = model()
103
104 #import data
105 mod_dir = os.getcwd()
106 dataraw = pickle.load(open('%s/../2_lsq_lims/results/latest.data
        ' % mod_dir,'rb'))
107
108 data = []
109 for spc in m.crit:
110     for i,j in dataraw.items():
111         data.append(j[spc][:n])
112
113 spl = [UnivariateSpline(np.arange(len(i)),i,k=k,s=1e10)(np.
        arange(len(i))) for i in data]
114
115 fig = pylab.figure()
116 ax = fig.add_subplot(111)
117 for i,j in zip(data,spl):
118     ax.plot(i,'ok')
119     ax.plot(j,'-r')
120 fig.savefig('plot_sd.pdf', format='pdf')
121 dd = [l for k in [i-j for i,j in zip(data,spl)] for l in k]
122 fig = pylab.figure()
123 ax = fig.add_subplot(111)
124 ax.plot(dd)
125
126
127 sd = np.std(dd)
128 print('SD = %.2g'%sd)
129 with open('sd.t','w') as f:
130     f.write(str(sd))
131
132 pylab.show()
```

D.3.2 PFK

```
import sys
import os
import scipy as sp, numpy as np
from scipy.integrate import odeint
import matplotlib
#matplotlib.use('Agg')
import pylab
from multiprocessing import Pool, cpu_count
from glob import glob
import pickle
import datetime
import lmfit
from time import time
from scipy.interpolate import UnivariateSpline


"""

This script estimates the noise in the data by fitting splines to the first n
data of all datasets and taking the total standard deviation, which is saved as
'sd.t'. A plot of the fit is saved as 'plot_sd.pdf'.

"""
n = 20
k = 3

def f_sim(mod, pardic, inits, t):
    integration = sp.integrate.odeint(mod.f, inits, t, args=tuple([pardic]))
    return dict(list(zip(mod.species, np.transpose(integration))))

def fp(pardic, data):
    result = {}
    for k,v in data.items():
        inits = [v[s][0] for s in m.species]
        #dataset-specific repetition time and total protein
        t = np.arange(len(list(v.values())[0]))
        pardic['et'] = v['et']
        pardic['g1p'] = v['g1p'][0]
        result[k] = f_sim(m, pardic, inits, t)
    return result

def get_res_from_pars(p):
    pardic = m.parameters.copy()
    for i in p:
        pardic[i] = p[i].value
```

```
46      err = res(pardic)
47      return err
48
49  def res(pardic):
50      sim = fp(pardic,data)
51      crit = m.crit
52      err = []
53      for k in data:
54          err += [data[k][c]-sim[k][c] for c in crit]
55          #err += [(data[k][c]-sim[k][c])/sim[k][c] for c in crit]
56      err = [j for i in err for j in i]
57
58      #catch nans and convert them to infs
59      for i in range(len(err)):
60          if np.isnan(err[i]):
61              err[i] == np.inf
62      open('text.h', 'ab').write('%f\n'%sum(np.array(err)**2))
63      return err #[sum(np.array(err, dtype='f8')**2)]
64
65  def do_min(d,p_fix):
66      p = lmfit.Parameters()
67      for i in d:
68          p.add(i, value=d[i], min=1e-6)
69      p[p_fix].vary = False
70      mz = lmfit.minimize(get_res_from_pars,p,method=fit_method)
71      #d = {i:p[i].value for i in p}
72      return mz.chisqr
73
74  def do_par(pd):
75      begin_time = time()
76      #d = {i:m.parameters[i] for i in m.fitted_params}
77      open('text.h','ab').write('%s\n'%pd[0])
78      d = {}
79      for i in m.fitted_params:
80          d[i] = m.parameters[i]
81      result = []
82      for i in pd[1:]:
83          d[pd[0]] = i
84          c2 = do_min(d,pd[0])
85          result.append(c2)
86      end_time = time()
87      run_time = ((end_time-begin_time)/60.0)
88      run_time_total = run_time*len(m.fitted_params)/float(nprocs)
89      open('text.h','ab').write('%s runtime: %.2f min (total: +-
            %.2f)\n'%(pd[0], run_time, run_time_total))
90      return pd+result
91
92  def do_tuple(pd):
93      open('text.h','ab').write('%s\n'%pd[0])
94      d = {}
95      for i in m.fitted_params:
```

```
96          d[i] = m.parameters[i]
97      d[pd[0]] = pd[1]
98      result = do_min(d,pd[0])
99      open('text.h','ab').write('%s completed\n'%(pd[0]))
100     return pd+[result]
101
102 #import model
103 mod_dir = os.getcwd()
104 sys.path.append(mod_dir+'/data')
105 from d_model_pgi_pfk import *
106 m = model()
107
108 #import data
109 mod_dir = os.getcwd()
110 dataraw = pickle.load(open('%s/../2_lsq_lims/results/latest.data
    ' % mod_dir,'rb'))
111
112 data = []
113 for spc in m.crit:
114     for i,j in dataraw.items():
115         data.append(j[spc][:n])
116
117 spl = [UnivariateSpline(np.arange(len(i)),i,k=k,s=1e10)(np.
    arange(len(i))) for i in data]
118
119 fig = pylab.figure()
120 ax = fig.add_subplot(111)
121 for i,j in zip(data,spl):
122     ax.plot(i,'ok')
123     ax.plot(j,'-r')
124 fig.savefig('plot_sd.pdf', format='pdf')
125 dd = [l for k in [i-j for i,j in zip(data,spl)] for l in k]
126 fig = pylab.figure()
127 ax = fig.add_subplot(111)
128 ax.plot(dd)
129
130
131 sd = np.std(dd)
132 print('SD = %.2g'%sd)
133 with open('sd.t','w') as f:
134     f.write(str(sd))
135
136 pylab.show()
```