Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Marius Fredriksen

Model Predictive Control of bioreactors based on the *Escherichia coli* core metabolic network model

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Rafael David de Oliveira, Caroline Satye Nakama
June 2023

NTNU
Norwegian University of
Science and Technology

Marius Fredriksen

Model Predictive Control of bioreactors based on the *Escherichia coli* core metabolic network model

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Rafael David de Oliveira, Caroline Satye Nakama
June 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

Optimal control of processes is important to ensure efficient and safe plant operations. The growing popularity and use of Dynamic Flux Balance Analysis (dFBA) models have paved the way to implement more advanced control structures for bioreactors, such as Model Predictive Control (MPC). As the dFBA consists of a set of dynamic mass balance equations and an optimization that calculates the cell's metabolic fluxes, a bi-level optimization problem arises when MPC is applied. The bi-level optimization problem can be solved by reformulating the inner optimization to a set of algebraic expressions utilizing the duality theory and Karush–Kuhn–Tucker (KKT) optimality conditions.

In this work, we first proposed reformulations of the dFBA model for batch and continuous stirred tank (CSTR) bioreactors that would make the dFBA feasible for MPC applications. The ordinary differential equation (ODE) system was discretized using the orthogonal collocation approach for finite elements and we implemented an adaptive mesh strategy to place the elements. A penalization method for the optimality conditions was also utilized, as too many hard constraints may lead to convergence error and infeasible problems. We evaluated this methodology in a case study of the *Escherichia coli* core metabolic network, emphasizing the accuracy and efficiency of the different dFBA reformulations. Finally, we applied MPC to our CSTR dFBA models, where we tested the controller's ability to handle changes in the setpoint, and disturbances in the glucose feed concentration and the maximal glucose uptake to the cells. The goal was to compare the KKT and duality theory reformulations of the MPC model regarding computation time and reliability.

We have shown that MPC can be applied to CSTR bioreactors based on the *Escherichia coli* core metabolic network model utilizing dFBA. The MPC controller performed well, keeping the biomass concentration close to the desired setpoint. It was found that the MPC can handle relatively large changes in the setpoint of biomass, disturbances in the glucose feed concentration, and disturbances in the maximal glucose uptake. Overall, the penalized duality theory reformulation was computationally faster than the penalized KKT reformulation and more reliable, returning fewer failed MPC optimizations than the penalized KKT reformulation when solved with the IPOPT solver.

Sammendrag

Optimal kontroll av prosesser er viktig for å sikre effektiv og trygg operasjon av prosessanlegg. Den økende populariteten og bruken av Dynamisk Fluks Balanse Analyse (dFBA) modeller har gjort det mulig å ta i bruk mer avanserte kontrollmetoder, slik som Modell Prediktiv Kontroll (MPC). Ettersom dFBA består av ett sett dynamiske massebalanser og ett optimaliseringsproblem som regner ut de metabolske fluksene i cellene, får vi ett to-nivå optimaliseringsproblem når vi tar i bruk MPC. Det to-nivå optimaliseringsproblemet kan bli løst ved å omformulere det indre optimaliseringsproblemet til ett sett av algebraiske utrykk ved hjelp av dualitets teori og Karush–Kuhn–Tucker (KKT) optimalitetskondisjoner.

I dette arbeidet har vi først foreslått omformuleringer av dFBA modellen for batch og kontinuerlig blande-tank (CSTR) bioreaktorer som kan bli tatt i brukt for MPC applikasjoner. Systemet av ordinære differensialligninger (ODE) ble diskretisert ved bruk av ortogonal kollokasjon og vi tok i bruk en adaptivt mesh-strategi for å plassere kollokasjonspunktene. En straffemetode ble også tatt i bruk for optimalitets kondisjon reformuleringene, ettersom for mange strenge begrensninger i MPC-optimaliseringsproblemet kan føre til konvergensproblemer eller uløselige problemer. Vi evaluerte modellene våre i en studie av *Escherichia coli* bakteriens kjernemetabolske nettverk, hvor vi la vekt på dFBA modellens nøyaktighet og effektivitet. Tilslutt tok vi i bruk MPC for CSTR-modellreformuleringene og testet kontrollerens evne til å håndtere endringer i settpunktet, forstyrrelser i glukosekonsentrasjonen for reaktorføden, og den maksimale glukoseinntaks-parameteren for cellen. Målet var å sammenligne dualitets teori og KKT reformuleringene av MPC-modellen, med fokus på reformuleringenes beregningstid og pålitelighet.

Vi har vist at MPC kan bli tatt i bruk for CSTR bioreaktorer som bruker *Escherichia coli* bakteriens kjerne metabolske nettverksmodell basert på dFBA. MPC-kontrolleren presterte bra, og holdt biomassekonsentrasjonen nært det ønskede settpunktet. Vi fant at MPC-kontrolleren kan håndterer relativt store endringer i settpunktet for biomasse, samt store forstyrrelser i glukosekonsentrasjonen for reaktorføden og den maksimale glukoseinntaksparameteren. Alt i alt, var den straffede dualitets teori reformuleringen raskere og mer pålitelig enn den straffede KKT-reformuleringen, med færre ikke vellykkede MPC optimaliseringer enn den straffede KKT-reformuleringen når systemet ble løst med løseren IPOPT.

Preface

This thesis was written as the final part of my M.Sc. in Chemical engineering at the Norwegian University of Science and Technology (NTNU) in the spring of 2023.

I would like to thank my supervisor Johannes Jäschke for the guidance and the opportunity to work on this topic during my master project. A big thank you to my co-supervisor Rafael David de Oliveira for the much appreciated help and support. In addition, I would like to thank my co-supervisor from my specialization project, Caroline Satye Nakama, for all help, inspiration, and for introducing me to the Julia programming language. Finally, I would like to thank my friends and family for their support, motivation and interesting discussions.

Declaration of Compliance

I, Marius Fredriksen, hereby declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Signature:

Place and Date: Trondheim - 19.06.2023

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

CS:	Complementary slackness
CSTR:	Continuous stirred tank reactor
DA:	Direct Approach
dFBA:	Dynamic Flux Balance Analysis
Dual:	Duality theory reformulation
DW:	Dry weight
FBA:	Flux Balance Analysis
FRD7:	Fumarate reductase reaction
fum:	Fumarate
HT-NGS:	High-throughput next-generation sequencing
IPD:	Infeasible problem detected
KKT:	Karush–Kuhn–Tucker/KKT reformulation
MIE:	Maximum number of iterations exceeded
MMK:	Michaelis-Menten kinetics
MPC:	Model Predictive Control
MSE:	Mean squared error
MV:	Manipulated variable
NLP:	Non-linear problem
NLPA:	Non-Linear Programming Approach
ODE:	Ordinary differential equation
PID:	Proportional integral derivative controller
pFBA:	Parsimonious Flux Balance Analysis
P. Dual:	Penalized duality theory reformulation
P. KKT:	Penalized KKT reformulation
RF:	Restoration failed
STAL:	Solved to acceptable level
SS:	Solved successfully
SUCDi:	Succinate dehydrogenase reaction
succ:	Succinate

Symbols

Symbol	Definition	Unit
A	Acetate concentration	mmol/L
A_3	Acetate concentration in third collocation point	mmol/L
$\mathbf{c}$	Objective vector for FBA	-
c_j	Slack variable at point j	-
C	MPC control parameter	-
d	Dual solution	-
D	Dilution rate	h^{-1}
$E(k+1)$	MPC error function	-
$f_i(x)$	General inequality function i	-
$F(\mathbf{x}, \mathbf{v})$	Right hand side of mass balances	mmol/h
F_{in}	Input flow rate	L/h
F_{out}	Output flow rate	L/h
$g(\mathbf{x})$	Kinetic expression for flux uptake	mmol/gDW h
$g(\lambda_i, \mu_i)$	Lagrangian dual function	-
G	Glucose concentration	mmol/L
G_3	Glucose concentration in third collocation point	mmol/L
h	Size of finite elements	h
$h_i(x)$	General equality function i	-
h^*	Equidistant element size	h
$\mathbf{LB}$	Metabolic reactions lower bounds	mmol/gDW h
$K_{M,i}$	Michaelis constant of component i	mmol/L
M	Orthogonal collocation M-matrix	-
p	Primal solution	-
Q	MPC control parameter	-
$r/\mathbf{r}$	Reaction rate	mmol/L h
r_{max}	Maximal reaction rate	mmol/L h
R	MPC control parameter	-
$\mathbf{S}$	Stoichiometric matrix	-
t	Time	h
u	Manipulated variable	-
$\mathbf{UB}$	Metabolic fluxes upper bounds	mmol/gDW h
$\mathbf{v}$	Metabolic fluxes	mmol/gDW h
$\mathbf{v}_{max}$	Maximal metabolic flux uptake	mmol/gDW h
$\mathbf{v}_{uptake}$	Metabolic fluxes for the cellular uptake	mmol/gDW h
V	Reactor volume	L
$\mathbf{W}$	Parsimonious FBA wight matrix	-
$\mathbf{x}$	States/Extracellular metabolite concentrations	mmol/L
$x_{biomass}$	Biomass concentration	gDW/L
x_i	Concentration of component i, except biomass	mmol/L
$\mathbf{x}_{in}$	Concentration of extracellular metabolites in the reactor feed	mmol/L
X	Biomass concentration	gDW/L
y	Simulation results	-
$\bar{y}$	Predicted results	-
λ_i	Lagrangian equality multipliers	-
μ_i	Lagrangian inequality multipliers	-

1 Introduction

1.1 Motivation

Bioprocessing is an important field of research as it is an essential part of many food, chemical and pharmaceutical industries [5]. Bioprocessing refers to the use of microbial, animal, and plant cells, and their components, such as enzymes, to manufacture goods and destroy harmful waste. Premodern biomanufacturing has its origins in antiquity, where it was used to create food products like bread, beer, cheese and wine. Since then the field of bioprocessing have expanded to encompass the manufacturing of a large range of commercial products. Everything from relatively cheap materials like industrial alcohol and organic solvents to expensive products like antibiotics, vaccines and therapeutic proteins. Everyday household products like citric acid and baker's yeast are also products of bioprocessing [5].

Optimal control of processes is important to ensure efficient and safe process plant operations. The topic of process control has thus become increasingly important in recent years, as stronger competition, tougher environmental and safety regulations, and rapidly changing economic conditions have made the performance requirements of process plants more difficult to satisfy [26]. Currently, control structures of bioprocesses are recipe-based with insufficient ability to handle possible uncertainties [10].

One of the main challenges of implementing more advanced control structures, such as Model Predictive Control (MPC), to bioprocesses is the lack of good mathematical models. Therefore is one of the most ambitious goals of systems biology to develop good mathematical models of biological systems. [3, 26]. The models currently used in control are unstructured and based on a very simplistic representation of the cellular metabolism [10]. This results in the models only being valid for a very limited range of process conditions and limit their ability to predict cellular behavior in response to changing environmental conditions [10].

In recent years improvements in genome sequencing have allowed for the reconstruction of genome-scale metabolic networks for multiple organisms. High-throughput next-generation sequencing (HT-NGS) technologies can produce over 100 times more data than the sequencers based on the Sanger method, introduced in 1977. Many models have been built based on these networks, for example the Flux Balance Analysis (FBA) model [19, 21].

Dynamic Flux Balance Analysis (dFBA) is a widely applied variant of the FBA model approach, growing in popularity because it can account for wide ranges of cellular behavior and operation conditions [18, 16]. The dFBA has therefore paved the way to implement more advanced control structures for bioprocesses, such as the MPC. The dFBA model consists of a set of dynamic mass balance equations and an optimization that calculates the cell's metabolic fluxes through the metabolic network [15]. Thus a bi-level optimization problem arises when MPC is applied, which can be solved by reformulating the inner optimization to a set of algebraic expressions utilizing the duality theory and Karush–Kuhn–Tucker (KKT) optimality conditions. We want to avoid bi-level optimization as they often are very time-consuming to solve. This is a result of the fact that we must analyse multiple upper-level optimization candidates at the same time as we have to analyse the corresponding lower-level optimization candidates [7]. dFBA models based on numerical solution approaches, such as those developed by Nakama and Jäschke [16], and Oliveira et al. [18] have shown that model-based control of bioprocesses employing dFBA

can be successfully formulated and solved with line search interior point solvers using the KKT reformulations. However, none of these works explore the utilization of the duality theory reformulations of the dFBA or MPC models.

1.2 Project goals

This project aims to apply MPC to a bioreactor using the dFBA model of the *Escherichia coli* core metabolism. The goal is to compare the KKT and duality theory reformulations of the dFBA model regarding computation time and reliability.

1.3 Scope of work

The paper by Oliveira et al. [18] is used as a basis to develop a reformulation of the dFBA model for batch and continuous stirred tank (CSTR) bioreactors that would make the dFBA feasible to MPC applications. The first step is to reformulate the inner optimization problem of the dFBA model with duality theory and KKT optimality conditions. The second step is to introduce the mass balances for the batch and CSTR bioreactors. The third step is to introduce a penalization method for the optimality conditions, as too many hard constraints may lead to convergence error and infeasible problems. The fourth step is to solve the dFBA with the direct approach (DA) and the non-linear programming approach (NLPA). In the DA we apply an ordinary differential equation (ODE) solver that calls a non-linear problem (NLP) optimizer at each step. In the NLPA the set of dynamic mass balances is discretized with the orthogonal collocation method with finite elements. The fifth step is to introduce an adaptive mesh strategy to place the finite elements, in order to increase the accuracy of the dFBA model reformulations. Finally, we applied MPC to our CSTR dFBA models and tested the controller's ability to handle changes in the setpoint, disturbances in the glucose feed concentration, and disturbances the maximal glucose uptake of the cells.

Relevant theory is presented, followed by a description of the dFBA model reformulations, key components, and implementation. Thereafter we present a description of the MPC applied to the dFBA reformulations for the CSTR. A case study of the *Escherichia coli* core metabolism is conducted to test the model reformulations and the MPC with emphasis on computational time, accuracy and reliability. Finally, a short conclusion and some suggestions for future improvements of the dFBA model reformulations are given. The julia code used in the project is presented in Appendix B.

2 Theoretical background

In this chapter relevant theory for the thesis is presented. First, the basic concepts of Flux Balance Analysis (FBA) and some of its variants are explained. Thereafter, mass balances for batch and continuous stirred tank bioreactors (CSTR) are presented, shortly followed by the expressions used for the uptake of substrates to the cells. These expressions are necessary to solve our Dynamic FBA (dFBA) model. Thereafter, the basics of some mathematical methods used in this thesis are presented. This includes an introduction to duality theory and Karush–Kuhn–Tucker (KKT) optimality conditions, and the concepts of orthogonal collocation. These methods are needed to rearrange the optimization problem of the FBA to a set of algebraic equations and to discretize the set of differential equations in the dFBA. Lastly, the basics of Model Predictive Control (MPC) are presented, as we want to apply this type of controller to our bioreactor models.

2.1 Flux balance analysis

Flux balance analysis is an important tool for studying metabolic networks as it does not require kinetic information about individual reactions or the concentrations of intermediate metabolites, thus allowing us to deal with our current difficulties with measuring metabolite levels and kinetics *in vivo* conditions [19]. The FBA calculates the metabolic fluxes through metabolic networks, which are experimentally determined from the genome of the organism and contain information about all the known reactions in the cells. The FBA may be used to approximate the growth rate, or the rate of production or consummation of specific metabolites [19, 5].

The first step in FBA is to present the metabolic reactions mathematically. The metabolic network of the organism is used to create a stoichiometric matrix ($\mathbf{S}$) that contains the stoichiometric coefficients of each reaction. Every row in the matrix represents a metabolite, and every column represents a metabolic reaction. Some reactions, like the reaction of biomass, are added by introducing an artificial "biomass-reaction", a reaction that consumes precursor metabolites to create biomass. The stoichiometries impose constraints on the flow of metabolites through the network. These constraints differentiate the FBA from other approaches that require hard-to-estimate reaction kinetic parameters [19].

The second step is to add lower ($\mathbf{LB}$) and upper ($\mathbf{UB}$) bounds of the reactions. These bounds are used to define the maximum and minimum allowed fluxes through the metabolic network. The lower bounds may be used to force a reaction to be irreversible by setting the bound larger or equal to zero.

The third step is to define an objective. Often the objective is to maximise the growth of biomass. The objective is usually given on the form $\mathbf{c}^T\mathbf{v}$, where $\mathbf{v}$ is a vector containing all the metabolic fluxes and $\mathbf{c}$ is a vector containing weights that indicate how much each reaction contribute to the objective.

The fourth step is to assume steady state. At steady state is the flux trough each reaction given by $\mathbf{Sv} = 0$.

The final step is to solve the FBA as an optimization problem. This is necessary as large networks usually have more reactions than metabolites, and therefore multiple solutions of the FBA usually exist, resulting in an optimization problem.

$$\min_{\mathbf{v}} -\mathbf{c}^T\mathbf{v} \tag{1a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{1b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{1c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{1d}$$

Like all models the FBA creates predictions that have to be verified by experimental data. Studies, such as those conducted by Edwards et al. [6], have shown that the FBA works well for predicting the growth rate of *Escherichia coli* on several different substrates [19].

2.1.1 Parsimonious FBA

The FBA usually has multiple optimal solutions [19]. One reason for this is that the metabolic networks may contain loops. For example, the loop present between the metabolites fumarate (fum) and succinate (succ) in the citric acid cycle in the *Escherichia coli* core metabolic network, see Figure 1 [20].

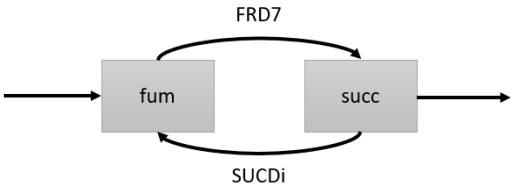


Figure 1: A loop in the core metabolic network of *Escherichia coli* is present between the metabolites fumarate (fum) and succinate (succ) in the citric acid cycle. SUCDi is the succinate dehydrogenase reaction and FRD7 is the fumarate reductase reaction [20].

There are multiple ways to limit the number of possible solutions for the FBA. A common solution is to minimize the total absolute value of all the fluxes in the metabolic network by adding a penalization term to the objective expression. This approach is called parsimonious FBA (pFBA) [14, 12]. A variation of the pFBA given by Ploch et al.[22] is presented below.

$$\min_{\mathbf{v}} -\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v} \tag{2a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{2b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{2c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{2d}$$

Where $\mathbf{W}$ is a diagonal matrix containing very small wights.

2.1.2 Dynamic Flux Balance Analysis

A drawback of the FBA is that it does not give any information about the concentration of the metabolites. A commonly applied variation of the FBA that allows us to track

the concentration of the extracellular metabolites is the dFBA. The main assumption of dFBA is that the intracellular reactions are much faster than the extracellular ones, thus we may consider the intracellular metabolites to be at a steady state [19]. The dFBA is thus implemented by adding mass balances for the extracellular metabolites and kinetics for the uptake of substrates into the cells. The dFBA model expression used in this project is presented below [18].

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, \mathbf{v}) \tag{3a}$$

$$\mathbf{v}_{uptake} = g(\mathbf{x}) \tag{3b}$$

$$\mathbf{x} \geq 0 \tag{3c}$$

$$\mathbf{v} = FBA(\mathbf{v}_{uptake}) \tag{3d}$$

Where $\mathbf{x}$ is a vector containing the concentration of the extracellular metabolites, $F(\mathbf{x}, \mathbf{v})$ is the right-hand side of the mass balances, $\mathbf{v}_{uptake}$ is a vector with the uptake of substrates to the cells and $g(\mathbf{x})$ is the kinetic expression used to calculate the substrate uptake.

To solve the dFBA we will be using the Direct Approach (DA) and an approach that we will refer to as the Non-Linear Programming Approach (NLPA). In the NLPA the ordinary differential equation (ODE) system is discretized and the FBA is solved for all the time steps in one optimization problem. In the DA an ODE solver that calls an optimizer that solves the FBA at each time step is applied. Many efficient ODE solvers are available, and the DA can therefore be solved relatively fast with modern ODE solvers [16]. However, the interactions between the ODE solvers and optimization problems may lead to errors. For example, if the optimization solver does not return a valid solution at an integration step [18]. Applying MPC directly to the DA would also lead to a bi-level optimization problem, which we are trying to avoid.

2.2 Bioreactors

To solve the dFBA the expressions for the mass balances of the external metabolites and the uptake of substrate into the cells are needed. The mass balances are dependent on the type of bioreactor used in the process, and we will consider batch and CSTR bioreactors in this project. The substrate uptake to the cells is represented by the Michaelis-Menten kinetics (MMK).

2.2.1 Batch bioprocesses

Most commercial bioreactors are operated in batch and fed-batch processes. Batch processes are operated as a closed system, where substrates are added at the beginning of the process and products are removed only at the end. As aerobic processes allow air to pass through the system they are not batch operations in the strictest sense. However, bioreactors with neither input or output of liquid and/or solids are classified as batch reactors. Fed-batch processes are similar to batch processes, but they have intermittent or continuous feeding of substrates during the process [5]. As we are focusing on batch processes, are we not going into more detail on fed-batch processes in this thesis.

Mass balances are given in the general form presented below.

$$\frac{d\mathbf{x}}{dt} = F_{in}\mathbf{x}_{in} - F_{out}\mathbf{x} + V\mathbf{r} \tag{4}$$

Where F_{in} and F_{out} are the flow rate in and out of the system respectively, $\mathbf{x}$ is a vector containing the concentration of the extracellular metabolites in the reactor, $\mathbf{x}_{in}$ is a vector containing the concentration of the metabolites in the inlet flow, V is the volume of the reactor, and $\mathbf{r}$ is a vector containing the reaction rate of the metabolites.

The reaction rate is given by the expression below [18].

$$\mathbf{r} = \mathbf{v}x_{biomass} \tag{5}$$

Where $\mathbf{v}$ is a vector containing the flux of the metabolites calculated by the FBA and $x_{biomass}$ is the concentration of biomass.

Introducing Equation 5 to Equation 4, and as the batch system is closed, $F_{in} = F_{out} = 0$, we obtain the final mass balance for batch reactors.

$$\frac{d\mathbf{x}}{dt} = V \cdot (\mathbf{v}x_{biomass}) \tag{6}$$

2.2.2 Continuous bioprocesses

In a few bioprocesses, such as brewing, production of bakers' yeast, and waste treatment, bioreactors are operated continuously [5]. This implies that the system is provided with new substrates and that products are removed during the operation. This is bad for some bioprocesses, for example bioreactors with free enzymes, as the catalyst are continuously withdrawn from the system. However, this is less of a problem for bioprocesses using cell cultures, as growth of new cells replace those who get removed from the system. It is also possible to use a recycle stream or perfusion mode to recover some of the lost cells and enzymes [5]. To obtain an expression for the mass balance of continually operated bioprocesses we are considering a CSTR.

First, we introduce the dilution rate (D) given by $D = \frac{F_{in}}{V} = \frac{F_{out}}{V}$, to the general mass balance, Equation 4.

$$\frac{d\mathbf{x}}{dt} = D \cdot (\mathbf{x}_{in} - \mathbf{x}) + V\mathbf{r} \tag{7}$$

Introducing the reaction rates given by Equation 5 results in the final expression for the mass balance for CSTR bioreactors.

$$\frac{d\mathbf{x}}{dt} = D \cdot (\mathbf{x}_{in} - \mathbf{x}) + x_{biomass}\mathbf{v} \tag{8}$$

2.2.3 Substrate uptake

The expressions of the kinetics of the substrate uptake into the cells are necessary to solve the dFBA, as the substrates have to enter the cells before they can take part in the

reactions. We use Michaelis-Menten kinetics (MMK) to express the uptake of substrates into the cells. MMK dates back to 1913 and is used to describe enzymatic reactions where the reaction rate increases in a hyperbolic fashion with substrate concentration [4].

$$r = \frac{r_{max} x_S}{K_M + x_S} \tag{9}$$

Where r is the reaction rate, r_{max} is the maximal reaction rate, x_S is the substrate concentration and K_M is the Michaelis constant.

To obtain the uptake kinetics we replace the reaction rate with the uptake of the substrates to the cells ($\mathbf{v}_{uptake}$), resulting in the following expression [18].

$$\mathbf{v}_{uptake} = \mathbf{v}_{max} \frac{x_S}{K_M + x_S} \tag{10}$$

Where $\mathbf{v}_{max}$ is the maximal uptake of the substrate.

2.3 Optimality conditions

Optimality theory is often used to create a certificate of optimality or stopping criteria for optimization algorithms [27]. We will use optimality conditions to convert an optimization problem, our FBA, into a system of algebraic equations. Both duality theory [30] and Karush–Kuhn–Tucker (KKT) optimality conditions [27] are applied to rearrange our FBA optimization problem.

In this subsection we will consider the generalized optimization problem given in standard form presented below.

$$\min_x \ f_0(x) \tag{11a}$$
$$\text{s.t. } f_i(x) \leq 0 \tag{11b}$$
$$h_i(x) = 0 \tag{11c}$$

Where $x \in \mathbb{R}^n$. Equation 11 will be referred to as the primal problem and we introduce its Lagrangian dual function.

$$g(\lambda_i, \mu_i) = inf_x(L(x, \lambda_i, \mu_i)) \tag{12a}$$
$$= inf_x(f_0(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{i=1}^{p} \mu_i f_i(x)) \tag{12b}$$

The vector μ_i is referred to as the Lagrangian multiplier associated with the ith inequality constraint and the vector λ_i is similarly referred to as the Lagrangian multiplier of the ith equality constraint. Together they are referred to as the dual variables [27].

For any λ_i and $\mu_i \geq 0$ the Lagrangian dual function gives a lower bound of the optimal solution of the primal problem [27]. As we want to find the highest lower bound possible

to solve the primal problem the Lagrangian dual function gives rise to the maximization problem presented below.

$$\max_{\lambda_i,\mu_i} g(\lambda_i, \mu_i) \tag{13a}$$

$$\text{s.t. } \mu_i \geq 0 \tag{13b}$$

This problem is called the Lagrangian dual problem associated with the primal problem and will be referred to as the dual problem [27].

We use * to indicate that a variable is optimized. As x^* minimizes $L(x, \lambda_i^*, \mu_i^*)$ over x, its gradient must vanish at x^*. This adds an additional constraint to the dual problem [30], resulting in the second dual constraint given below.

$$\nabla L(x^*, \lambda_i^*, \mu_i^*) = 0 \tag{14}$$

Finally, before presenting the KKT conditions and duality theory we have to introduce the weak and strong duality theorems.

The weak duality theorem establishes bounding relations between feasible primal solutions (p^*) and feasible dual solutions (d^*).

$$d^* \leq p^* \tag{15}$$

The strong duality theorem states that if either the primal or dual problem has finite optimal values, so does the other one and the optimal objective values are equal to one another [27, 30].

$$d^* = p^* \tag{16}$$

Note that strong duality does not hold in general. However, if the primal problem is convex with f_0 and f_i convex, strong duality usually holds. One simple way to check if strong duality holds is Slater's conditions, which states that strong duality holds if the problem is convex and there exists a $x \in relint(D)$ such that

$$f_i(x) < 0, \ i = 1, .., m, \tag{17a}$$

$$h_i(x) = 0. \tag{17b}$$

Where $relint(D)$ refers to the relative interior of set D given by $\cap_{i=0}^{m} dom(f_i) \cap \cap_{i=1}^{p} dom(h_i)$. $dom(f_i)$ is the domain of function f_i, and $dom(h_i)$ is the domain of function h_i [27, 30].

2.3.1 Duality theory optimality conditions

According to the weak and strong duality theorem, if the primal and dual solutions are feasible and their objective values are equal, then the primal and dual solutions are optimal.

Therefore, one can solve the primal and dual problem simultaneously by collecting all the primal and dual constraints, and setting the primal and dual objectives equal to one another. The resulting set of constraints of the optimality conditions of the primal problem presented in Equation 11 is given below [30].

$$f_i(x^*) \leq 0 \tag{18a}$$
$$h_i(x^*) = 0 \tag{18b}$$
$$\nabla L(x^*, \lambda_i^*, \mu_i^*) = 0 \tag{18c}$$
$$\mu_i^* \geq 0 \tag{18d}$$
$$f(x^*) = g(\lambda_i^*, \mu_i^*) \tag{18e}$$

2.3.2 Karush–Kuhn–Tucker optimality conditions

For any optimization problem with a differentiable objective and constraint functions for which strong duality holds, any pair of primal and dual optimal solutions must satisfy the KKT conditions. The KKT conditions is composed of three parts: the primal feasibility, the dual feasibility and the complementary slackness (CS) constraints [30].

The CS constraints are a product of strong duality.

$$f_0(x^*) = g(\lambda_i^*, \mu_i^*) \tag{19a}$$
$$= inf_x(f_0(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{i=1}^{p} \mu_i f_i(x)) \tag{19b}$$
$$\leq f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* h_i(x^*) + \sum_{i=1}^{p} \mu_i^* f_i(x^*) \tag{19c}$$

As the second CS is composed of equality constraints, it may be neglected as it is already zero. Resulting in the CS conditions given below.

$$\sum_{i=1}^{p} \mu_i^* f_i(x^*) = 0 \tag{20}$$

The KKT conditions are presented below for the primal problem presented in Equation 11 [30, 27].

$$f_i(x^*) \leq 0 \tag{21a}$$
$$h_i(x^*) = 0 \tag{21b}$$
$$\nabla L(x^*, \lambda_i^*, \mu_i^*) = 0 \tag{21c}$$
$$\mu_i^* \geq 0 \tag{21d}$$
$$\sum_{i=1}^{p} \mu_i^* f_i(x^*) = 0 \tag{21e}$$

The KKT approach are quite similar to the duality theory approach, and the main difference is their implementation of the strong duality theorem, as the KKT conditions use CS constraints instead of setting the primal and dual objectives equal to one another. Whether one chooses to use the duality theory or KKT reformulations is often dependent on whether one of the reformulations contain nonlinear or nonconvex terms [30]. However, the use of CS has a drawback in the form of non-linear problem (NLP) solvers not being able to handle them [29], as models utilizing CS constraints are locally dependent at all feasible points, resulting in a non-smooth system. The KKT problem is therefore usually reformulated into an NLP [16]. This can be done by relaxing the CS by replacing Equation 20 with,

$$\sum_{i=1}^{p} \mu_i^* f_i(x^*) = \epsilon, \tag{22}$$

where ϵ is a small constant, or by introducing a penalization term [13]. Where we convert the problem back to an optimization problem, and move the CS from the set of constraints to objective function.

2.4 Orthogonal collocation on finite elements

To avoid bi-level optimization problems when we solve the dFBA with the NLPA, we need to discretize the set of ordinary differential equations (ODE). It was decided to use orthogonal collocation as orthogonal collocation is found equivalent to implicit Runge-Kutta methods with high-order accuracy and strong stability properties [18].

The objective in orthogonal collocation is to determine a matrix (M) that relates the derivatives to the non-derivative values over a horizon consisting of points 1 to n. In Equation 23 the relationship is presented with three points and an initial point [8].

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = M \cdot \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} \right) \tag{23}$$

Where x_0 is the initial value, x_i is the x-value in point i and x_i' is the value of the derivative of x in point i.

The derivatives at each point are approximated by Lagrange interpolating polynomials as shown below.

$$x(t) = A + Bt + Ct^2 + Dt^3 \tag{24a}$$
$$x'(t) = B + 2Ct + 3Dt^2 \tag{24b}$$

Where A, B, C and D are vectors with coefficients and t is a vector containing the time at each point. For initial value problems the coefficients of A are equal to x_0.

Substituting Equation 24 into Equation 23 gives the following expression.

$$\begin{bmatrix} B + 2Ct_1 + 3Dt_1^2 \\ B + 2Ct_2 + 3Dt_2^2 \\ B + 2Ct_3 + 3Dt_3^2 \end{bmatrix} = M \cdot \left(\begin{bmatrix} A + Bt_1 + Ct_1^2 + Dt_1^3 \\ A + Bt_2 + Ct_2^2 + Dt_2^3 \\ A + Bt_3 + Ct_3^2 + Dt_3^3 \end{bmatrix} - \begin{bmatrix} A \\ A \\ A \end{bmatrix} \right) \tag{25a}$$

$$\begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} = M \begin{bmatrix} t_1 + t_1^2 + t_1^3 \\ t_2 + t_2^2 + t_2^3 \\ t_3 + t_3^2 + t_3^3 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} \tag{25b}$$

Rearranging and solving for M gives us an expression for the M-matrix.

$$M = \begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix} \begin{bmatrix} t_1 + t_1^2 + t_1^3 \\ t_2 + t_2^2 + t_2^3 \\ t_3 + t_3^2 + t_3^3 \end{bmatrix}^{-1} \tag{26}$$

We get an expression for the x-value at each point by rearranging Equation 23 and introducing a factor to accommodate for the length of the finite elements (h) [8].

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} + h \cdot \left(M \begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} \right) \tag{27}$$

2.5 Model Predictive Control

We want to test the behavior of our model by applying MPC to the dFBA implemented for a CSTR. MPC is an important advanced control technique for difficult multi-variable control problems. Assume that we have a multiple input and output process with inequality constraints on the input and output variables. If we have a reasonably accurate model we can use the model and the current measurements to predict the future value of the outputs. Then the appropriate changes to the input variables can be calculated [26]. The overall objectives of an MPC controller have been summarized by Qin and Badgwell [23].

- Prevent violations of input and output constraints.
- Drive the output variables to their steady state optimal values, while maintaining other outputs within specific ranges.
- Drive the input variables to their steady state optimal values using the remaining degrees of freedom.
- Prevent excessive movement of the input variables.
- Control as many process variables as possible when signals and actuators fail.

The general objective of the MPC is to find the control moves for the next M number of time steps, however only the first one is implemented. Then a new set of control moves are calculated. The important advantage of this approach is that new information is implemented instantly instead of waiting until we calculate the next set of control moves. The control calculations of an MPC are based on minimizing the predicted deviation from

the reference trajectory ($E(k+1)$) and the changes to the manipulated variables (MV) ($\Delta u(k)$). k is the current sampling instant. The MPC control calculations are thus usually presented on the general form given below [26].

$$\min_{\Delta u(k)} E(k+1)^T Q E(k+1) + \Delta u(k)^T R \Delta u(k) \tag{28a}$$

$$s.t. \quad u_{min}(k) \leq u(k+j) \leq u_{max}(k) \tag{28b}$$

$$\Delta u_{min}(k) \leq \Delta u(k+j) \leq \Delta u_{max}(k) \tag{28c}$$

$$x_{min}(k+j) - c_j \leq x(k+j) \leq x_{max}(k+j) + c_j \tag{28d}$$

$$x(k+1) = F(x(k), u(k)) \tag{28e}$$

Where u, u_{min}, u_{max}, Δu_{min}, and Δu_{max} are the value, the minimal value, the maximal value, the minimal change, and the maximal change of the MV respectively. c_j is the slack variables, x, x_{min}, and x_{max} are the value, the minimal value, an the maximal value of the output, $F(x(k), u(k))$ is a function used to calculate the output, Q and R are penalty matrices, and j is given by $0, 1, 2, ..., M-1$. Both Q and R are usually diagonal matrices with positive elements [26].

3 Problem statement

This chapter presents the key components of our Dynamic Flux Balance Analysis (dFBA) models and Model Predictive Controller (MPC). First the implementation of the dFBA is given. Where the parsimonious Flux Balance Analysis (pFBA) optimization problem has been reformulated with the use of duality theory and Karush–Kuhn–Tucker (KKT) optimality conditions. Thereafter we show how we solve the dFBA with the Direct Approach (DA) and Non-Linear Programming Approach (NLPA). We then present the mass balances used for the batch and the continuous stirred tank reactor (CSTR), thus completing our dFBA models. Finally, we introduce the MPC applied to our dFBA models for CSTR. The models are implemented in the julia programming language, and the code is provided in Appendix B.

3.1 FBA reformulation

We have decided to use the pFBA implementation presented in Equation 29 as this version of Flux Balance Analysis (FBA) returns a single optimal solution and does not require multiple optimization operations to be solved.

$$\min_{\mathbf{v}} -\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v} \tag{29a}$$
$$\text{s.t. } \mathbf{S}\mathbf{v} = 0 \tag{29b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{29c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{29d}$$

$\mathbf{v}$ is a vector containing the metabolic fluxes, $\mathbf{S}$ is the stoichiometric matrix, $\mathbf{c}$ is a vector that combined with the fluxes gives the objective function, $\mathbf{W}$ is a diagonal penalty matrix, $\mathbf{LB}$ is the lower bounds and the $\mathbf{UB}$ is the upper bounds.

As the goal is to implement MPC, it is necessary to impose the pFBA optimality conditions through algebraic equations to avoid bi-level optimization problems. We consider two approaches to rearrange the pFBA, the duality theory approach, Equation 18, and the KKT approach, Equation 21. First, we find the Lagrangian of the pFBA.

$$L(\mathbf{v}, \lambda, \mu_U, \mu_L) = -\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v} + \lambda^T\mathbf{S}\mathbf{v} + \mu_L^T(\mathbf{LB} - \mathbf{v}) + \mu_U^T(\mathbf{v} - \mathbf{UB}) \tag{30a}$$
$$= (-\mathbf{c} + \mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U)^T\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB} \tag{30b}$$
$$= (-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U)^T\mathbf{v} - \mathbf{v}^T\mathbf{W}\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB} \tag{30c}$$

Where the Lagrangian multipliers are vectors given by λ, μ_U and μ_L.

Since $L(\mathbf{v}, \lambda, \mu_U, \mu_L)$ is a convex quadratic function of $\mathbf{v}$ the Lagrangian dual function, $inf_{\mathbf{v}}L(\mathbf{v}, \lambda, \mu_U, \mu_L)$, is found from the condition [27],

$$\nabla_{\mathbf{v}}L(\mathbf{v}, \lambda, \mu_U, \mu_L) = -\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0. \tag{31}$$

The resulting Lagrangian dual function, $g(\lambda, \mu_U, \mu_L)$, is given below.

$$g(\lambda, \mu_U, \mu_L) = inf_{\mathbf{v}}(L(\lambda, \mu_U, \mu_L)) \tag{32a}$$
$$= inf_{\mathbf{v}}((-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U)^T\mathbf{v} - \mathbf{v}^T\mathbf{W}\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB}) \tag{32b}$$
$$= -\mathbf{v}^T\mathbf{W}\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB} \tag{32c}$$

The duality theory optimality conditions are found by combining the primal feasibility, dual feasibility and strong duality conditions as in Equation 18. We solve the system as an optimization problem with a dummy objective function. Resulting in the dual theory optimality conditions presented below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L} \quad 1 \tag{33a}$$
$$\text{s.t. } \mathbf{S}\mathbf{v} = 0 \tag{33b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{33c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{33d}$$
$$\mu_U, \mu_L \geq 0 \tag{33e}$$
$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{33f}$$
$$-\mathbf{v}^T\mathbf{W}\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB} = -\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v} \tag{33g}$$

Similarly, we find the KKT optimality conditions by replacing the strong duality condition with the complimentary slackness conditions as in Equation 21. Resulting in the KKT optimality conditions given below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L} \quad 1 \tag{34a}$$
$$\text{s.t. } \mathbf{S}\mathbf{v} = 0 \tag{34b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{34c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{34d}$$
$$\mu_U, \mu_L \geq 0 \tag{34e}$$
$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{34f}$$
$$\mu_L^T(\mathbf{LB} - \mathbf{v}) = 0 \tag{34g}$$
$$\mu_U^T(\mathbf{v} - \mathbf{UB}) = 0 \tag{34h}$$

To avoid the potential drawbacks of implementing the complementary slackness (CS) constraints, as presented in Section 2.3, we utilize the penalization method [18]. We choose this method because it can be solved in a single optimization problem and Baumrucker et al. [1] have shown that non-linear problems (NLP) based on penalisation methods are effective strategies to solve CS problems. We also decided to apply the penalization method to the duality theory reformulation of the pFBA to compare the performance of the penalized and non-penalized reformulations and to remove one of the hard constraints, as too many hard constraints in MPC problems can lead to infeasible solutions [26]. The strong duality and CS conditions are thus moved to the objective function as a penalization term.

The penalized dual theory approach is given by the expression below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L} \quad 2\mathbf{v}^T\mathbf{W}\mathbf{v} - \mu_L^T\mathbf{LB} + \mu_U^T\mathbf{UB} - \mathbf{c}^T\mathbf{v} \tag{35a}$$

$$\text{s.t.} \quad \mathbf{S}\mathbf{v} = 0 \tag{35b}$$

$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{35c}$$

$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{35d}$$

$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{35e}$$

$$\mu_U, \mu_L \geq 0 \tag{35f}$$

The penalized KKT approach is given below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L} \quad \mu_U^T(\mathbf{UB} - \mathbf{v}) + \mu_L^T(\mathbf{v} - \mathbf{LB}) \tag{36a}$$

$$\text{s.t.} \quad \mathbf{S}\mathbf{v} = 0 \tag{36b}$$

$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{36c}$$

$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{36d}$$

$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{36e}$$

$$\mu_U, \mu_L \geq 0 \tag{36f}$$

Note that when we introduce the dummy objective or the penalization method to the reformulations of the pFBA, the reformulations revert back to optimization problems. However, this is not an issue as the reformulations can be expanded to account for the MPC without ending up with bi-level optimization problems.

3.2 Implementation of dFBA

We apply the pFBA model, see Equation 29, and the Michaelis-Menten kinetics (MMK) expressions for the consumption of the substrates, see Equation 10, to the dFBA formulation by Oliveira et al. [18], see Equation 3. The extracellular metabolites considered in the dFBA are glucose (G), biomass (X), and acetate (A). Glucose is the primary substrate, the biomass is the desired product and the acetate switches between byproduct and substrate dependent on the glucose concentration. Note that the $\mathbf{v}_{uptake}$ vector in Equation 3 and Equation 10 has been replaced with an expression for the lower bound of glucose (LB_G) and an expression for the lower bound of acetate (LB_A). The resulting expression of the dFBA is given below.

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, \mathbf{v}) \tag{37a}$$

$$\mathbf{x} \geq 0 \tag{37b}$$

$$LB_G = v_{G,max}\frac{x_G}{K_{M,G} + x_G} \tag{37c}$$

$$LB_A = v_{A,max}\frac{x_A}{K_{M,A} + x_A} \tag{37d}$$

$$\min_{\mathbf{v}} \quad -\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v} \tag{37e}$$

$$\text{s.t. } \mathbf{Sv} = 0 \tag{37f}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{37g}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{37h}$$

Where $F(\mathbf{x}, \mathbf{v})$ is the right-hand side of the mass balances, $v_{G,max}$ is the maximum uptake of glucose, $v_{A,max}$ is the maximum uptake of acetate, $K_{M,G}$ is the Michaelis constant for glucose, $K_{M,A}$ is the Michaelis constant for acetate, and $\mathbf{x}$ is a vector containing the concentration of the extracellular metabolites. Note that LB_G and LB_A are a part of the **LB** vector.

We solve the dFBA model, Equation 37, with the Direct Approach (DA) and the Non-Linear Programming Approach (NLPA).

3.2.1 Direct Approach for dFBA

In the DA we apply an ordinary differential equation (ODE) solver that calls the pFBA model, Equation 37, at each time step. The ODE solver calculates the metabolite concentrations, the lower bound of the substrates, and provides them to our NLP optimizer that solves the pFBA and returns the metabolic flux for each of the metabolites, see Figure 2.

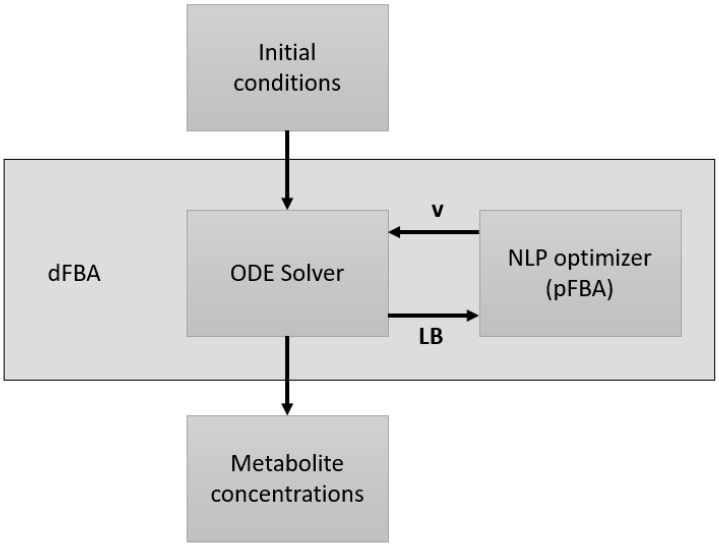


Figure 2: In the direct approach we apply an ODE solver that calls the pFBA model, Equation 37, at each time step. The ODE solver calculates the metabolite concentrations, the lower bound of the substrates (**LB**) and provides them to our NLP optimizer that solves the pFBA and returns the metabolic fluxes (**v**) at each time step.

It is decided to use an implicit, multi-step ODE solver with variable step size, because the system is reasonably large and the level of stiffness is unknown. The ODE solver Quasi-constant time step Numerical Differentiation Function (QNDF) from the package DifferentialEquations [24] is used, as it is recommended for these kinds of problems. We also added an if statement to the ODE solver to ensure that the concentrations of the extracellular metabolites do not go below zero.

The optimization problems are solved with the interior point optimizer IPOPT [28], with

the MA97 linear solver from HSL [25]. The IPOPT solver is chosen as it is designed to find local solutions for large-scale non-linear optimization problems.

We do not apply Model Predictive Control (MPC) to the DA as this would result in a bi-level optimization problem. We instead use the DA as a reference to the NLPA because the DA solves the original pFBA problem and the adaptive mesh utilized by the ODE solver makes the DA more accurate in areas with rapid change of the metabolite concentrations than the NLPA, assuming that the optimization problem returns feasible solutions at each time step.

3.2.2 Non-Linear Programming Approach for dFBA

The NLPA is the approach we are going to use for the MPC, because it does not result in a bi-level optimization problem when the MPC is applied. To solve the dFBA model with the NLPA we apply our duality theory and KKT reformulations of the pFBA problem, see Section 3.1, to the dFBA formulation, Equation 37, and discretize the set of dynamic mass balances utilizing orthogonal collocation on finite elements, see Equation 27. Resulting in the reformulations of the dFBA model presented below.

First we present the non-penalized dFBA model reformulations.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,A,X,\mathbf{LB}} \quad 1 \tag{38a}$$

$$\text{s.t.} \quad \mathbf{Sv} = 0 \tag{38b}$$

$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{38c}$$

$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{38d}$$

$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{38e}$$

$$\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{38f}$$

$$\mu_U, \mu_L \geq 0 \tag{38g}$$

$$G = G_0 + h \cdot (M \cdot F_G(G, \mathbf{v})) \tag{38h}$$

$$A = A_0 + h \cdot (M \cdot F_A(A, \mathbf{v})) \tag{38i}$$

$$X = X_0 + h \cdot (M \cdot F_X(X, \mathbf{v})) \tag{38j}$$

$$G, A, X \geq 0 \tag{38k}$$

$$LB_G = v_{G,max}\frac{G}{K_{M,G} + G} \tag{38l}$$

$$LB_A = v_{A,max}\frac{A}{K_{M,A} + A} \tag{38m}$$

Where M is the the orthogonal collocation matrix presented in Equation 26, $v_{G,max}$ is the maximum uptake of glucose, $v_{A,max}$ is the maximum uptake of acetate, $K_{M,G}$ is the Michaelis constant for glucose, $K_{M,A}$ is the Michaelis constant for acetate, G, A and X is the glucose, acetate and biomass concentrations respectively, F_i is the right-hand side of the mass balance of metabolite i, and h is the size of the finite elements. $\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$ is the strong duality expression or the complementary slackness constraints dependent on whether we use the duality theory or KKT pFBA reformulation.

Duality theory $\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$:

$$\mathbf{c}^T\mathbf{v} - 2\mathbf{v}^T\mathbf{W}\mathbf{v} + \mu_L^T\mathbf{LB} - \mu_U^T\mathbf{UB} = 0 \tag{39}$$

KKT $\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$:

$$\mu_L^T(\mathbf{LB} - \mathbf{v}) = 0 \tag{40a}$$
$$\mu_U^T(\mathbf{v} - \mathbf{UB}) = 0 \tag{40b}$$

Finally we present the penalized dFBA model reformulations.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,A,X,\mathbf{LB}} \Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{41a}$$
$$\text{s.t. } \mathbf{S}\mathbf{v} = 0 \tag{41b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{41c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{41d}$$
$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{41e}$$
$$\mu_U, \mu_L \geq 0 \tag{41f}$$
$$G = G_0 + h \cdot (M \cdot F_G(G, \mathbf{v})) \tag{41g}$$
$$A = A_0 + h \cdot (M \cdot F_A(A, \mathbf{v})) \tag{41h}$$
$$X = X_0 + h \cdot (M \cdot F_X(X, \mathbf{v})) \tag{41i}$$
$$G, A, X \geq 0 \tag{41j}$$
$$LB_G = v_{G,max}\frac{G}{K_{M,G} + G} \tag{41k}$$
$$LB_A = v_{A,max}\frac{A}{K_{M,A} + A} \tag{41l}$$

Where $\Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$ is an objective function depending on the pFBA reformulation.

Duality theory $\Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$

$$2\mathbf{v}^T\mathbf{W}\mathbf{v} - \mu_L^T\mathbf{LB} + \mu_U^T\mathbf{UB} - \mathbf{c}^T\mathbf{v} \tag{42}$$

KKT $\Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$:

$$\mu_U^T(\mathbf{UB} - \mathbf{v}) + \mu_L^T(\mathbf{v} - \mathbf{LB}) \tag{43}$$

For the orthogonal collocation utilized to discretize the dynamic mass balances, we used the Radau collocation points presented below.

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 0.155051 \\ 0.644949 \\ 1.000000 \end{bmatrix} \tag{44}$$

3.3 Reactor mass balance implementation

To complete our dFBA model we introduce the mass balances of the metabolites. The expressions of the mass balances are dependent on the type of bioreactor considered. We will look at batch reactors as they are one of the most commonly used bioreactors in industry and will allow us to compare our result with other recently developed models, such as the dFBA model developed by Oliveira et al. [18]. However, we can not apply MPC directly to our batch reactor models as the batch reactor designs do not have any manipulated variables we can use in the controller. Therefore, will also consider a CSTR, despite continuous bioreactors not being very commonly used commercially in bioprocessing [5], because it is a important and widely used abstraction of reactors in industry that allow for implementation of control structures, such as MPC.

3.3.1 Batch reactor mass balances

From Equation 6 we are given the right-hand side of the mass balances for a batch reactor with glucose, acetate, and biomass.

$$F_G(G, \mathbf{v}) = V \cdot (\mathbf{v}_G \cdot X) \tag{45a}$$
$$F_A(A, \mathbf{v}) = V \cdot (\mathbf{v}_A \cdot X) \tag{45b}$$
$$F_X(X, \mathbf{v}) = V \cdot (\mathbf{v}_X \cdot X) \tag{45c}$$

V is the volume of the reactor.

3.3.2 CSTR mass balances

From Equation 8 we are given the right-hand side of the mass balances for a CSTR with glucose, acetate, and biomass.

$$F_G(G, \mathbf{v}) = D \cdot (G_{in} - G) + \mathbf{v}_G \cdot X \tag{46a}$$
$$F_A(A, \mathbf{v}) = D \cdot (A_{in} - A) + \mathbf{v}_A \cdot X \tag{46b}$$
$$F_X(X, \mathbf{v}) = D \cdot (X_{in} - X) + \mathbf{v}_X \cdot X \tag{46c}$$

D is the dilution rate and G_{in}, A_{in}, and X_{in} are the concentration of glucose, acetate and biomass in the reactor feed respectively.

The mass balance expressions are added to our expressions of the dFBA model presented in Section 3.2.

3.4 Steady state model implementation

Before we apply MPC to our CSTR dFBA models, we present the CSTR steady state models. The steady state models are useful because they can be used to predict the final concentrations of the extracellular metabolites in the dynamic models, and to find good

starting points for our dynamic model simulations. The steady state models are developed by setting the dynamic mass balance expressions of the CSTR dFBA solved with the NLPA, Section 3.2.2, to zero.

Non-penalized dFBA steady state models implementation:

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,A,X,\mathbf{LB}} 1 \tag{47a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{47b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{47c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{47d}$$
$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{47e}$$
$$\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{47f}$$
$$\mu_U, \mu_L \geq 0 \tag{47g}$$
$$D \cdot (G_{in} - G) + \mathbf{v}_G \cdot X = 0 \tag{47h}$$
$$D \cdot (A_{in} - A) + \mathbf{v}_A \cdot X = 0 \tag{47i}$$
$$D \cdot (X_{in} - X) + \mathbf{v}_X \cdot X = 0 \tag{47j}$$
$$G, A, X \geq 0 \tag{47k}$$
$$LB_G = v_{G,max}\frac{G}{K_{M,G} + G} \tag{47l}$$
$$LB_A = v_{A,max}\frac{A}{K_{M,A} + A} \tag{47m}$$

Penalized dFBA steady state models implementation:

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,A,X,\mathbf{LB}} \Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{48a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{48b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{48c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{48d}$$
$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{48e}$$
$$\mu_U, \mu_L \geq 0 \tag{48f}$$
$$D \cdot (G_{in} - G) + \mathbf{v}_G \cdot X = 0 \tag{48g}$$
$$D \cdot (A_{in} - A) + \mathbf{v}_A \cdot X = 0 \tag{48h}$$
$$D \cdot (X_{in} - X) + \mathbf{v}_X \cdot X = 0 \tag{48i}$$
$$G, A, X \geq 0 \tag{48j}$$
$$LB_G = v_{G,max}\frac{G}{K_{M,G} + G} \tag{48k}$$
$$LB_A = v_{A,max}\frac{A}{K_{M,A} + A} \tag{48l}$$

3.5 Model Predictive Control implementation

We have models with multiple inputs and outputs with inequality constraints, and MPC is therefore a good control approach for our models [26]. A simplified block diagram of the MPC implementation is presented in Figure 3. The MPC utilizes the dFBA models solved with the NLPA and the plant is represented by the dFBA model solved with the DA. We assume that the underlying control structures, such as proportional integral derivative (PID) controllers, are very fast compared to the MPC, and therefore we do not consider them in this project.

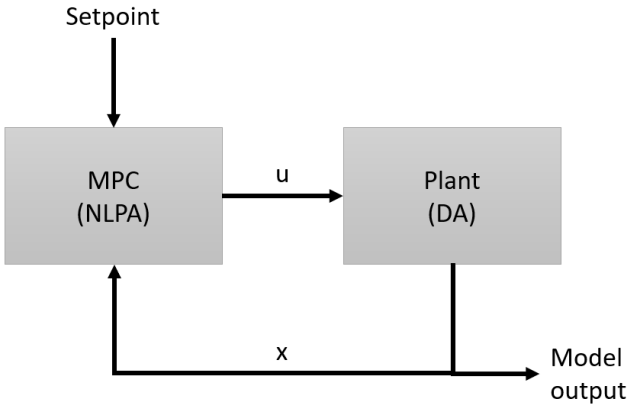


Figure 3: A simplified block diagram of the MPC implementation. The MPC utilizes the dFBA models solved with the NLPA and the plant is represented by the dFBA model solved with the DA. We assume that the underlying control structures are very fast compared to the MPC, and thus we do not consider them in this project. u is the manipulated variables and x are the states in the system.

We use the dilution rate (D) as the manipulated variable, and because we wish to control the biomass concentration in the reactor, we use the biomass concentration as the MPC setpoint.

The implementation of the MPC based on the dFBA model utilizing the non-penalized pFBA reformulations is presented below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,X,A,\mathbf{LB},D} \quad Q \cdot (X - X_{sp})^2 + R \cdot (\Delta u)^2 \tag{49a}$$

$$\text{s.t.} \quad \mathbf{S}\mathbf{v} = 0 \tag{49b}$$

$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{49c}$$

$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{49d}$$

$$-\mathbf{c} + 2\mathbf{W}\mathbf{v} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{49e}$$

$$\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{49f}$$

$$\mu_U, \mu_L \geq 0 \tag{49g}$$

$$G = G_0 + h \cdot (M \cdot (D \cdot (G_{in} - G) + \mathbf{v}_G \cdot X)) \tag{49h}$$

$$A = A_0 + h \cdot (M \cdot (D \cdot (A_{in} - A) + \mathbf{v}_A \cdot X)) \tag{49i}$$

$$X = X_0 + h \cdot (M \cdot (D \cdot (X_{in} - X) + \mathbf{v}_X \cdot X)) \tag{49j}$$

$$G, A, X \geq 0 \tag{49k}$$

$$LB_G = v_{G,max}\frac{G}{K_{M,G}+G} \tag{49l}$$
$$LB_A = v_{A,max}\frac{A}{K_{M,A}+A} \tag{49m}$$
$$\Delta D \leq -\Delta D_{max} \tag{49n}$$
$$\Delta D \geq \Delta D_{max} \tag{49o}$$
$$D \geq D_{min} \tag{49p}$$
$$D \leq D_{max} \tag{49q}$$
$$\tag{49r}$$

Where Q and R are control parameters, ΔD is the change in the dilution rate, X_{sp} is the setpoint for biomass, ΔD_{max} is the maximal change in the dilution rate, and D_{min} and D_{max} is the lowest and highest acceptable value of the dilution rate respectively.

The implementation of the MPC based on the dFBA model utilizing the penalized pFBA reformulations is presented below.

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,G,X,A,\mathbf{LB},D} \; Q\cdot(X-X_{sp})^2 + R\cdot(\Delta u)^2 + C\Phi(\mathbf{v},\mathbf{LB},\mathbf{UB}) \tag{50a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{50b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{50c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{50d}$$
$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{50e}$$
$$\mu_U, \mu_L \geq 0 \tag{50f}$$
$$G = G_0 + h\cdot(M\cdot(D\cdot(G_{in}-G) + \mathbf{v}_G\cdot X)) \tag{50g}$$
$$A = A_0 + h\cdot(M\cdot(D\cdot(A_{in}-A) + \mathbf{v}_A\cdot X)) \tag{50h}$$
$$X = X_0 + h\cdot(M\cdot(D\cdot(X_{in}-X) + \mathbf{v}_X\cdot X)) \tag{50i}$$
$$G, A, X \geq 0 \tag{50j}$$
$$LB_G = v_{G,max}\frac{G}{K_{M,G}+G} \tag{50k}$$
$$LB_A = v_{A,max}\frac{A}{K_{M,A}+A} \tag{50l}$$
$$\Delta D \leq -\Delta D_{max} \tag{50m}$$
$$\Delta D \geq \Delta D_{max} \tag{50n}$$
$$D \geq D_{min} \tag{50o}$$
$$D \leq D_{max} \tag{50p}$$
$$\tag{50q}$$

Where C is a control parameter.

4 Results and discussion

In this section, we test our models based on Dynamic Flux Balance Analysis (dFBA) developed in Section 3 with a case study of the *Escherichia coli* core metabolism. The specifications for the case study are presented in Section 4.1. First, we test our reformulations of the parsimonious Flux Balance Analysis (pFBA), and compare how the reformulations preform with emphasis on solver time and divergence from the original pFBA problem. Second, we test our dFBA models for a batch reactor and improve the accuracy of the models by assuming constant fluxes inside each finite element and by adding an adaptive mesh strategy. Third, we test our dFBA models for a continuous stirred tank reactor (CSTR). Finally, we apply Model Predictive Control (MPC) to the dFBA model reformulations using CSTR. We tune the controller and test how it handles changes in the setpoint, disturbances in the glucose feed concentration, and disturbances the maximal glucose uptake.

4.1 Case study specifications

The case study is performed with the *Escherichia coli* core metabolic network model. The *Escherichia coli* core metabolism is chosen because it is a simple and well-established metabolic model, and we can therefore compare the behavior of our model to existing literature. The stoichiometric matrix (**S**), the **c** vector, the lower bound vector (**LB**), and the upper bound vector (**UB**) of the *Escherichia coli*, strain K-12 substrain MG1655, core metabolism are gathered from the database BiGG Models [11]. The core metabolism of the *Escherichia coli* cell contains 95 metabolic reactions and 72 metabolites.

The case study is performed under aerobic growth with glucose and acetate, with the pFBA objective set to maximize the growth rate of the cell. The growth rate is given by the biomass flux, $v_{biomass}$. The diagonal elements in the **W** matrix are set to 10^{-6}, as they need to be very small.

4.2 pFBA reformulations comparison

We test the performance of our reformulations of the pFBA problem, given in Section 3.1, on the the case study presented in Section 4.1. We have changed the oxygen and glucose lower bounds to the values given by the FBA tutorial for the COBRA Toolbox [9], an existing model developed to make quantitative predictions of cellular and multicellular biochemical networks. The new lower bound values are given in Table 1.

Table 1: Lower bounds of oxygen and glucose used to test the performance of the pFBA reformulations presented in Section 3.1 on a case study of the *Escherichia coli* core metabolism. The values are gathered from the FBA tutorial for the COBRA Toolbox [9].

Parameter	Value	Unit
LB_{oxygen}	-30	$mmol \cdot gDW^{-1} \cdot h^{-1}$
$LB_{glucose}$	-10	$mmol \cdot gDW^{-1} \cdot h^{-1}$

To calculate the difference between the original pFBA and our reformulations of the pFBA we use the mean squared error (MSE). The MSE is calculated with the following expression.

$$MSE = \frac{\sum_{i=1}^{j}(y_i - \bar{y}_i)^2}{j} \tag{51}$$

Where y is a vector containing the simulation results of our reformulations, j is the number of elements in y, and $\bar{y}$ is a vector containing the predicted values we want to compere to the elements in y, in this case $\bar{y}$ is the metabolic fluxes of the original pFBA problem.

Table 2 presents the solver status, the objective function, the difference between the objective value of the original pFBA problem and the reformulations, and the MSE between the metabolic fluxes in the original pFBA and the reformulations.

Table 2: Simulation of the pFBA reformulations on a case study of the *Escherichia coli* core metabolism. In this table we present the solver status, objective function, difference between the objective value of the original pFBA problem and the reformulations, and the MSE between the metabolic fluxes from the original pFBA and from the reformulations. SS stands for solved successfully, and STAL stands for solved to acceptable level. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Status:	Objective function:	Growth rate difference:	MSE fluxes:
Original	SS	$-\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v}$	0	0
Dual	SS	1	$-5.96 \cdot 10^{-8}$	$2.06 \cdot 10^{-9}$
P. Dual	SS	$-\mathbf{c}^T\mathbf{v} + 2\mathbf{v}^T\mathbf{W}\mathbf{v} - \mu_L\mathbf{LB} + \mu_U\mathbf{UB}$	$6.06 \cdot 10^{-7}$	$1.42 \cdot 10^{-6}$
KKT	STAL	1	$1.37 \cdot 10^{-4}$	$4.46 \cdot 10^{-3}$
P. KKT	STAL	$\mu_U(\mathbf{v} - \mathbf{UB}) + \mu_L(\mathbf{LB} - \mathbf{v})$	$6.05 \cdot 10^{-7}$	$4.68 \cdot 10^{-1}$

From Table 2 we see that the IPOPT solver successfully solved (SS) the duality theory reformulations and the original pFBA formulation. However, the KKT reformulations were only solved to the solvers acceptable level (STAL). This implies that the solver algorithm did not converge to our desired tolerances, set to 10^{-8}, but found a point satisfying the acceptable tolerance level, set to 10^{-6}, which are the default tolerances of the IPOPT solver [28]. This is reflected in the KKT reformulations having a larger MSE values of the metabolic fluxes, and a larger difference in the objective value, than the solutions found for the duality theory reformulations. The non-penalized KKT approach performed worst for the objective value with a difference from the original solution equal to $1.37 \cdot 10^{-4}$, compared to the other reformulations with differences from the original solution less than 10^{-6}. The penalized KKT reformulation performed well for the difference in the objective function, but the MSE for the metabolic fluxes are very high, $4.68 \cdot 10^{-1}$, compared to the duality theory reformulations, $2.06 \cdot 10^{-9}$ and $1.42 \cdot 10^{-6}$ for the non-penalized and penalized version respectively. Overall, has the non-penalized duality theory approach the closest solution to the original pFBA problem.

The size of the optimization problems of the pFBA formulations and the time required to solve them are presented in Table 3. The time presented is the average of ten simulations.

Table 3: Simulation of the pFBA reformulations on a case study of the *Escherichia coli* core metabolism. In this table we present the size of the optimization problems and the time used to solve the original pFBA problem and its reformulations. The time presented is the average of 10 simulations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of iterations:	Number of variables:	Number of equality constraints:	Number of inequality constraints:
Original	0.016	37	95	72	190
Dual	0.037	33	357	168	380
P. Dual	0.043	46	357	167	380
KKT	0.095	83	357	169	380
P. KKT	0.194	224	357	167	380

The original optimization problem is much smaller than the optimization problems of the pFBA reformulations, as shown in Table 3. The number of variables in the original formulation are 95 while the number of variables in the reformulations are 357. The reformulations also have more equality and inequality constraints than the original problem. The original problem has 72 equality and 190 inequality constraints while the reformulations have between 167 and 169 equality constraints and 380 inequality constraints. Overall is the size difference between the reformulations very small, only varying with one or two equality constraints. It was thus expected that the original problem would be much faster to solve than the reformulations, roughly two times faster than the fastest pFBA reformulation. This is promising for the Direct Approach we later will use to solve the Dynamic FBA model (dFBA). We see that the duality theory reformulations are much faster to solve than the KKT reformulations. The non-penalized duality theory reformulation is roughly 2.5 times faster to solve than the non-penalized KKT, and the penalized duality theory reformulation is roughly 4.5 times faster to solve than the penalized KKT. We also see that the penalized reformulations are slower than their non-penalized versions. The penalized KKT is the slowest of all the approaches, requiring 0.194 seconds to solve compared to the other approaches requiring less than 0.100 seconds.

4.3 pFBA reformulations with decreased tolerances comparison

To deal with the poor performance of the KKT reformulations it is decided to decrease the constraint violation tolerance and acceptable tolerance parameters of the Ipopt solver. The acceptable tolerance level is set to our desired tolerance, and to ensure that we do not break any of our constraints we decrease the constraint violation tolerance to a very small value.

- Acceptable tolerance: $10^{-6} \rightarrow 10^{-8}$
- Constraint violation tolerance: $10^{-4} \rightarrow 10^{-10}$

We run one more simulation of the pFBA reformulations with the lower bounds presented in Table 1.

Table 4 presents the solver status, the objective function, the difference between the objective value of the original pFBA problem and the reformulations, and the MSE between the metabolic fluxes in the original pFBA and the reformulations.

Table 4: Simulation of the pFBA reformulations on a case study of the *Escherichia coli* core metabolism with decreased tolerances of the IPOPT solver. In this table we present the solver status, objective function, difference between the objective value of the original pFBA problem and the reformulations, and the MSE between the metabolic fluxes from the original pFBA and from the reformulations. The optimizers acceptable tolerance and constraint violation tolerance are reduced from the default values. SS stands for solved successfully, Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Status:	Objective function:	Growth rate difference:	MSE fluxes:
Original	SS	$-\mathbf{c}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{v}$	0	0
Dual	SS	1	$-4.38 \cdot 10^{-8}$	$4.27 \cdot 10^{-8}$
P. Dual	SS	$-\mathbf{c}^T\mathbf{v} + 2\mathbf{v}^T\mathbf{W}\mathbf{v} - \mu_L\mathbf{LB} + \mu_U\mathbf{UB}$	$6.06 \cdot 10^{-7}$	$1.10 \cdot 10^{-6}$
KKT	SS	1	$-6.62 \cdot 10^{-8}$	$4.79 \cdot 10^{-8}$
P. KKT	SS	$\mu_U(\mathbf{v} - \mathbf{UB}) + \mu_L(\mathbf{LB} - \mathbf{v})$	$6.06 \cdot 10^{-7}$	$1.31 \cdot 10^{-6}$

From Table 4 we see that all the reformulations, including the KKT reformulations, of the pFBA are solved successfully with the decreased tolerances. The absolute value of the deviation between the optimal solution of the reformulations and the original pFBA problem for the objective is very small, less than $7 \cdot 10^{-7}$. The MSE between the metabolic fluxes are also very small, less than $2 \cdot 10^{-6}$ for all the reformulations. Overall is the non-penalized reformulations closer to the original solution than their penalized versions. The non-penalized reformulations have a deviation from the original solution to the order of 10^{-8}, compared to the penalized reformulations that have a deviation to the order of 10^{-7}. Similarly are the MSE of the metabolic fluxes for the non-penalized reformulations to the order of 10^{-8} compared to the penalized versions with an order of 10^{-6}. It was expected that the non-penalized reformulations would be more accurate than the penalized ones, as the penalized versions have moved some of the constraints to the objective function and are thus more relaxed than the non-penalized reformulations. Overall are the differences in the solution between the duality theory approaches and the KKT approaches neglectable.

The size of the optimization problems of the pFBA formulations and the time required to solve them are presented in Table 5. The time is given as a average of ten simulations.

Table 5: Simulation of the pFBA reformulations on a case study of the *Escherichia coli* core metabolism with decreased tolerances of the IPOPT solver. In this table we present the size of the optimization problems and the time used to solve the original pFBA problem and its reformulations. The time presented is the average of 10 simulations and the optimizers acceptable tolerance and constraint violation tolerance are reduced from the default values. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of iterations:	Number of variables:	Number of equality constraints:	Number of inequality constraints:
Original	0.019	36	95	72	190
Dual	0.048	42	357	168	380
P. Dual	0.060	60	357	167	380
KKT	0.047	42	357	169	380
P. KKT	0.177	170	357	167	380

The difference in solver time between the non-penalized duality theory and the non-penalized KKT reformulations are neglectable, 0.001 seconds, as shown in Table 5. On the other hand, is the penalized duality theory reformulation roughly 3 times faster to

solve than the penalized KKT reformulation. The penalized approaches are slower than the non-penalized ones, with the non-penalized duality theory reformulation being roughly 25% faster than the penalized reformulation and the non-penalized KKT reformulation being roughly 3.7 times faster than the penalized reformulation.

We compare the solver time and accuracy of the reformulations for the pFBA problem before and after we decreased the tolerances of the solver. In Table 6 we present the differences as the solution with decreased tolerances minus the solution without the tolerance decrease.

Table 6: Comparison of the solver time and deviation of the pFBA reformulations from the original pFBA problem before and after the tolerances of the solver were decreased from its default values. The differences as the solution with decreased tolerances minus the solution without the tolerance decrease. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time difference: [s]	Time difference: [%]	Difference in objective:	Difference in MSE:
Original	0.003	18.75	0.00	0.00
Dual	0.011	29.73	$-1.58 \cdot 10^{-8}$	$4.06 \cdot 10^{-8}$
P. Dual	0.017	39.53	0.00	$-3.02 \cdot 10^{-7}$
KKT	-0.048	-50.53	$-1.37 \cdot 10^{-4}$	$-4.46 \cdot 10^{-3}$
P. KKT	-0.017	-8.76	$1.00 \cdot 10^{-9}$	$-4.68 \cdot 10^{-1}$

From Table 6 we see that the solver time of the original problem and the duality theory reformulations increased when we decreased the tolerances. The time to solve the original problem increased by roughly 18.75 %, and the duality reformulations increased by roughly 29.73 % for the non-penalized version and 39.53 % for the penalized versions. On the other hand, we see that the solver time of the KKT reformulations has decreased. The solver time of the non-penalized KKT reformulation decreased by roughly 50.53% and the solver time for the penalized version decreased by roughly 8.76 %. Overall the accuracy of the reformulations increases or has an insignificant decrease, less than 10^{-6}, when we decreased the tolerances. We see the largest improvement in accuracy for the MSE of the metabolic fluxes in the penalized KKT reformulation with a reduction of the difference between the reformulation and original pFBA of $-4.68 \cdot 10^{-1}$.

Overall we see that the decreased tolerances are necessary for the accuracy of the KKT reformulations, while the change in the accuracy of the duality theory reformulations is neglectable. Despite the time increase for the duality theory reformulations it is decided to keep the decreased tolerances for all the pFBA reformulations. The reasoning for this is that more constraints are added when we introduce the dFBA model which will affect the solver time, and we want a fair comparison between the duality theory and KKT reformulations.

4.4 Batch bioreactor model based on dFBA

We test our dFBA model reformulations for a batch reactor, see Section 3.2 and Section 3.3. The initial conditions of our dFBA models are gathered from Oliveira et al. [18] and provided in Table 7. Note that the Michaelis constant for acetate ($K_{A,M}$) is not given, but we assume that it is equal to the Michaelis constant for glucose ($K_{G,M}$).

Table 7: Initial conditions for the simulations of the batch bioreactor model based on dFBA. The parameters are gathered from Oliveira et al. [18]. The Michaelis constant of acetate ($K_{A,M}$) is not given, but we assume that it is equal to the Michaelis constant of glucose ($K_{G,M}$)

Parameter:	Symbol:	Value:	Unit:
Size of finite elements	h	0.888	h
Initial glucose concentration	G_0	10.5	mmol/L
Initial biomass concentration	X_0	0.01	gDW/L
Initial acetate concentration	A_0	0.10	mmol/L
Michaelis constant glucose	$K_{G,M}$	0.01	mmol/L
Michaelis constant acetate	$K_{A,M}$	0.01	mmol/L
Maximal glucose uptake	$v_{max,G}$	-10.5	mmol/gDW h
Maximal acetate uptake	$v_{max,A}$	-2.5	mmol/gDW h
Oxygen lower bound	LB_O	-19.0	mmol/gDW h

As glucose is the main source of substrate in the reactor we stop the simulation right before we run out of glucose. It was found that the glucose is consumed after roughly 5.33 hours, by running a simulation with the direct approach (DA), and 6 finite elements are used in the orthogonal collocation used for the Non-Linear Programming Approach (NLPA).

The solver time, number of times the pFBA where solved in the DA, the solver status, and the MSE between the glucose, biomass and acetate concentration calculated by the NLPA reformulations and the DA are presented in Table 8.

Table 8: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. MIE stands for maximum number of iterations reached, STAL stands for solved to acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	7.65	242	SS/STAL	0	0	0
Dual	113	-	MIE	$2.12 \cdot 10^{-4}$	$4.60 \cdot 10^{-7}$	$8.23 \cdot 10^{-4}$
P. Dual	22.1	-	SS	$2.12 \cdot 10^{-4}$	$4.60 \cdot 10^{-7}$	$8.22 \cdot 10^{-4}$
KKT	11.1	-	SS	$2.12 \cdot 10^{-4}$	$4.60 \cdot 10^{-7}$	$8.23 \cdot 10^{-4}$
P. KKT	7.79	-	SS	$2.12 \cdot 10^{-4}$	$4.60 \cdot 10^{-7}$	$8.22 \cdot 10^{-4}$

The DA called the non-linear problem (NLP) solver 242 times, and the NLP solver solved successfully (SS) or to an acceptable level (STAL) each time, see Table 8. The penalized NLPA reformulations and the non-penalized KKT reformulation are also solved successfully. The non-penalized duality theory NLPA reached the maximum number of iterations, 3000, which implies that the reformulation did not converge, and thus failed to solve successfully. 3000 is the default maximum number of iterations and was not increased, as model reformulations requiring this amount of iterations are to slow to compete with the other reformulation. We expected that the non-penalized reformulations would fail to solve successful as they are less robust than the penalized reformulations. However, we also expected the non-penalized KKT reformulation to fail to converge as the IPOPT-solver should have more difficulty solving the complementary constraint problem present in the KKT reformulation.

All the reformulations returned the same result, except for a small difference in the MSE of acetate for the non-penalized and penalized reformulations. We did not expect any large deviation between the NLPA reformulations, as they all solve the same mathematical problem. However, as the penalized reformulations are more relaxed, as some of the hard constraints are moved to the objective, are the small deviation between the penalized and non-penalized methods expected.

The non-penalized duality theory reformulation is the slowest method requiring 113 seconds to solve, compered to the penalized duality theory reformulation, which is the second slowest method, requiring 22.1 seconds to solve. The fastest method is the DA which used 7.65 seconds to solve. This is quite close to the penalized KKT reformulation which required 7.79 seconds. It was expected that the DA would be one of the fastest methods as the time span is quite small, 5.33 hours, and the ODE solver therefore only need to call the NLP-optimizer 242 times. Overall we see that the KKT reformulations are faster than the duality theory reformulations and that the penalized reformulations are faster than the non-penalized reformulations. We did not expect that the penalized KKT reformulation would be faster than the penalized duality theory reformulations as the penalized duality reformulations were faster than the penalized KKT reformulation in the test of the pFBA in Section 4.3. Nor did we expect such a large difference in solver time between the non-penalized KKT and duality theory reformulations, as they preformed quite similarly when we tested the reformulations of the pFBA problem in Section 4.3. However, this is most likely a result of the duality theory reformulation failing to converge for the non-penalized duality theory reformulation.

The concentration profiles of biomass, glucose and acetate are presented in Figure 4, Figure 5 and Figure 6 respectively.

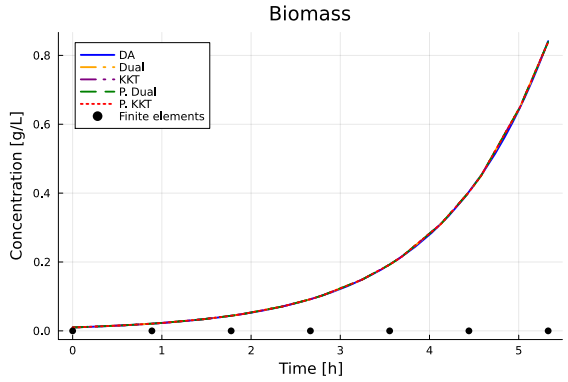


Figure 4: Concentration profile of the biomass for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

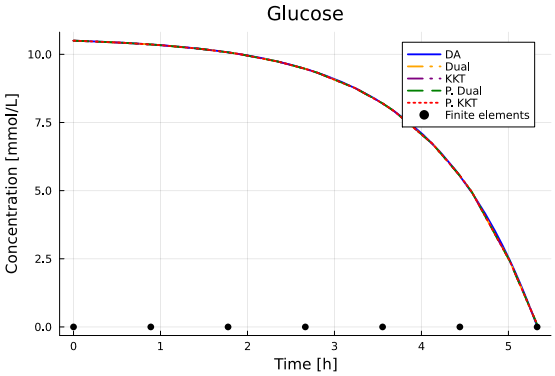


Figure 5: Concentration profile of the glucose for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

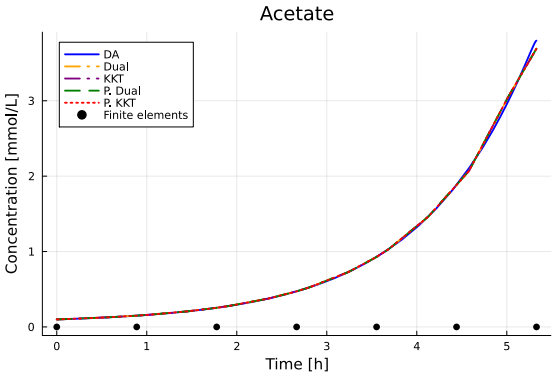


Figure 6: Concentration profile of the acetate for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 4, Figure 5 and Figure 6 it is apparent that the concentration profiles calculated by the NLPA reformulations follow the concentration profiles of the DA closely. This is consistent with the small values of the MSE, presented in Table 8. The shape of the concentration profiles are the same as those found by Oliveira et al. [18].

We also present the metabolic fluxes of the extracellular metabolites for some of the finite elements in the NLPA models. The fluxes of the extracellular metabolites are useful to see how the changes in concentration affects the cellular behavior at each time step. As the difference in the results of the different reformulations is neglectable, we will only be considering the penalized duality theory reformulation. The metabolic fluxes of the extracellular metabolites found by the penalized duality theory reformulation are presented in Table 9. The fluxes are given for each point in a finite element and three of the finite elements are presented.

Table 9: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the metabolic fluxes of the metabolic reactions found by the penalized duality theory reformulation for some of the finite elements. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. Point 1 refers to the first point in a finite element, Point 2 refers to the second point in a finite element, and Point 3 refers to the third point in a finite element.

Metabolite:	Time: [s]	Concentration: [g/L and mmol/L]	Point 1:	Point 2:	Point 3:
Biomass	0.14 - 0.89	0.01 - 0.02	$8.32 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
	1.91 - 2.67	0.05 - 0.09	$8.32 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
	4.58 - 5.33	0.45 - 0.84	$8.32 \cdot 10^{-1}$	$8.31 \cdot 10^{-1}$	$7.85 \cdot 10^{-1}$
Glucose	0.14 - 0.89	10.48 - 10.36	-10.49	-10.49	-10.49
	1.91 - 2.67	10.01 - 9.47	-10.49	-10.49	-10.49
	4.58 - 5.33	4.95 - 0.09	-10.48	-10.46	-9.46
Acetate	0.14 - 0.89	0.11 - 0.15	3.78	3.78	3.78
	1.91 - 2.67	0.28 - 0.47	3.78	3.78	3.78
	4.58 - 5.33	2.07 - 3.70	3.76	3.72	1.73

From Table 9 it is apparent that the metabolic fluxes of biomass, glucose and acetate are dependent on the glucose concentration. The metabolic fluxes are more or less constant until the glucose concentration approaches zero in the third point in the last element. At this point the absolute value of the metabolic fluxes decrees. This behavior was expected as the cells consume as much glucose as possible and turn it into biomass and acetate. When the glucose concentration decreases less glucose enters the cells, as described by the Michaelis-Menten kinetics, resulting in less energy and material being available to the cells to produce biomass and acetate.

4.5 Batch bioreactor model based on dFBA - increased time span

We increase the time span for the dFBA models used in Section 4.4 to include the point where the acetate in the reactor is consumed. This point was found by a simulation with the DA to be at roughly 7.1 hours. The orthogonal collocation used to discretize the mass balances in the NLPA reformulations uses 10 finite elements, resulting in a length of 0.71 hours. The initial conditions are presented in Table 7.

The solver time, number of times the pFBA where solved in the DA, the solver status, and the MSE between the glucose, biomass and acetate concentration calculated by the NLPA reformulations and the DA are presented in Table 10.

Table 10: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements. IPD stands for infeasible problem detected, MIE stands for maximum number of iterations reached, STAL stands for solved to acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	9.26	559	SS/STAL	0	0	0
Dual	596	-	MIE	7.98	$4.63 \cdot 10^{-2}$	1.70
P. Dual	69.5	-	SS	8.70	$5.07 \cdot 10^{-2}$	1.70
KKT	192	-	IPD	8.64	$5.05 \cdot 10^{-2}$	1.65
P. KKT	75.4	-	STAL	8.69	$5.06 \cdot 10^{-2}$	1.70

From the MSE is it apparent that the NLPA reformulations diverge a lot from the DA, see Table 10. The MSE for all the three extracellular metabolites are very high, especially for glucose, which is between 7.98 and 8.70, and for the acetate, which is between 1.65 and 1.70. This was expected as Oliveira et al. [18] also got large deviations when compering his NLPA reformulations of the dFBA model to the DA approach.

From the solver status of the NLPA reformulations we see that the non-penalized duality theory approach reached the maximum number of iterations (MIE) and that the non-penalized KKT approach resulted in an infeasible problem (IPD). Thus these two approaches failed to find the optimal solution. The penalized duality reformulation solved the problem successfully (SS) and the penalized KKT reformulation solved the problem to an acceptable level (STAL), but we see from the MSE that the results from the penalized reformulations are very similar to the result from the non-penalized ones. Implying that also the penalized approaches failed to find the correct solution. As the penalized reformulations are more robust than the penalized ones, is it expected that the penalized reformulations can return solver status as successfully solved, despite the non-penalized reformulations returning failures. On the other hand, did we not expect the results between the non-penalized and penalized NLPA reformulations to be so close when the non-penalized reformulation fails to find the optimal solution.

The concentration profiles of biomass, glucose and acetate are presented in Figure 7, Figure 8 and Figure 9 respectively.

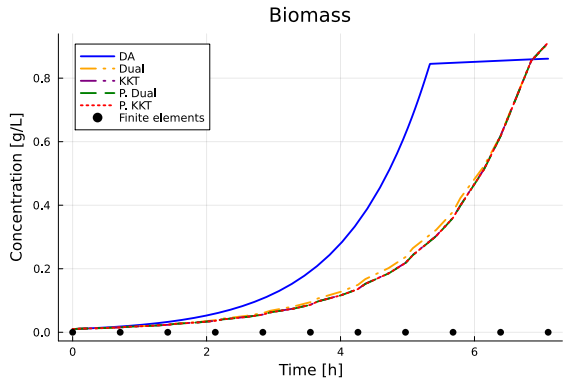


Figure 7: Concentration profile of the biomass for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

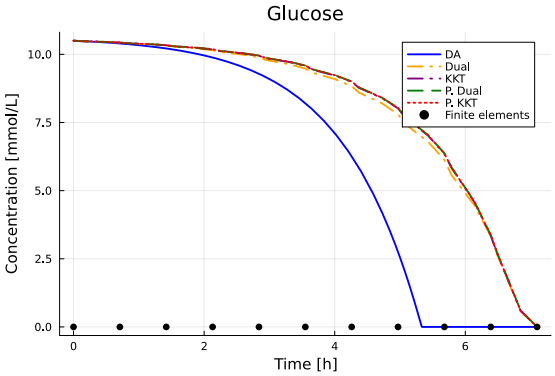


Figure 8: Concentration profile of the glucose for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

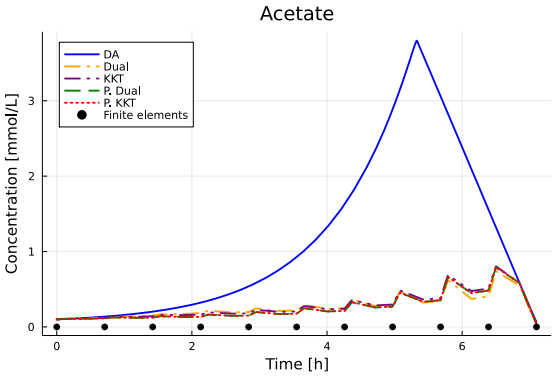


Figure 9: Concentration profile of the acetate for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch simulation is run modeled for 7.1 hours with 10 finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 7, Figure 8 and Figure 9 is it apparent that the concentration profiles calculated by the NLPA reformulations of the dFBA do not fit the DA concentration profiles. This was expected from the high MSE values in Table 10. We see that the glucose is consumed slower in the NLPA reformulations than the DA reformulation, resulting in a slower increase in the biomass concentration of the NLPAs. From Figure 9 we also see that the acetate concentration profile is rapidly changing between increasing and decreasing, implying that some of the acetate is consumed before all the glucose is used up by the cells. This was expected as Oliveira et al. [18] got similar results from his models, and found that the NLPAs do not find the correct solution because of rapid changes in the metabolite fluxes inside each finite element.

To confirm that we experience the same problem as the one identified by Oliveira et al. [18] we present the metabolic fluxes of the extracellular metabolites at each time step for four of the finite elements. As the solution from the different NLPA reformulations are very similar we are only considering the penalized duality theory reformulation. The metabolic fluxes of the extracellular metabolites are presented in Table 11.

Table 11: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the metabolic fluxes of the metabolic reactions found by the penalized duality theory reformulation for some of the finite elements. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements. Point 1 refers to the first point in a finite element, Point 2 refers to the second point in a finite element, and Point 3 refers to the third point in a finite element.

Metabolite:	Time: [s]	Concentration: [g/L and mmol/L]	Point 1:	Point 2:	Point 3:
Biomass	0.11 - 0.71	0.01 - 0.02	$8.32 \cdot 10^{-1}$	$3.70 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
	2.95 - 3.55	0.06 - 0.09	$8.32 \cdot 10^{-1}$	$4.15 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
	5.08 - 5.68	0.24 - 0.36	$8.32 \cdot 10^{-1}$	$5.90 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
	6.50 - 7.10	0.67 - 0.91	$8.04 \cdot 10^{-1}$	$4.84 \cdot 10^{-1}$	$2.08 \cdot 10^{-9}$
Glucose	0.11 - 0.71	10.48 - 10.43	-10.49	-4.37	-10.49
	2.95 - 3.55	9.86 - 9.59	-10.49	-4.41	-10.49
	5.08 - 5.68	7.70 - 6.37	-10.49	-6.25	-10.48
	6.50 - 7.10	2.64 - 0.00	-9.88	-5.13	$-1.03 \cdot 10^{-4}$
Acetate	0.11 - 0.71	0.11 - 0.11	3.78	$-5.43 \cdot 10^{-1}$	3.78
	2.95 - 3.55	0.20 - 0.18	3.78	-2.36	3.78
	5.08 - 5.68	0.45 - 0.35	3.78	-2.43	3.77
	6.50 - 7.10	0.79 - 0.03	2.56	-2.46	-1.97

From Table 11 we see the changes in the metabolic fluxes in each finite element as described by Oliveira et al. [18]. The biomass flux is smaller in the second collocation point of the finite elements than expected compared to the first and third collocation points. The glucose and acetate fluxes show that both acetate and glucose are consumed in the second collocation point.

4.6 Batch bioreactor model based on dFBA - constant metabolic fluxes in each finite element

Oliveira et al. [18] found that the failure of the NLPA reformulations in Section 4.5 are contributed to the rapid changes in the metabolic fluxes inside each finite element. The rapid changes gives rise to convergence problems for the optimization problems. Similar convergence problems were also reported by Biegler et al. [2].

Oliveira et al. solve this problem by assuming that the metabolic fluxes are constant inside of each finite element. Considering the MMK, Equation 10, used to calculate the lower bounds of the substrates it is apparent that this assumption is valid as long as the concentration of substrates is large compared to the value of the Michaelis constant, K_M, associated with the substrate.

Oliveira et al. suggest adding constraints to the metabolic fluxes to prevent the fluxes to change considerably inside the finite elements. This was done by setting the change between each metabolic flux in the finite element equal to zero.

$$v_i^k - v_i^{k-1} \leq 0 \tag{52}$$

Where v_i^k is the metabolic flux of the ith metabolic reaction in collocation point k.

However, this implementation has two main drawbacks. Firstly, we have to solve the pFBA three times inside each of the finite elements despite only using one flux value. Resulting in the system being unnecessary large. Secondly, the flux constraints may contradict the constraints for the lower bounds.

Oliveira et al. [18] proposes a more efficient implementation in his code [17], where the pFBA is only solved once for each finite element, by only using the concentrations of the substrates in one of the collocation points when solving the pFBA.

In Figure 10 we present the concentration profiles of glucose for the penalized duality theory NLPA dFBA model reformulation using the concentration in the first, second and last collocation point in each finite element to solve the pFBA.

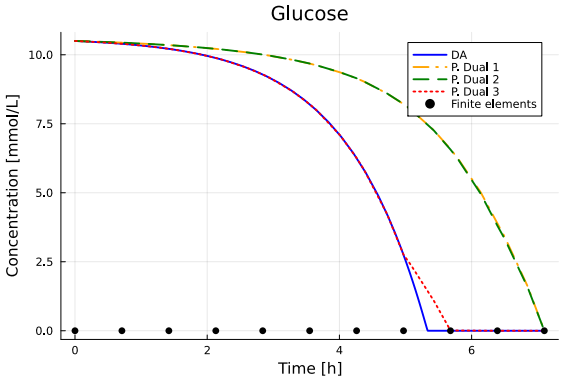


Figure 10: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this figure we present the concentration profiles of glucose calculated from the penalized duality theory NLPA using the metabolic fluxes calculated in the first, second and last collocation point in each finite element. DA refers to the concentration profile calculated by the direct approach, P. Dual 1 refers to the reformulation using the metabolic fluxes calculated in the first collocation point, P. Dual 2 refers to the reformulation using the metabolic fluxes calculated in the second collocation point, and P. Dual 3 refers to the reformulation using the metabolic fluxes calculated in the third collocation point.

It is apparent that only the model utilizing the last collocation point fit the glucose concentration from the DA dFBA model, as shown in Figure 10. This was expected because using the substrate concentrations from one of the previous points will result in a too large growth rate, resulting in an overshoot. Consider the collocation point right before the glucose concentration reaches zero. If we use one of the first two collocation points, the growth rate in the last point will be too large, and the concentration of glucose will dip below zero. This violates the constraints for the concentrations being larger or equal to zero, thus resulting in an infeasible problem. It was thus decided to solve the pFBA with the substrate concentrations from the last collocation point in each finite element.

Our new NLPA dFBA model reformulations use the third collocation point to calculate the lower bound of the substrates, replacing the lower bounds of glucose and acetate with the expressions presented below.

$$LB_G = v_{G,max} \frac{G_3}{K_{M,G} + G_3} \tag{53a}$$

$$LB_A = v_{A,max} \frac{A_3}{K_{M,A} + A_3} \tag{53b}$$

G_3 is the glucose concentration and A_3 is the acetate concentration at the third collocation point in each finite element.

We run a new simulation similar to the one in Section 4.5. The solver time, number of times the pFBA where solved in the DA, the solver status, and the MSE between the

glucose, biomass and acetate concentration calculated by the NLPA reformulations and the DA are presented in Table 12.

Table 12: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements and constant metabolic fluxes in each of the finite elements. RF stands for restoration failed, IPD stands for infeasible problem detected, STAL stands for solved to acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	9.26	559	SS/STAL	0	0	0
Dual	47.3	-	RF	$4.30 \cdot 10^{-2}$	$2.48 \cdot 10^{-4}$	$3.70 \cdot 10^{-2}$
P. Dual	5.88	-	SS	$4.32 \cdot 10^{-2}$	$2.37 \cdot 10^{-4}$	$7.16 \cdot 10^{-2}$
KKT	12.7	-	IPD	$4.30 \cdot 10^{-2}$	$2.48 \cdot 10^{-4}$	$3.70 \cdot 10^{-2}$
P. KKT	4.34	-	SS	$4.32 \cdot 10^{-2}$	$2.37 \cdot 10^{-4}$	$7.16 \cdot 10^{-2}$

From Table 12 we see that the solver fails to converge for the non-penalized NLPA reformulations. However, despite failing to find the optimal solution, the non-penalized NLPA reformulations still return results close to the DA and penalized NLPA reformulations. With the MSE of the glucose and acetate concentration being smaller for the the non-penalized NLPAs than the MSE of the penalized NLPA reformulations. The penalized NLPAs solved successfully and the MSE for the extracellular metabolites are relatively small, between $2.37 \cdot 10^{-4}$ and $7.16 \cdot 10^{-2}$. The failure of the non-penalized reformulations and the similar MSE values for the different NLPA reformulations was expected as these results are quite similar to our previous ones.

We see a large improvement in the solver time of the NLPA reformulations for the new dFBA model, and both penalized reformulations are solved faster than the DA reformulation of the dFBA model. This was expected as we only solve the pFBA one time for each finite element instead of three. Overall is the KKT reformulations faster to solve than the duality theory reformulations.

The concentration profiles of biomass, glucose and acetate are presented in Figure 11, Figure 12 and Figure 13 respectively.

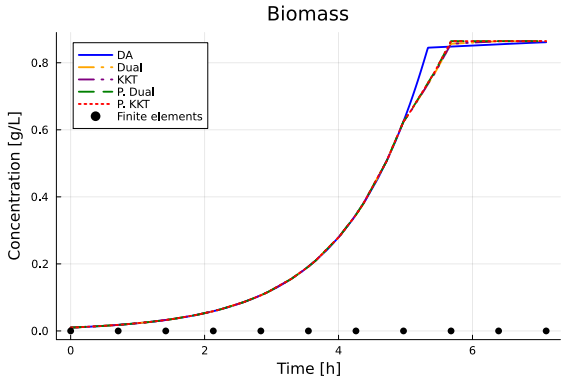


Figure 11: Concentration profile of the biomass for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

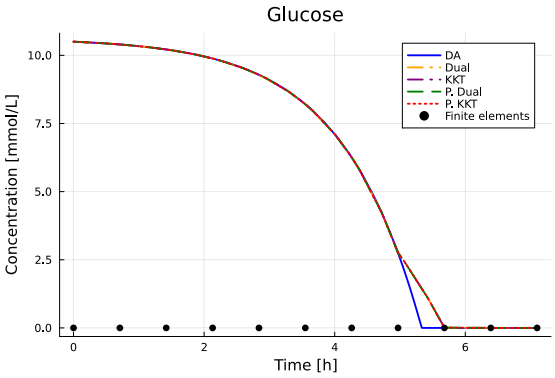


Figure 12: Concentration profile of the glucose for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

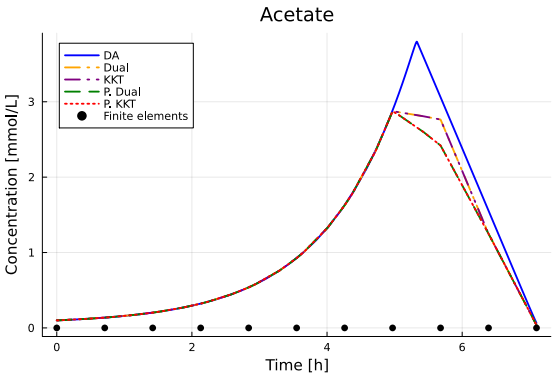


Figure 13: Concentration profile of the acetate for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 11, Figure 12 and Figure 13 we see that the concentration profiles of the metabolites calculated by the NLPA reformulations are very close to the concentration profiles found by the DA. The difference in the results is partly contributed to the placement of the finite elements. As the glucose concentration reaches zero inside one of the finite elements, are the metabolic fluxes changing considerably inside the element at this point. This is in violation with our assumption of the fluxes not changing inside the finite elements. This results in a premature decrease in the acetate concentration, as only the

last collocation point is used to calculate the metabolic fluxes. Oliveira et al. [18] got similar results and found that the accuracy of the models can be improved by introducing an adaptive mesh strategy to adjust the size and placement of the finite elements.

4.7 Batch bioreactor model based on dFBA - adaptive mesh

To improve the accuracy of the dFBA model presented in Section 4.6 Oliveira et al. [18] introduced an adaptive mesh where he let the size of the finite elements (h) change freely based on the equidistant element size (h^*).

$$0 \leq h_i \leq 2h^* \tag{54a}$$

$$\sum_{i=1}^{N} h_i = t_{end} \tag{54b}$$

Where N is the total number of finite elements and t_{end} is the end time.

We introduce a similar approach to Oliveira et al. [18], but we add an additional penalization term to the optimization objective. This term penalises the sum of the change in the glucose concentration in a finite element multiplied with the step size of the finite element. We added this term as the solver did not place the finite elements in the desired locations just using the implementation suggested by Oliveira et al., and this term thus gives us more control over the placement of the finite elements. For the batch reactor we decided to use h^* equal to 0.5 hours instead of 0.71 hours, because we want to keep the finite elements slightly closer to another.

Non-penalized dFBA model implementation:

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,h} \sum_{n=1}^{N} h_n(G_{1,n} - G_{3,n}) \tag{55a}$$

$$\text{s.t. } \mathbf{Sv} = 0 \tag{55b}$$

$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{55c}$$

$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{55d}$$

$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{55e}$$

$$\Psi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) \tag{55f}$$

$$\mu_U, \mu_L \geq 0 \tag{55g}$$

$$G = G_0 + h \cdot (M \cdot F_G(G, \mathbf{v})) \tag{55h}$$

$$A = A_0 + h \cdot (M \cdot F_A(A, \mathbf{v})) \tag{55i}$$

$$X = X_0 + h \cdot (M \cdot F_X(X, \mathbf{v})) \tag{55j}$$

$$G, A, X \geq 0 \tag{55k}$$

$$LB_G = v_{G,max} \frac{G_3}{K_{M,G} + G_3} \tag{55l}$$

$$LB_A = v_{A,max} \frac{A_3}{K_{M,A} + A_3} \tag{55m}$$

$$\sum_{n=1}^{N}(h_n) = t_{end} \tag{55n}$$
$$h \leq 2h^* \tag{55o}$$
$$h \geq 0 \tag{55p}$$

Where $G_{1,n}$ is the glucose concentration at the first collocation point in finite element n, $G_{3,n}$ is the glucose concentration at the third collocation point in finite element n, and h_n is the length of collocation point n.

Penalized dFBA model implementation:

$$\min_{\mathbf{v},\lambda,\mu_U,\mu_L,h} \Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB}) + 0.1\sum_{n=1}^{N} h_n(G_{1,n} - G_{3,n}) \tag{56a}$$
$$\text{s.t. } \mathbf{Sv} = 0 \tag{56b}$$
$$\mathbf{LB} - \mathbf{v} \leq 0 \tag{56c}$$
$$\mathbf{v} - \mathbf{UB} \leq 0 \tag{56d}$$
$$-\mathbf{c} + 2\mathbf{Wv} + \mathbf{S}^T\lambda - \mu_L + \mu_U = 0 \tag{56e}$$
$$\mu_U, \mu_L \geq 0 \tag{56f}$$
$$G = G_0 + h \cdot (M \cdot F_G(G, \mathbf{v})) \tag{56g}$$
$$A = A_0 + h \cdot (M \cdot F_A(A, \mathbf{v})) \tag{56h}$$
$$X = X_0 + h \cdot (M \cdot F_X(X, \mathbf{v})) \tag{56i}$$
$$G, A, X \geq 0 \tag{56j}$$
$$LB_G = v_{G,max}\frac{G_3}{K_{M,G} + G_3} \tag{56k}$$
$$LB_A = v_{A,max}\frac{A_3}{K_{M,A} + A_3} \tag{56l}$$
$$\sum_{n=1}^{N}(h_n) = t_{end} \tag{56m}$$
$$h \leq 2h^* \tag{56n}$$
$$h \geq 0 \tag{56o}$$

We run a simulation with our adaptive mesh for the dFBA batch model reformulations. The initial conditions are the same as those presented for our simulation in Section 4.5.

The solver time, number of times the pFBA where solved in the DA, the solver status, and the MSE between the glucose, biomass and acetate concentration calculated by the different NLPA reformulations and the DA are presented in Table 13.

Table 13: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements, constant metabolic fluxes in each of the finite elements, and adaptive mesh. MIE stands for maximum number of iterations reached, STAL stands for solved to acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	9.26	559	SS/STAL	0	0	0
Dual	84.5	-	MIE	$1.67 \cdot 10^{-3}$	$6.88 \cdot 10^{-6}$	$6.86 \cdot 10^{-3}$
P. Dual	9.55	-	SS	$1.62 \cdot 10^{-3}$	$6.67 \cdot 10^{-6}$	$6.64 \cdot 10^{-3}$
KKT	81.9	-	SS	$1.62 \cdot 10^{-3}$	$6.67 \cdot 10^{-6}$	$6.64 \cdot 10^{-3}$
P. KKT	10.0	-	SS	$1.62 \cdot 10^{-3}$	$6.67 \cdot 10^{-6}$	$6.64 \cdot 10^{-3}$

Comparing the result from the dFBA model using the adaptive mesh strategy, see Table 13, to the results from the dFBA model without the adaptive mesh, see Table 12, we see that the NLPA reformulations utilizing the adaptive mesh is much closer to the solution of the DA than the NLPA reformulations not utilizing the adaptive mesh strategy.

Considering the penalized NLPA reformulations that were solved successfully in both cases we see that the MSE of glucose is reduced from $4.32 \cdot 10^{-2}$ to $1.62 \cdot 10^{-3}$, the MSE of biomass is reduced from $2.37 \cdot 10^{-4}$ to $6.67 \cdot 10^{-6}$, and the MSE of acetate is reduced from $7.16 \cdot 10^{-2}$ to $6.64 \cdot 10^{-3}$. Similar results are found for the non-penalized NLPAs. However, we see that the time required to solve the systems have increased for the NLPA reformulations. With the exception of the non-penalized KKT reformulation are the solver times of the NLPAs has roughly doubled. The solver time of the non-penalized KKT increased much more, from 12.7 seconds to 81.9 seconds. Despite the increase in the solver time are the penalized NLPA reformulations still able to compete with the DA, as the difference between the penalized NLPA reformulations and the DA is less than one second. Overall is the increase in required solver time expected as we have increase the size of the system by adding additional variables and constraints to our optimization problems.

The concentration profiles of biomass, glucose and acetate are presented in Figure 14, Figure 15 and Figure 16 respectively.

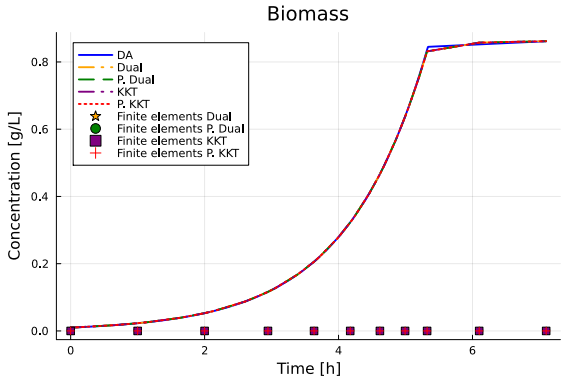


Figure 14: Concentration profile of the biomass for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements, constant metabolic fluxes in each of the finite elements, and adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

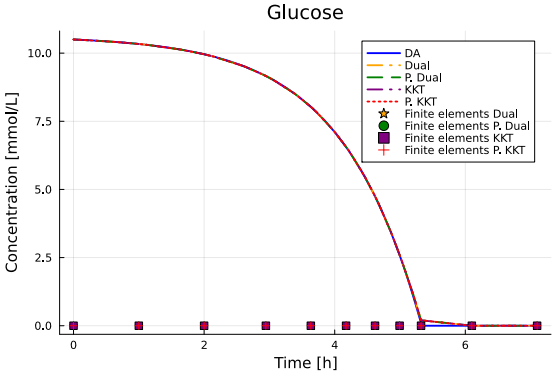


Figure 15: Concentration profile of the glucose for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements, constant metabolic fluxes in each of the finite elements, and adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

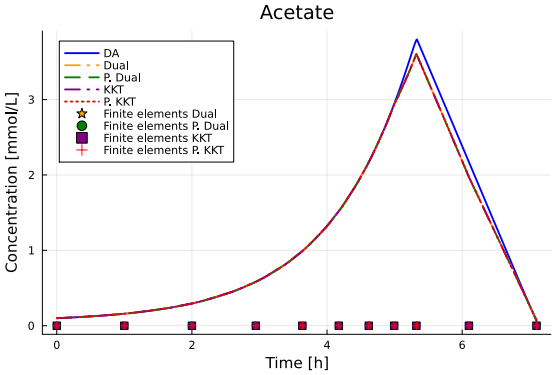


Figure 16: Concentration profile of the acetate for the batch bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements, constant metabolic fluxes in each of the finite elements, and adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 14, Figure 15 and Figure 16 we see that all the solvers place the finite elements in the same positions. We also see that the concentration profiles of the NLPA reformulations are very close to the concentration profiles of the DA. However, there is still a small deviation where the glucose concentration reaches zero, despite one finite element starting roughly at this point. Resulting in the following finite element having a small decrease of the glucose concentration when it should be zero. This was expected as our dFBA model is

less accurate where we have low substrate concentrations, as a consequence of our assumption that the fluxes are constant in each finite element and the MMK used to describe the substrate uptake to the cells. Another drawback of the NLPA model reformulations is that they have difficulties in describing the behavior of the metabolites when one of the concentration reaches zero, as a result of the assumption of constant fluxes. For example when the glucose concentration in the third collocation point reaches zero will the lower bound of the glucose uptake be set to zero for the entire element, resulting in glucose present in the first two collocation points not being consumed. Thus the metabolite concentrations can not reach zero. This problem is also present in the DA reformulation, but hidden in the sheer amount of points used by the ODE solver to describe the behavior in this area. Another limitation of our dFBA model utilizing the adaptive mesh strategy is that we have to gather some information about the behavior of the substrates, in particular which substrate that reaches zero first, to be able to initialize the adaptive mesh penalization term in the objective function.

4.8 CSTR bioreactor model based on dFBA

As we wish to apply MPC to our dFBA model, we must change from a batch reactor to a reactor design with possible manipulated variables that can be used in control applications. We decided to use a CSTR bioreactor because it is an important abstraction of reactors in industry, that allow for implementation of control structures. We test our dFBA model developed in Section 4.6 for a batch reactor on a CSTR by replacing the mass balances for batch processes with the mass balances for CSTR processes, presented in Equation 46.

The initial conditions of our dFBA model are provided in Table 14. The values of the maximal glucose and acetate uptake, the lower bound of oxygen, and the Michaelis constant for glucose are gathered from Oliveira et al. [18] and are the same as those used to initialize the batch reactor in Section 4.4. We assume that the Michaelis constant for acetate is equal to the one given for glucose and decided to use a feed of glucose roughly half the size of the maximal glucose uptake. The parameters for the dilution rate and the extracellular metabolite concentrations are found by a combination of using the steady state model, presented in Section 3.4, and by trial and error where the goal was to start close to, but not at, steady state. It was decided to stop the simulation after 30 hours and to use 30 finite elements.

Table 14: Initial conditions for the simulations of the CSTR bioreactor model based on dFBA for a case study of the *Escherichia coli* core metabolism. The parameters are gathered from Oliveira et al. [18]. The Michaelis constant of acetate ($K_{A,M}$) is not given, but we assume that it is equal to the Michaelis constant of glucose ($K_{G,M}$), and the parameters for the dilution rate and the extracellular metabolite concentrations are found by trial and error where the goal was to start close to, but not at steady state.

Parameter:	Symbol:	Value:	Unit:
Size of finite elements	h	1.00	h
Initial glucose concentration	G_0	1.00	mmol/L
Initial biomass concentration	X_0	0.30	gDW/L
Initial acetate concentration	A_0	1.00	mmol/L
Michaelis constant glucose	$K_{G,M}$	0.01	mmol/L
Michaelis constant acetate	$K_{A,M}$	0.01	mmol/L
Maximal glucose uptake	$v_{max,G}$	-10.5	mmol/gDW h
Maximal acetate uptake	$v_{max,A}$	-2.5	mmol/gDW h
Oxygen lower bound	LB_O	-19.0	mmol/gDW h
Dilution rate	D	0.80	h^{-1}
Glucose feed concentration	G_f	5.00	mmol/L
End time	t_{end}	30	h

The solver time, the number of times the pFBA were solved in the DA, the solver status, and the MSE between the glucose, biomass and acetate concentration calculated by the NLPA reformulations and the DA are presented in Table 15.

Table 15: Simulation of the CSTR dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements and constant metabolic fluxes in each of the finite elements. STAL stands for solved to an acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	6.96	418	SS/STAL	0	0	0
Dual	4.54	-	SS	$2.00 \cdot 10^{-4}$	$1.24 \cdot 10^{-6}$	$1.26 \cdot 10^{-4}$
P. Dual	17.2	-	SS	$2.01 \cdot 10^{-4}$	$1.24 \cdot 10^{-6}$	$1.26 \cdot 10^{-4}$
KKT	9.14	-	SS	$2.01 \cdot 10^{-4}$	$1.24 \cdot 10^{-6}$	$1.26 \cdot 10^{-4}$
P. KKT	5.24	-	SS	$2.01 \cdot 10^{-4}$	$1.24 \cdot 10^{-6}$	$1.26 \cdot 10^{-4}$

The time required to solve the KKT NLPA reformulations is relatively small, below 10 seconds, and the non-penalized KKT reformulation requires more time to solve than the penalized KKT reformulation. This corresponds well with the results from the batch reactor simulation in Section 4.6. The computational time required to solve the duality theory NLPAs is the opposite of what we expected from our previous simulations, with the non-penalized reformulation being very fast, 4.54 seconds, and the penalized reformulation being the slowest of all the NLPA reformulation, requiring 17.2 seconds to solve. The reason for the large computational time required to solve the penalized duality theory reformulation compared to the other ones is unknown. However, it might be a result of the IPOPT solver having a hard time solving this reformulation for the given initial extracellular metabolite concentrations. The performance of the solver might be improved by changing the initial conditions or the number of finite elements.

The concentration profiles of biomass, glucose and acetate are presented in Figure 17, Figure 18 and Figure 19 respectively.

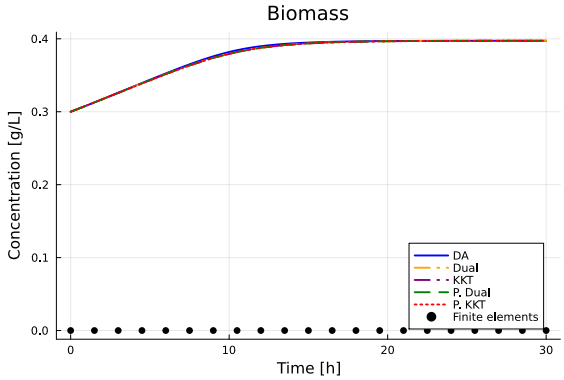


Figure 17: Concentration profile of the biomass for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

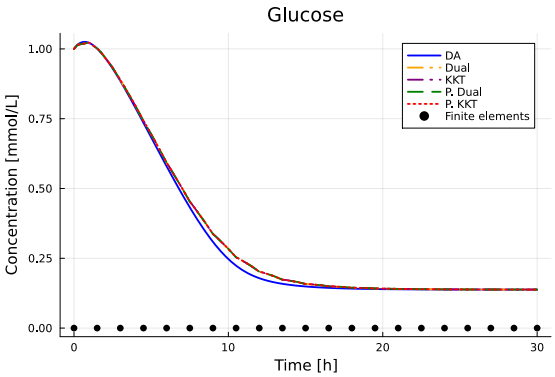


Figure 18: Concentration profile of the glucose for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

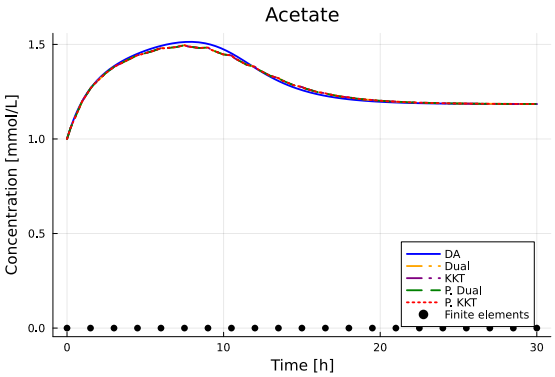


Figure 19: Concentration profile of the acetate for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements and constant metabolic fluxes in each of the finite elements. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Table 15 all the NLPA reformulations are solved successfully and with roughly the same degree of divergence from the DA reformulations. The value of the MSE for the NLPA dFBA model reformulations are very low, MSE of the glucose concentration is roughly $2.01 \cdot 10^{-4}$, MSE of the biomass concentration is $1.24 \cdot 10^{-6}$, and MSE of the acetate

concentration is $1.26 \cdot 10^{-4}$. The small MSE was expected as none of the extracellular metabolite concentrations reached zero, as shown in Figure 17, Figure 18 and Figure 19, where our NLPA reformulations had the largest divergence from the DA reformulation of the dFBA model for batch reactors in Section 4.6.

From Figure 17, Figure 18 and Figure 19 we also see that the different NLPA model reformulations and the DA reformulation stabilize at the same steady state metabolite concentrations. We compare the end concentration found by the dFBA to the steady state model presented in Section 3.4 and see that the steady state concentrations found by the two models are the same. Roughly 0.1379 $mmol/L$ for the glucose concentration, 0.3973 gDW/L for the biomass concentration, and 1.1845 $mmol/L$ for the acetate concentration. However, the NLPA reformulations deviate from the DA when we enter the area where the cells start to consume acetate. This was expected, as our results from the batch model NLPA reformulations also differed from the DA in this area, see Section 4.6. The deviation between the NLPAs and DA might be decreased in this area by introducing the adaptive mesh strategy from Section 4.7 or by increasing the number of finite elements.

4.9 CSTR bioreactor model based on dFBA - adaptive mesh

To decrease the deviation between the NLPA reformulations of the CSTR dFBA model and the DA reformulation of the CSTR model in Section 4.8 it was decided to introduce the adaptive mesh strategy utilized for the batch reactor in Section 4.7. The equidistant element size (h^*) used by the adaptive mesh is set to 1 hour, and the weight used for the adaptive mesh penalization term is set to 0.01 instead of 0.1. The remaining parameters used to initialize the CSTR are the same as those used in Section 4.8 and presented in Table 14.

The solver time, number of times the pFBA where solved in the DA, the solver status, and the MSE between the glucose, biomass and acetate concentration calculated by the NLPA reformulations and the DA are presented in Table 16.

Table 16: Simulation of the CSTR dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the solver time, number of times the DA solved the pFBA, solver status, and the MSE between the glucose, biomass and acetate concentration found by the NLPA reformulations and the DA. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements, constant metabolic fluxes in each of the finite elements, and an adaptive mesh. MIE stands for maximum number of iterations reached, STAL stands for solved to an acceptable level, and SS stands for solved successfully. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Number of calls:	Status:	MSE glucose:	MSE biomass:	MSE acetate:
DA	6.96	418	SS/STAL	0	0	0
Dual	102	-	MIE	$1.48 \cdot 10^{-4}$	$8.71 \cdot 10^{-7}$	$9.50 \cdot 10^{-5}$
P. Dual	14.2	-	SS	$1.33 \cdot 10^{-4}$	$7.76 \cdot 10^{-7}$	$8.50 \cdot 10^{-5}$
KKT	18.2	-	SS	$1.35 \cdot 10^{-4}$	$7.89 \cdot 10^{-7}$	$8.64 \cdot 10^{-5}$
P. KKT	12.1	-	STAL	$1.33 \cdot 10^{-4}$	$7.76 \cdot 10^{-7}$	$8.50 \cdot 10^{-5}$

From Table 16 we see that the non-penalized duality theory NLPA reached the maximum number of iterations, 3000, and thus did not converge to the optimal solutions, such as the other NLPA reformulations. This is reflected in the large computational time required to solve the non-penalized duality theory reformulation and the high MSE value

of the extracellular metabolites compared to the DA reformulation. Where the MSE of the non-penalized duality theory NLPA is slightly higher than the one found by the other approaches. We also see that the two penalized NLPA reformulations were the closest to the DA. It was expected that the non-penalized duality theory reformulation would fail as this is consistent with our previous results for the batch reactor, see Section 4.7.

The quickest NLPA was the penalized KKT reformulation, which only required 12.1 seconds to solve, the second fastest reformulation was the penalized duality theory NLPA, which required 14.2 seconds to solve. From Section 4.7 we expected the time required to solve the penalized reformulations to be relatively close, and faster than the non-penalized reformulations.

Comparing the result from the dFBA model using the adaptive mesh, see Table 16, to the results from the dFBA model without the adaptive mesh, see Table 15, we see that the NLPA model reformulations utilizing the adaptive mesh strategy are closer to the solution of the DA than the NLPA reformulations not utilizing the adaptive mesh strategy. Considering the penalized reformulation that was solved successfully in both cases we see that the MSE of glucose is reduced from $2.01 \cdot 10^{-4}$ to $1.33 \cdot 10^{-4}$, the MSE of biomass is reduced from $1.24 \cdot 10^{-6}$ to $7.76 \cdot 10^{-7}$, and the MSE of acetate is reduced from $1.26 \cdot 10^{-4}$ to $8.50 \cdot 10^{-5}$. On the other hand, we see that the time required to solve the system increased for all the NLPA reformulations, with the exception of the penalized duality theory reformulation which was reduced from 17.2 seconds to 14.2 seconds. The non-penalized duality theory reformulation increased from 4.54 seconds to 102 seconds, the non-penalized KKT reformulation increased from 9.14 to 18.2, and the penalized KKT reformulation increased from 5.24 seconds to 12.1 seconds. The time increase is expected as we have increased the size of the optimization problem. Therefore is the time decrease of the penalized duality theory reformulation unexpected, but this might imply a too large computational time observed when the penalized duality theory reformulation was solved in Section 4.8.

The concentration profiles of biomass, glucose and acetate are presented in Figure 20, Figure 21 and Figure 22 respectively.

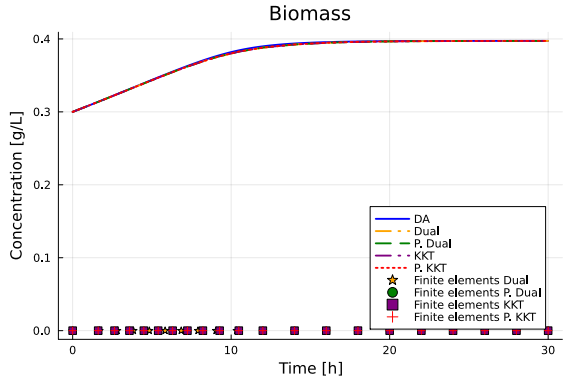


Figure 20: Concentration profile of the biomass for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements, constant metabolic fluxes in each of the finite elements, and an adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

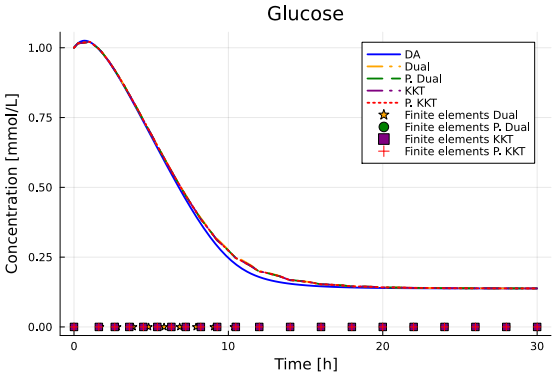


Figure 21: Concentration profile of the glucose for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements, constant metabolic fluxes in each of the finite elements, and an adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

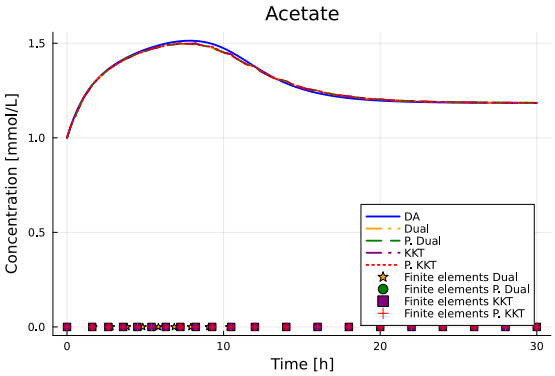


Figure 22: Concentration profile of the acetate for the CSTR bioreactor model based on dFBA, for a case study of the *Escherichia coli* core metabolism. The CSTR bioreactor simulation is run for 30 hours with 30 finite elements, constant metabolic fluxes in each of the finite elements, and an adaptive mesh. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 20, Figure 21 and Figure 22 we see that most of the finite elements are placed in the beginning of the simulation, where we have the largest changes in the extracellular metabolite concentrations. Resulting in a reduction the deviance between the NLPA reformulations and the DA results. However, the NLPA reformulations are still deviating from the DA when the system starts to consume acetate, and it might be necessary to increase the number of finite elements to further improve the results of the NLPA reformulations.

Overall, the accuracy improvement of the NLPA reformulations is very small compared to the increase in time required to solve the optimization problems, and therefore we will not utilize the adaptive mesh strategy in the dFBA CSTR model used for the MPC part of this thesis.

4.10 Tuning of the MPC control parameters

We apply MPC to our CSTR model based on the NLPA dFBA reformulations. Resulting in the optimization problems presented in Section 3.5.

Before we test how the performance of the MPC applied to our dFBA models we tune the control parameters by trial and error. The goal is to find values of the control parameters that result in little to no oscillation and small divergence from the setpoint. We keep the Q control parameter constant and tune the R and C parameters.

The initial conditions for the system used to test the control parameters are the same as those used to initialize the CSTR model in Section 4.8. However, we have changed the

initial dilution rate and extracellular metabolites to the steady state values for the system. The steady state values are found with the steady state model presented in Section 3.4. The initial values used in the simulations are presented in Table 17.

Table 17: Initial conditions used to tune the MPC model based on dFBA for CSTR on a case study of the *Escherichia coli* core metabolism. The parameters are gathered from Oliveira et al. [18]. The Michaelis constant of acetate ($K_{A,M}$) is not given, but we assume that it is equal to the Michaelis constant of glucose ($K_{G,M}$), and the parameters for the dilution rate and the extracellular metabolite concentrations are found with the steady state model presented in Section 3.4.

Parameter:	Symbol:	Value:	Unit:
Size of finite elements	h	1.00	h
Initial glucose concentration	G_0	1.23	mmol/L
Initial biomass concentration	X_0	0.30	gDW/L
Initial acetate concentration	A_0	1.32	mmol/L
Michaelis constant glucose	$K_{G,M}$	0.01	mmol/L
Michaelis constant acetate	$K_{A,M}$	0.01	mmol/L
Maximal glucose uptake	$v_{max,G}$	-10.5	mmol/gDW h
Maximal acetate uptake	$v_{max,A}$	-2.5	mmol/gDW h
Oxygen lower bound	LB_O	-19.0	mmol/gDW h
Dilution rate	D	0.829	h^{-1}
Glucose feed concentration	G_f	5.00	mmol/L
End time	t_{end}	30	h
Number of control actions	M	4	-

We only present the results from the tuning of the penalized duality theory control model, as the behavior is similar for all the NLPA control model reformulations.

4.10.1 R control parameter

First, we tune the R control parameter. The simulation is performed with a controller that changes the setpoint from 0.3 gDW/L to 0.35 gDW/L biomass. The control action of the controller is updated each hour.

The time required to solve one MPC optimization, the status of the solver, and the MSE between the biomass concentration and the setpoint for the penalty duality theory NLPA control model is presented in Table 18.

Table 18: In this table we present the time required to solve one MPC optimization, the solver status, and the MSE of the deviation between the biomass concentration and the setpoint for different values of the R controller parameter in the MPC for a case study of the *Escherichia coli* core metabolism. SS stands for solved successfully, and the time is the average of 30 MPC optimizations.

R-value:	Time: [s]	Status:	MSE biomass:
1.00	0.53	SS	$3.70 \cdot 10^{-4}$
0.10	0.48	SS	$1.79 \cdot 10^{-4}$
0.01	0.81	SS	$1.63 \cdot 10^{-4}$

The biomass concentration, dilution rate, glucose concentration, and acetate concentration profiles are presented in Figure 23, Figure 24, Figure 25 and Figure 26 respectively.

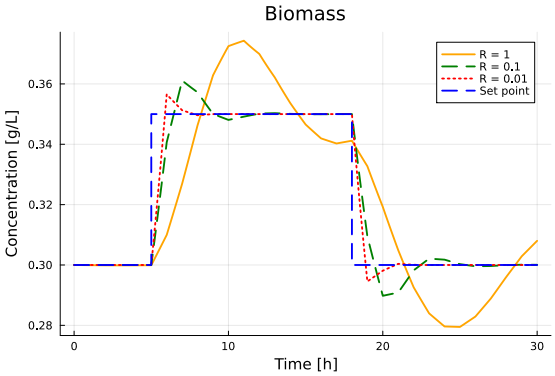


Figure 23: The biomass concentration profile for the tuning of the R control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

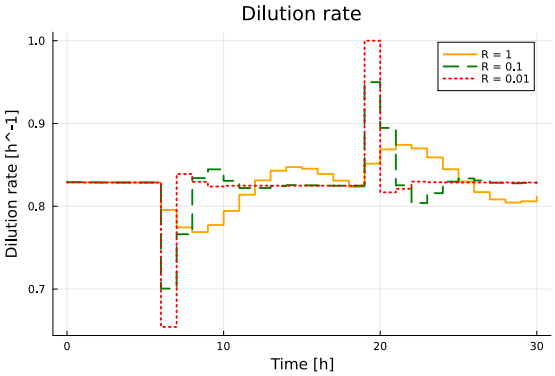


Figure 24: The dilution rate profile for the tuning of the R control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

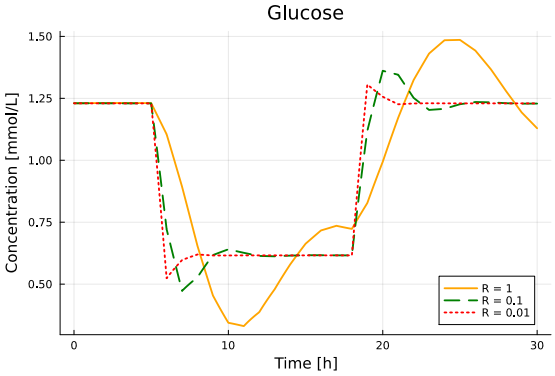


Figure 25: The glucose concentration profile for the tuning of the R control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

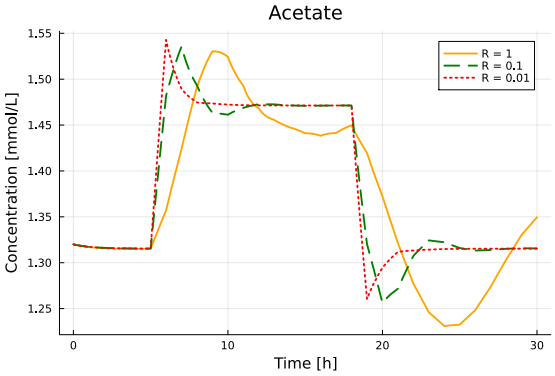


Figure 26: The acetate concentration profile for the tuning of the R control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

From Table 18, Figure 23, Figure 24, Figure 25 and Figure 26 we see that the system oscillates more for large values of the R control parameter, resulting in a larger deviation from the setpoint. This was expected as large R values will penalize the system for large changes in the manipulated variable, the dilution rate (D), compared to the penalization of the deviation from the setpoint. Therefore is it desirable to keep R small compared to the Q control parameter as we wish the system to return to the setpoint quickly.

From Table 18 we also see that the time necessary to solve the system first decreases when we increase the R control parameter, but starts to increase when the value of R keeps decreasing. Implying that decreasing the R value unnecessarily might result in slow control actions. However, as the changes in time overall are quite small, less than 0.4 seconds, and our primary goal is to have as little deviation from the setpoint as possible, it is decided to use very small values of R.

4.10.2 C control parameter

Second, we tune the C control parameter for the penalized dFBA models. This is done by testing the behavior of the MPC utilizing the penalized duality theory NLPA dFBA model with different values of C. The test is performed with a constant setpoint set to 0.3 gDW/L biomass, and the control action is updated each hour.

The time required to solve one MPC optimization, the status of the solver and the MSE between the biomass concentration and the setpoint for the penalty duality theory NLPA control model is presented in Table 19.

Table 19: In this table we present the time required to solve one MPC optimization, the solver status, and the MSE of the deviation between the biomass concentration and the setpoint for different values of the R controller parameter in the MPC for a case study of the *Escherichia coli* core metabolism. SS stands for solved successfully, MIE stands for the maximum number of iterations reached, and the time is the average of 30 MPC optimizations.

C-value:	Time: [s]	Status:	MSE biomass:
1000	1.13	SS	$2.55 \cdot 10^{-11}$
1	0.50	SS	$2.39 \cdot 10^{-16}$
0.001	12.3	MIE	$2.73 \cdot 10^{-9}$

The biomass concentration, dilution rate, glucose concentration, and acetate concentration profiles are presented in Figure 27, Figure 28, Figure 29 and Figure 30 respectively.

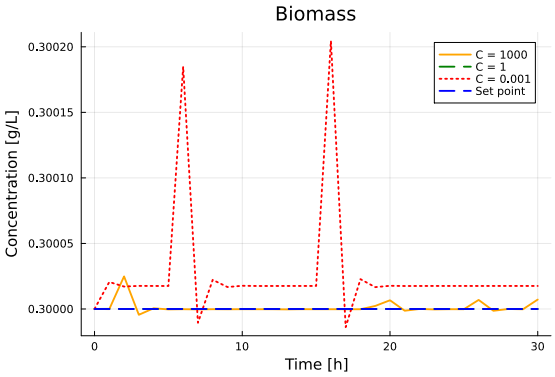


Figure 27: The biomass concentration profile for the tuning of the C control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

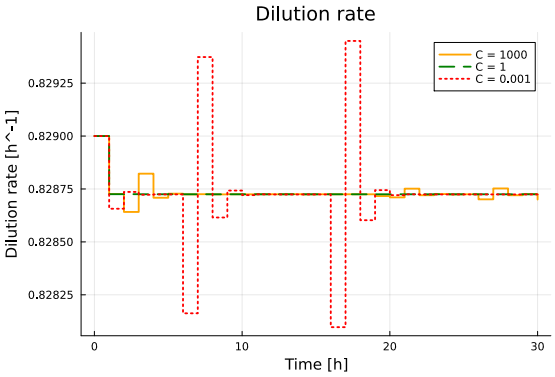


Figure 28: The dilution rate profile for the tuning of the C control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

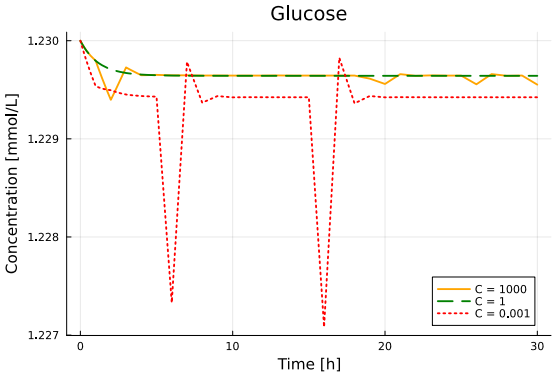


Figure 29: The glucose concentration profile for the tuning of the C control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

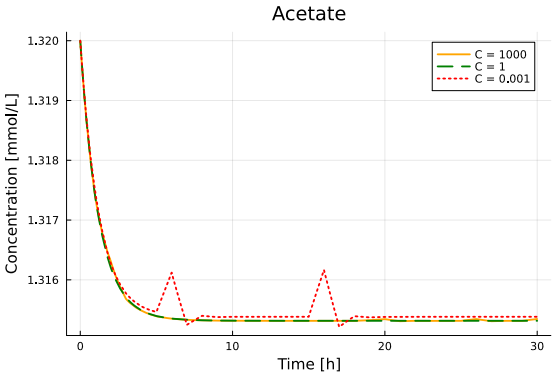


Figure 30: The acetate concentration profile for the tuning of the C control parameter in the MPC for a case study of the *Escherichia coli* core metabolism. The model uses the penalized duality NLPA dFBA reformulation for a CSTR.

From Table 19 we see that the control model fails to converge if the value of the C control parameter is too low, this is apparent as the reformulations utilizing the lowest C-value reach the maximum number of iterations, thus not converging properly. Furthermore, from Figure 27, Figure 28, Figure 29 and Figure 30 we see that low C-values results in sudden spikes and a slight increase in the biomass concentration above the setpoint This was expected as the C control parameter is used to enforce the optimality conditions of the original pFBA problem for the dFBA model.

On the other hand is the deviation between the biomass concentration and the setpoint increasing if the C control parameter is too high and the time required to solve a control action increases. High values of the C control parameter also result in sudden spikes in the concentration profiles of biomass and glucose. This is expected as too large C-values result in the enforcement of the optimality conditions being prioritized over the necessary control actions.

4.11 MPC changing the setpoint

We apply the tuned MPC to our dFBA model for the CSTR and test the controller's ability to handle changes in the setpoint of biomass. The value of the control parameters is presented in Table 20. As we prioritize to stay close to the setpoint it was decided to use a very small value, 10^{-10}, for the R control parameter. The C control parameter and maximal change in the dilution rate (Δu_{max}) where found by trial and error, where the goal was to avoid unsuccessful MPC optimizations.

Table 20: Tuned control parameters for the MPC used to test the MPC model based on dFBA for CSTR for a case study of the *Escherichia coli* core metabolism.

Parameter	P.dual	P. KKT	Dual/KKT
Q	1.0	1.0	1.0
R	10^{-10}	10^{-10}	10^{-10}
C	10	10^{-2}	-
Δu_{max}	0.2	0.2	0.2

We change the setpoint of the biomass concentration from 0.2 gDW/L to 0.4 gDW/L and keep all the other parameters constant. The initial conditions of the test are the same as those used to tune the control parameters in Section 4.10, but the dilution rate and the extracellular metabolite concentrations are changed to the steady state conditions for biomass concentration at 0.2 gDW/L. The control actions are updated every hour and the controller solves the system for four control actions each time. The initial values are presented in Table 21.

Table 21: Initial conditions used to test the MPCs ability to handle changes in the setpoint. The parameters are gathered from Oliveira et al. [18]. The Michaelis constant of acetate ($K_{A,M}$) is not given, but we assume that it is equal to the Michaelis constant of glucose ($K_{G,M}$), and the parameters for the dilution rate and the extracellular metabolite concentrations are found with the steady-state model presented in Section 3.4.

Parameter:	Symbol:	Value:	Unit:
Size of finite element	h	1.00	h
Initial dilution rate	D_0	0.831	h^{-1}
Michaelis constant glucose	K_G	0.01	mmol/L
Michaelis constant acetate	K_A	0.01	mmol/L
Maximal glucose uptake	$v_{G,max}$	-10.5	mmol/gDW h
Maximal acetate uptake	$v_{A,max}$	-2.5	mmol/gDW h
Lower bound oxygen	LB_O	-19.0	mmol/gDW h
Initial biomass concentration	$x_{X,0}$	0.200	gDW/L
Initial glucose concentration	$x_{G,0}$	2.480	mmol/L
Initial acetate concentration	$x_{A,0}$	0.895	mmol/L
Glucose feed concentration	G_f	5	mmol/L
Number of control actions	M	4	-
End time	t_{end}	35	h

The time required to solve one MPC optimization, the number of times the MPC optimization failed and the MSE between the biomass concentration and the setpoint are presented in Table 22.

Table 22: Simulation of the MPC based on the dFBA model for a CSTR on a case study of the *Escherichia coli* core metabolism. In this table we present the time required to solve one MPC optimization, the solver status, and the MSE of the deviation between the biomass concentration and the setpoint. The setpoint for the biomass concentration was increased from 0.2 gDW/L to 0.4 gDW/L. The time is the average of 30 MPC optimizations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Solver failures:	MSE biomass:
Dual	8.67	19	$1.01 \cdot 10^{-2}$
P. Dual	0.96	1	$3.79 \cdot 10^{-3}$
KKT	8.80	25	$1.03 \cdot 10^{-2}$
P. KKT	3.14	1	$3.79 \cdot 10^{-3}$

From Table 22 we see that the controllers using the non-penalized NLPA reformulations have many unsuccessfully solved MPC optimizations. The non-penalized KKT reformulation fails 25 of 30 times and the non-penalized duality theory reformulation fails 19 of 30 times. The penalized NLPA reformulations failed less often, with only one failed MPC optimization each. It was expected that the non-penalized reformulations would fail more

often than the penalized ones as the non-penalized reformulations have more hard constraints than the penalized ones, and too many hard constraints may lead to convergence problems for MPC controllers [26].

The penalized NLPA reformulations have the same degree of divergence from the setpoint of biomass, see the small MSE. However, the penalized duality theory NLPA reformulation, requiring on average of 0.96 seconds to solve one MPC optimization, is much faster than the penalized KKT NLPA reformulation, requiring an average of 3.14 seconds to solve one MPC optimization. This was unexpected as the penalized KKT reformulation was computationally faster than the penalized duality theory reformulation when we tested the dFBA models for a CSTR in Section 4.8. As the only difference between the two penalized reformulations is the $\Phi(\mathbf{v}, \mathbf{LB}, \mathbf{UB})$ term in the objective function and the values of the control parameters, which are a part of the objective, must the difference in solver time stem from how the IPOPT-solver handles the objective functions. The performance of the penalized KKT reformulation might be improved by adjusting the value control parameters.

The biomass concentration, dilution rate, glucose concentration, and acetate concentration profiles are presented in Figure 31, Figure 32, Figure 33 and Figure 34 respectively.

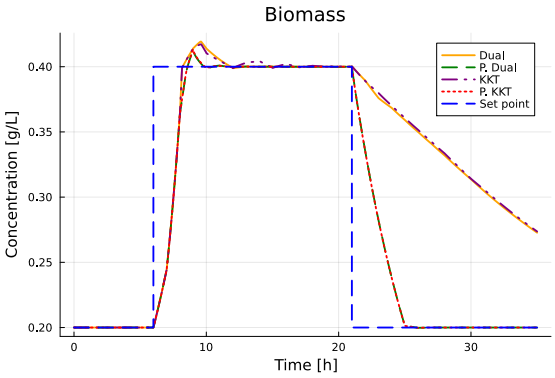


Figure 31: The biomass concentration profile from the MPC test with changing setpoint for a case study of the *Escherichia coli* core metabolism. The setpoint for the biomass concentration was increased from 0.2 gDW/L to 0.4 gDW/L. The time is the average of 30 MPC optimizations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

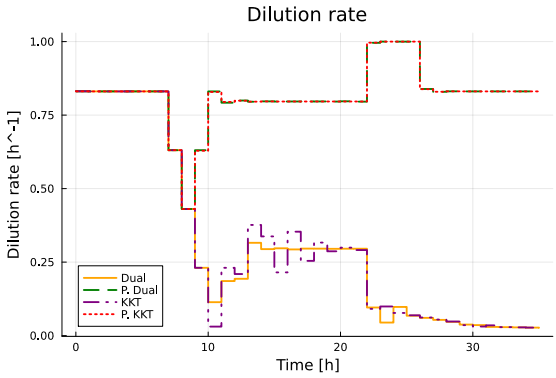


Figure 32: The dilution rate profile from the MPC test with changing setpoint for a case study of the *Escherichia coli* core metabolism. The setpoint for the biomass concentration was increased from 0.2 gDW/L to 0.4 gDW/L. The time is the average of 30 MPC optimizations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

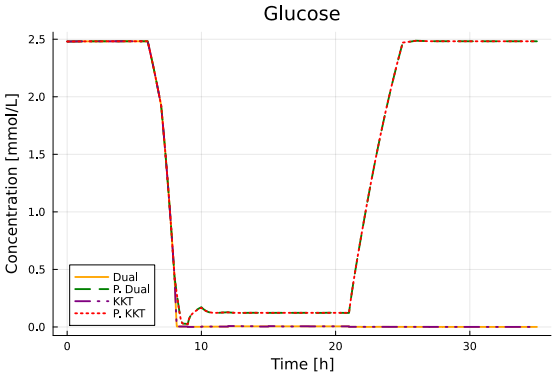


Figure 33: The glucose concentration profile from the MPC test with changing setpoint for a case study of the *Escherichia coli* core metabolism. The setpoint for the biomass concentration was increased from 0.2 gDW/L to 0.4 gDW/L. The time is the average of 30 MPC optimizations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

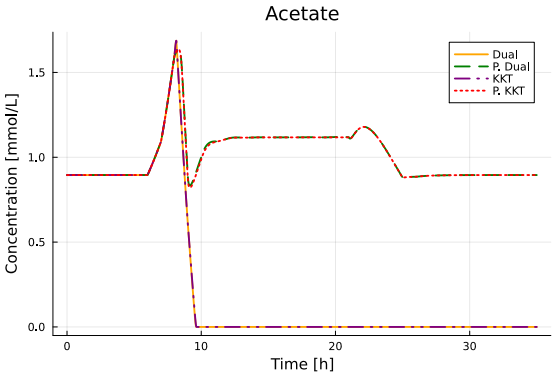


Figure 34: The acetate concentration profile from the MPC test with changing setpoint for a case study of the *Escherichia coli* core metabolism. The setpoint for the biomass concentration was increased from 0.2 gDW/L to 0.4 gDW/L. The time is the average of 30 MPC optimizations. Dual refers to the duality theory reformulation of the pFBA, P. Dual refers to the penalized duality theory reformulation of the pFBA, KKT refers to the KKT reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 31, Figure 32, Figure 33 and Figure 34 we see that the MPC models utilizing the non-penalized NLPA reformulations fail to return to steady state when we increase the setpoint. While the glucose concentration of the penalized reformulations stops decreasing right before it reaches zero, do the concentration from the non-penalized reformulations keep decreasing until they reaches and stay at zero. As our model reformulations are least accurate in the area where the substrate concentrations approaches zero, was it expected that the non-penalized model reformulations would fail in this area.

From Figure 31, Figure 32, Figure 33 and Figure 34 we also see that the MPC utilizing the penalized NLPA reformulation stays close to the setpoint, without much oscillation when the setpoint changes. This was expected as we used very small values of the R control parameter for all the MPC models.

From the small changes in the steady state of the dilution rate when we change the setpoint, it is apparent the system is very sensitive to changes in the manipulated variable. This was expected as we are working with very low concentrations of extracellular metabolites.

4.12 MPC with disturbance in the glucose feed concentration

Second, we test the MPCs ability to handle disturbances in the glucose feed concentration. We change the glucose feed from 5.0 mmol/L to 8.0 mmol/L and keep all the other parameter constant. We use the same initial conditions as in Section 4.11.

The time required to solve one MPC optimization, how many times the solver failed to converge, and the MSE between the biomass concentration and the setpoint are presented

in Table 23.

Table 23: Simulation of the MPC based on the dFBA model for a CSTR on a case study of the *Escherichia coli* core metabolism. In this table we present the time required to solve one MPC optimization, the solver status, and the MSE of the deviation between the biomass concentration and the setpoint. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method:	Time: [s]	Solver failures:	MSE biomass:
P. Dual	0.95	0	$1.51 \cdot 10^{-10}$
P. KKT	2.29	0	$1.49 \cdot 10^{-10}$

Both the penalized duality theory reformulation and the penalized KKT reformulation were solved successfully at all the MPC optimizations, see Table 23, and the difference in MSE between the two methods are neglectable, less than $1.0 \cdot 10^{-10}$. However, the penalized KKT approach requires much more time to solve one MPC optimization, 2.29 seconds, than the penalized duality theory approach, 0.95 seconds. This is consistent with our results from the changing setpoint test in Section 4.11.

The biomass concentration, dilution rate, glucose concentration, acetate concentration, and glucose feed concentration profiles are presented in Figure 35, Figure 36, Figure 37, Figure 38, and Figure 39 respectively.

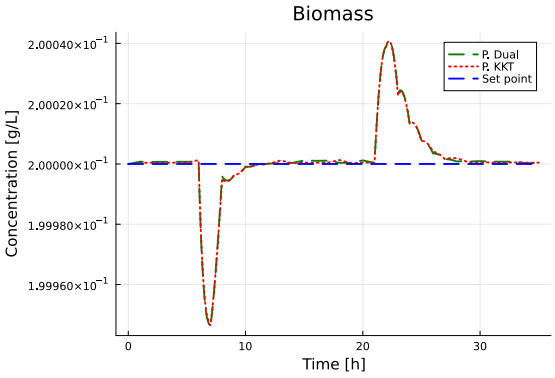


Figure 35: The biomass concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

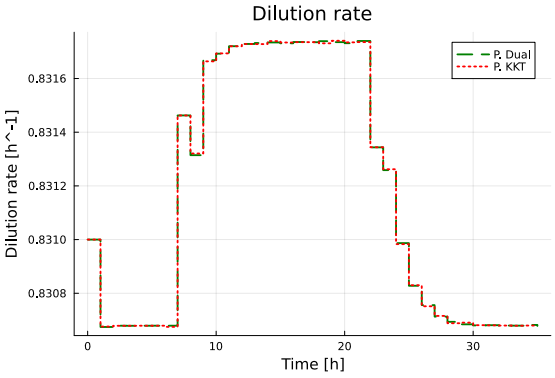


Figure 36: The dilution rate profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

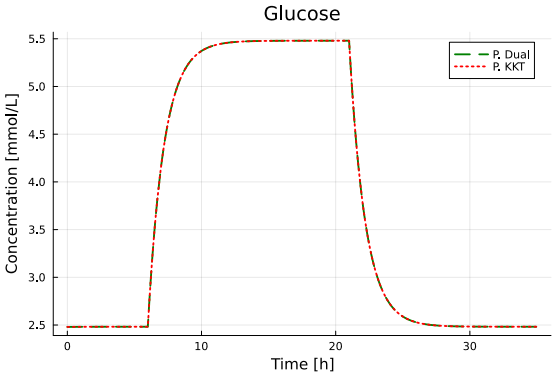


Figure 37: The glucose concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

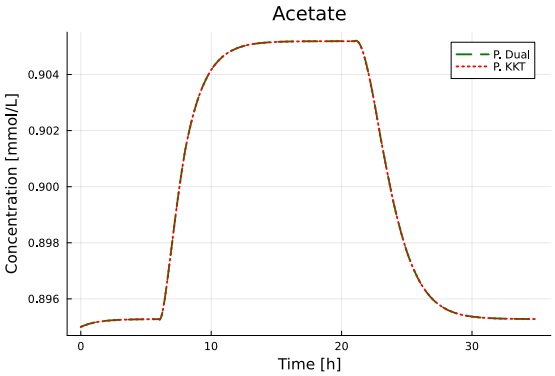


Figure 38: The acetate concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

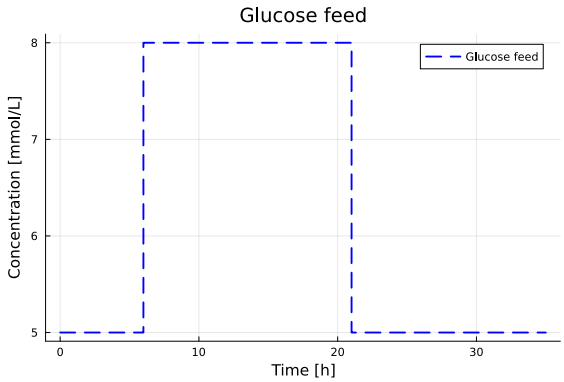


Figure 39: The glucose feed concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The glucose feed concentration was increased from 5.0 mmol/L to 8.0 mmol/L. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 35, Figure 36, Figure 37, Figure 38, and Figure 39, we see that the controller performs well. The deviation in the biomass concentration from the setpoint is very small. However, the behavior of the MPC controllers is quite interesting. Right after the glucose feed is increased the controllers increase the dilution rate, to lower the biomass concentration. Thereafter the controllers lower the dilution rate, allowing the biomass concentration to increase, before once more increasing the dilution rate to its steady state value. The biomass is then slowly rising back up to the setpoint. This behavior was expected as the biomass growth rate is dependent on the biomass concentration. Therefore, by washing out some of the biomass at the initial increase of the glucose feed concentration, the MPC controllers avoid a rapid increase of the biomass concentration that could have resulted in a large overshoot of biomass, that probably would have required longer time to wash out.

4.13 MPC with disturbance in maximal glucose uptake

Finally, we test the MPCs ability to handle disturbances in the maximal glucose uptake rate. We change the maximal glucose uptake from -10.5 $mmol/L$ to -5.25 $mmol/L$ and keep all the remaining parameters constant. We use the same initial conditions as in Section 4.11.

The time required to solve one MPC optimization, the status of the solver, and the MSE between the biomass concentration and the setpoint are presented in Table 24.

Table 24: Simulation of the MPC based on the dFBA model for a CSTR on a case study of the *Escherichia coli* core metabolism. In this table we present the time required to solve one MPC optimization, the solver status, and the MSE of the deviation between the biomass concentration and the setpoint. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

Method	Time [s]	Solver failures:	MSE X
P. Dual	0.76	0	$1.34 \cdot 10^{-4}$
P. KKT	2.92	1	$1.34 \cdot 10^{-4}$

The penalized KKT reformulation fails to solve one of the MPC optimizations, see Table 24. However, comparing the MSE values of the penalized duality theory and KKT reformulations show that one failure did not result in a noticeable difference in the results from the two approaches. The penalized duality theory reformulation is much faster than the penalized KKT reformulation, which requires 0.76 seconds to solve one MPC optimization compared to the KKT, which requires 2.92 seconds to solve one MPC optimization. The time difference is consistent with our previous results for the controllers, and the one failed optimization for the KKT reformulation might be contributing to the larger KKT solver time.

The biomass concentration, dilution rate, glucose concentration, acetate concentration, and the maximum glucose uptake profiles are presented in Figure 40, Figure 41, Figure 42, Figure 43, and Figure 44 respectively.

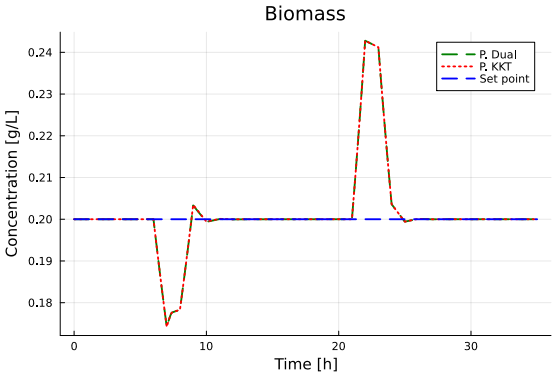


Figure 40: The biomass concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

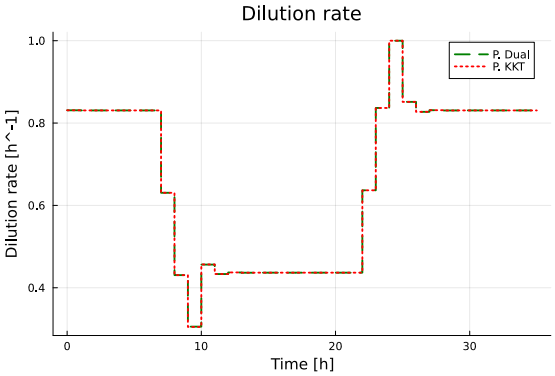


Figure 41: The dilution rate profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

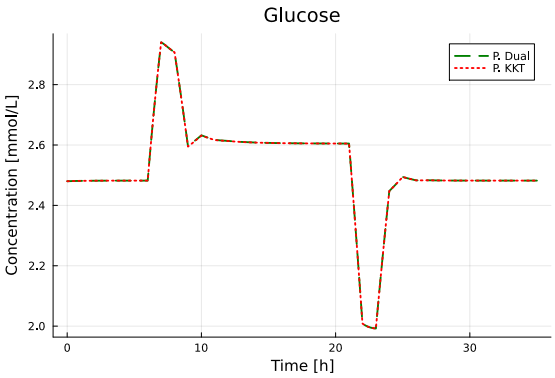


Figure 42: The glucose concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

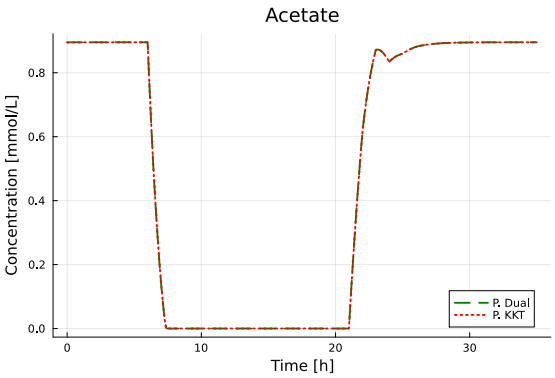


Figure 43: The acetate concentration profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

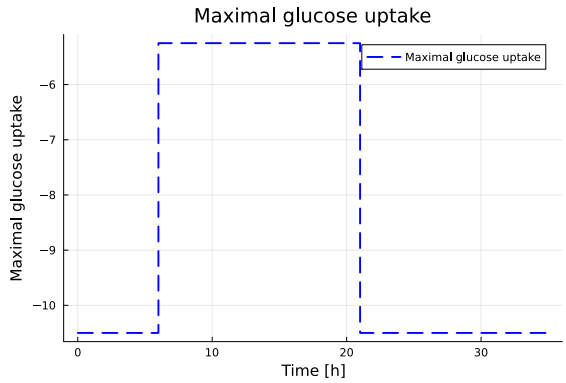


Figure 44: The maximal glucose uptake profile from the MPC test with disturbance in glucose feed concentration for a case study of the *Escherichia coli* core metabolism. The maximal glucose uptake was changed from -10.5 $mmol/L$ to -5.25 $mmol/L$. The time is the average of 30 MPC optimizations. P. Dual refers to the penalized duality theory reformulation of the pFBA, and P. KKT refers to the penalized KKT reformulation of the pFBA.

From Figure 40, Figure 41, Figure 42, Figure 43, and Figure 44 we see that the controller preforms well. The deviation in the biomass concentration from the setpoint is small. This was expected from the low MSE values in Table 24. However, we observe a slight oscillation in the glucose and biomass concentrations when the absolute value of the maximal glucose uptake is decreased, and a slight oscillation in the acetate concentration when the absolute value of the maximal glucose uptake is increased. The controllers behave as expected, lowering the dilution rate when the absolute value of the maximal glucose uptake is decreased, to wash out less of the biomass, and increasing the dilution rate when the absolute value of the maximal glucose uptake is increased.

Overall, the MPC utilizing the penalized duality theory reformulation of the pFBA is much faster than the MPC utilizing the penalized KKT reformulations. It is also slightly more robust, returning fewer failed MPC optimizations than the KKT reformulation. This is a bit surprising as the penalized KKT performed better than the penalized duality theory reformulations for the test of the CSTR model implementation, Section 4.8. This might be attributed to the value of the control parameters, or how the IPOPT solver treats the expanded objective functions.

5 Conclusion

The duality theory reformulation of the Parsimonious Flux Balance Analysis (pFBA) is presented as an alternative to the more commonly used Karush–Kuhn–Tucker (KKT) optimality condition reformulations. It was found that the non-penalized reformulations are faster to solve than their penalized versions, and that the duality theory reformulations are faster than the KKT reformulations. The duality theory reformulations are also closer to the results of the original pFBA problem than the KKT reformulations, and the non-penalized reformulations are closer than the penalized ones. It was also found that the KKT reformulation needs a lower acceptable tolerance level and constraint violation tolerance to compete with the performance of the duality theory reformulations.

When developing the Dynamic Flux Balance Analysis (dFBA) model for the batch reactor it was found that large changes in the metabolic fluxes between the collocation points inside a finite element utilized by the Non-Linear Programming Approach (NLPA) may lead to convergence problems. These problems may arise when the systems changes from producing to consuming one of the extracellular metabolites.

The problems arising from rapidly changing fluxes inside the finite elements can be avoided by assuming constant metabolic fluxes inside each finite element. An advantage of this assumption is that we only need to calculate the fluxes for one of the collocation points inside each of the finite elements, thus reducing the size of our optimization problems and the time required to solve them. It was found that the concentration in the last collocation point must be used to calculate the metabolic fluxes, as using any of the previous points would result in an overshoot. A drawback of this assumption is that the metabolite concentrations may not reach zero, resulting in small deviations in the areas where one of the extracellular metabolites are consumed. This problem is also present when the Direct Approach (DA) is used to solve our dFBA problem, but is less pronounced as the ordinary differential equation (ODE) solver utilized by the DA uses a lot more points than the NLPA reformulations.

It was found that the deviation between the DA and NLPA reformulations can be reduced by introducing an adaptive mesh strategy to our NLPA reformulations of the dFBA. By strategically placing the finite elements, to avoid having the transition from producing to consuming one of the extracellular metabolites inside one of the elements, are the deviation between the DA and NLPA reformulations for the batch reactor greatly reduced. However, the improvement form utilizing the adaptive mesh strategy is less pronounced for the continuous stirred tank reactor (CSTR) models. A drawback of the adaptive mesh strategy is that it increases the size of the optimization problems, thus also increasing the required solver time.

From our simulations of the models for batch and CSTR bioreactors based on dFBA solved with the NLPA reformulations it was found that the non-penalized NLPA reformulations overall are less reliable than the penalized NLPA reformulations. The IPOPT solver was more likely to fail for the non-penalized reformulations than the penalized ones. The penalized reformulations of the dFBA are also in general more efficient than the non-penalized reformulations, requiring less time to solve. The penalized KKT reformulation was overall the fastest of the NLPA reformulations, solving faster or as fast as the duality theory reformulation. The difference in deviation from the DA between the NLPA reformulations are neglectable.

We have shown that Model Predictive Control (MPC) can be applied to CSTR bioreactors based on dFBA. The MPC controller performed well, keeping the biomass concentration close to the desired setpoint. It was found that the MPC can handle relatively large changes in the setpoint, disturbances in the glucose feed, and disturbances in the maximal glucose uptake. Overall, the penalized duality theory reformulation was much faster than the penalized KKT reformulation and more reliable when solved with the IPOPT solver. However, it should be mentioned that this may be attributed to the weighting of the control parameters used in the two different MPC model reformulations.

6 Future Work

For future work, the challenges associated with expanding the application of our models developed in this paper to the genome scale *Escherichia coli* metabolic network must be investigated. Larger metabolic networks will increase the size of the optimization problems and we therefore expect the computational time required to solve a MPC optimization to increase. We need to test whether our models can be solved in a timely manner for commercially used bioprocesses. This also holds true for the genome scale metabolic network of other commercially used organisms. A larger metabolic network would also presumably give larger differences between the penalized duality theory and penalized KKT reformulations, making potential differences between the methods more reliable.

The dFBA and MPC models should also be applied to other commonly used bioreactor designs, such as the fed-batch bioreactor. Our models had a harder time handling batch processes as the concentration of the substrates reached zero, than handling the CSTR where the concentrations of the extracellular metabolites for the most part stay above zero. Therefore, we expect our MPC models to have a harder time handling fed-batch than CSTR bioreactors, and it might be necessary to introduce adaptive mesh for the MPC models. This will likely increase the computationally time required to solve the system, and more efficient adaptive mesh strategies should therefore be explored.

From our current results is it difficult to say with certainty whether the observed differences between the penalized duality theory and penalized KKT MPC reformulations stem from the pFBA reformulations or the differences in control parameters used in the MPC simulations. The MPC models should therefore also be tested for larger ranges of control parameter values to test the effect of the control parameters on our observed results.

References

[1] BT Baumrucker, Jeffrey G Renfro, and Lorenz T Biegler. "MPEC problem formulations and solution strategies with chemical engineering applications". In: *Computers & Chemical Engineering* 32.12 (2008), pp. 2903–2913.

[2] Lorenz T Biegler, Arturo M Cervantes, and Andreas Wächter. "Advances in simultaneous strategies for dynamic process optimization". In: *Chemical engineering science* 57.4 (2002), pp. 575–593.

[3] Christophe Chassagnole et al. "Dynamic modeling of the central carbon metabolism of *Escherichia coli*". In: *Biotechnology and bioengineering* 79.1 (2002), pp. 53–73.

[4] I. Chorkendorff and J. W. Niemantsverdriet. *Concepts of Modern Catalysis and Kinetics*. 3rd ed. Wiley-VCH, 2017. ISBN: 978-3-527-33268-7.

[5] Pauline M. Doran. *Bioprocess Engineering Principles*. 2nd ed. Academic Press, 2013. ISBN: 978-0-12-220851-5.

[6] Jeremy S Edwards, Rafael U Ibarra, and Bernhard O Palsson. "In silico predictions of *Escherichia coli* metabolic capabilities are consistent with experimental data". In: *Nature biotechnology* 19.2 (2001), pp. 125–130.

[7] Abhishek Gupta, Jacek Mańdziuk, and Yew-Soon Ong. "Evolutionary multitasking in bi-level optimization". In: *Complex & Intelligent Systems* 1 (2015), pp. 83–95.

[8] John D Hedengren et al. "Nonlinear modeling, estimation and predictive control in APMonitor". In: *Computers & Chemical Engineering* 70 (2014), pp. 133–148.

[9] Laurent Heirendt et al. "Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v. 3.0". In: *Nature protocols* 14.3 (2019), pp. 639–702.

[10] Banafsheh Jabarivelisdeh et al. "Adaptive predictive control of bioprocesses with constraint-based modeling and estimation". In: *Computers Chemical Engineering* 135 (2020), p. 106744. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.106744. URL: https://www.sciencedirect.com/science/article/pii/S0098135419308142.

[11] Zachary A King et al. "BiGG Models: A platform for integrating, standardizing and sharing genome-scale models". In: *Nucleic acids research* 44.D1 (2016), pp. D515–D522.

[12] Nathan E Lewis et al. "Omic data from evolved *E. coli* are consistent with computed optimal growth from genome-scale models". In: *Molecular systems biology* 6.1 (2010), p. 390.

[13] Sven Leyffer, Gabriel López-Calva, and Jorge Nocedal. "Interior methods for mathematical programs with complementarity constraints". In: *SIAM Journal on Optimization* 17.1 (2006), pp. 52–77.

[14] Daniel Machado and Markus Herrgård. "Systematic evaluation of methods for integration of transcriptomic data into constraint-based models of metabolism". In: *PLoS computational biology* 10.4 (2014), e1003580.

[15] Radhakrishnan Mahadevan, Jeremy S Edwards, and Francis J Doyle. "Dynamic flux balance analysis of diauxic growth in *Escherichia coli*". In: *Biophysical journal* 83.3 (2002), pp. 1331–1340.

[16] Caroline S.M. Nakama and Johannes Jäschke. "Analysis of control models based on dFBA for fed-batch bioreactors solved by interior-point methods". In: *IFAC-PapersOnLine* 55.7 (2022). 13th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems DYCOPS 2022, pp. 131–136. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2022.07.433. URL: https://www.sciencedirect.com/science/article/pii/S2405896322008345.

[17] Rafael D de Oliveira. *DC_dFBA*. January 2, 2023. URL: https://github.com/LMSE/DC_dFBA/tree/main/Ecoli_iJO1366.

[18] Rafael D de Oliveira, Galo AC Le Roux, and Radhakrishnan Mahadevan. "Nonlinear programming reformulation of dynamic flux balance analysis models". In: *Computers & Chemical Engineering* 170 (2023), p. 108101.

[19] Jeffrey D Orth, Ines Thiele, and Bernhard Ø Palsson. "What is flux balance analysis?" In: *Nature biotechnology* 28.3 (2010), pp. 245–248.

[20] Jeffrey D. Orth, R. M. T. Fleming, and Bernhard Ø. Palsson. "Reconstruction and Use of Microbial Metabolic Networks: the Core *Escherichia coli* Metabolic Model as an Educational Guide". In: *EcoSal Plus* 4.1 (2010). DOI: 10.1128/ecosalplus.10.2.1. eprint: https://journals.asm.org/doi/pdf/10.1128/ecosalplus.10.2.1. URL: https://journals.asm.org/doi/abs/10.1128/ecosalplus.10.2.1.

[21] Chandra Shekhar Pareek, Rafal Smoczynski, and Andrzej Tretyn. "Sequencing technologies and genome sequencing". In: *Journal of applied genetics* 52 (2011), pp. 413–435.

[22] Tobias Ploch et al. "Simulation of differential-algebraic equation systems with optimization criteria embedded in Modelica". In: *Computers Chemical Engineering* 140 (2020), p. 106920. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.106920. URL: https://www.sciencedirect.com/science/article/pii/S0098135419313195.

[23] S Joe Qin and Thomas A Badgwell. "A survey of industrial model predictive control technology". In: *Control engineering practice* 11.7 (2003), pp. 733–764.

[24] Christopher Rackauckas and Qing Nie. "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of Open Research Software* 5.1 (2017).

[25] Science and Technology Facilities Council (STFC). *A collection of Fortran codes for large scale scientific computation.* URL: https://www.hsl.rl.ac.uk/index.html (visited on July 6, 2023).

[26] Dale E Seborg et al. *Process Dynamics and Control*. 4th ed. John Wiley & Sons, 2016. ISBN: 978-1-119-28591-5.

[27] Lieven Vandenberghe Stephen Boyd. *Convex Optimization*. Vol. 7. Cambridge university press, 2009.

[28] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106 (2006), pp. 25–57.

[29] Wei Wan and Lorenz T Biegler. "Structured regularization for barrier NLP solvers". In: *Computational Optimization and Applications* 66 (2017), pp. 401–424.

[30] Ali R Zomorrodi and Costas D Maranas. *Optimization methods in metabolic networks*. John Wiley & Sons, 2016.

A Lower bounds

A.1 Lower bounds for batch bioreactor model based on dFBA

The lower bound of the glucose and acetate found by the penalized duality theory reformulation in Section 4.4 are presented in Table 25. The lower bounds are given for each point in a finite element, and three finite elements are presented.

Table 25: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the lower bounds of the metabolic reactions found by the penalized duality theory reformulation for some of the finite elements. The batch bioreactor simulation is run for 5.33 hours with 6 finite elements. Point 1 refers to the first point in a finite element, Point 2 refers to the second point in a finite element, and Point 3 refers to the third point in a finite element.

Metabolite:	Time: [s]	Concentration: [mmol/L]	Point 1:	Point 2:	Point 3:
Glucose	0.14 - 0.89	10.48 - 10.36	−10.49	−10.49	−10.49
	1.91 - 2.67	10.01 - 9.47	−10.49	−10.49	−10.49
	4.58 - 5.33	4.95 - 0.09	−10.48	−10.46	−9.46
Acetate	0.14 - 0.89	0.11 - 0.15	−2.28	−2.32	−2.34
	1.91 - 2.67	0.28 - 0.47	−2.41	−2.44	−2.45
	4.58 - 5.33	2.07 - 3.70	−2.48	−2.49	−4.49

As expected from the Michaelis-Menten kinetics (MMK) used for the substrate uptake, Equation 10, are the lower bound of glucose not changing until the glucose concentration becomes very low, see Table 25. We can see the same behavior for the acetate as the change in the acetate lower bound stabilizes as the acetate concentration increases.

A.2 Lower bounds for batch bioreactor model based on dFBA - increased time span

The lower bound of the substrates found by the penalized duality theory reformulation in Section 4.5 are presented in Table 26.

Table 26: Simulation of the batch dFBA model on a case study of the *Escherichia coli* core metabolism. In this table we present the lower bounds of the metabolic reactions found by the penalized duality theory reformulation for some of the finite elements. The batch bioreactor simulation is run for 7.1 hours with 10 finite elements. Point 1 refers to the first point in a finite element, Point 2 refers to the second point in a finite element, and Point 3 refers to the third point in a finite element.

Metabolite:	Time: [s]	Concentration: [mmol/L]	Point 1:	Point 2:	Point 3:
Glucose	0.11 - 0.71	10.48 - 10.43	−10.49	−10.49	−10.49
	2.95 - 3.55	9.86 - 9.59	−10.49	−10.49	−10.49
	5.08 - 5.68	7.70 - 6.37	−10.49	−10.49	−10.48
	6.50 - 7.10	2.64 - 0.00	−10.46	−10.33	$-1.03 \cdot 10^{-4}$
Acetate	0.11 - 0.71	0.11 - 0.11	−2.29	−2.29	−2.30
	2.95 - 3.55	0.20 - 0.18	−2.38	−2.36	−2.37
	5.08 - 5.68	0.45 - 0.35	−2.45	−2.43	−2.43
	6.50 - 7.10	0.79 - 0.03	−2.47	−2.46	−1.97

From Table 26 we see that the lower bounds behave as expected with changes in the glucose and acetate concentrations. The lower bound of glucose is not changing until the glucose concentration becomes very low. The lower bound of acetate is increasing and decreasing, but this is consistent with the rapid changes we see in the acetate concentration, Figure 9.

B Project code

In this section we present some of the code scripts used in this project. For the code in its entirety please see the Process-Optimization-and-Control GitHub, `https://github.com/Process-Optimization-and-Control`.

B.1 Julia packages

This project used julia versions 1.8.5 with the packages presented below.

- Blink v0.12.6
- DifferentialEquations v7.7.0
- Distributions v0.25.87
- Ipopt v1.2.1
- JuMP v1.10.0
- PlotlyJS v0.18.10
- Plots v1.38.9
- TimerOutputs v0.5.22
- Trebuchet v0.2.2

The IPOPT solver used the MA97 linear solver from HSL [25].

B.2 *Escherichia coli* core model

The code presented below gathers the stoichiometric matrix (**S**), the c-vector (**c**), the lower bounds (**LB**) and the upper bounds (**UB**) of the metabolic reactions, the reaction names (**Rxnames**), and the metabolic fluxes (**v**) of the *Escherichia coli* core metabolic model and stores them in a structure. The function *printFluxes* prints all the fluxes sufficiently, 10^{-5}, different from zero, and the *changeRxnBounds* function changes the lower or upper bounds for a given metabolic reaction.

```
include("init.jl")

mutable struct E_coli_Model

    S::Matrix
    c::Vector
    LB::Vector
    UB::Vector
    Rxnames::Vector
    v::Vector

    function E_coli_Model(S::Matrix, c::Vector, LB::Vector, UB::Vector,
    ↪  Rxnames::Vector)
```

```
        v = zeros(size(S)[2])
        new(S, c, LB, UB, Rxnames, v)
    end
end

# Print fluxes
# ------------------------------------------------------------------------
function printFluxes(model::Any, print_all::Any)
    println("Fluxes:")
    println("-----------------------------------------------")
    for i = 1:length(model.v)
        if print_all == false
            if (abs(model.v[i]) > 1e-5)
                println("$(model.Rxnames[i]): $(model.v[i])")
            end
        else
            println("$(model.Rxnames[i]): $(model.v[i])")
        end
    end
end

# Change reaction bounds
# ------------------------------------------------------------------------
function changeRxnBounds(model::Any, rec_target::String, val_target::Any,
↪  bound_target::String)
    for i = 1:length(model.v)
        if model.Rxnames[i] == rec_target
            if bound_target == "l"
                model.LB[i] = val_target
            elseif bound_target == "u"
                model.UB[i] = val_target
            else
                println("l for lower bound and u for upper bound")
            end
            return nothing
        end
    end
end

E_coli_Model_init = E_coli_Model(init_mod[:S], init_mod[:c], init_mod[:LB],
↪  init_mod[:UB], init_mod[:Rxnames])
```

B.3 Batch models with adaptive mesh

In this subsection we present some the code used to simulate the batch process with adaptive mesh.

main.jl

main.jl is used to initialize the system, run the batch simulation, and to plot the results.

```
# include("dFBA Batch/main adaptive mesh.jl")

import Ipopt, JuMP, Plots, TimerOutputs, DifferentialEquations
```

```
include("E. coli core/E_coli_Model.jl")
include("init.jl")
include("DAP/DAP.jl")
include("DAP/DAPsaveat.jl")
include("MSE.jl")

include("Adaptive mesh/Duality.jl")
include("Adaptive mesh/P-Duality.jl")
include("Adaptive mesh/KKT.jl")
include("Adaptive mesh/P-KKT.jl")

timer = TimerOutputs.TimerOutput()

# Initialize model:
mod = E_coli_model
changeRxnBounds(mod, "EX_o2(e)", v_omax , "l")

Bio_index =  13
Ace_index =  20
Glu_index =  28

t_0 = 0     # h
t_end = 7.1 # h
N = 10      # Number of finite elements

# Dual:
display("Dual")
TimerOutputs.reset_timer!(timer)
t_Dual, G_Dual, X_Dual, A_Dual, status_Dual = TDual(mod, G_0, X_0, A_0, K_g, K_a,
↪  v_gmax, v_amax, Bio_index, Ace_index, Glu_index, t_0, t_end)
display(timer)

# P. Dual:
display("P. Dual:")
TimerOutputs.reset_timer!(timer)
t_PDual, G_PDual, X_PDual, A_PDual, status_PDual = TPDual(mod, G_0, X_0, A_0,
↪  K_g, K_a, v_gmax, v_amax, Bio_index, Ace_index, Glu_index, t_0, t_end)
display(timer)

# KKT:
display("KKT")
TimerOutputs.reset_timer!(timer)
t_KKT, G_KKT, X_KKT, A_KKT, status_KKT = TKKT(mod, G_0, X_0, A_0, K_g, K_a,
↪  v_gmax, v_amax, Bio_index, Ace_index, Glu_index, t_0, t_end)
display(timer)

# P. KKT:
display("P. KKT")
TimerOutputs.reset_timer!(timer)
t_PKKT, G_PKKT, X_PKKT, A_PKKT, status_PKKT = TPKKT(mod, G_0, X_0, A_0, K_g, K_a,
↪  v_gmax, v_amax, Bio_index, Ace_index, Glu_index, t_0, t_end)
display(timer)

# Display results:

t_DAP, G_DAP, X_DAP, A_DAP = DAPsave(mod, G_0, X_0, A_0, K_g, K_a, v_gmax,
↪  v_amax, Bio_index, Ace_index, Glu_index, t_end, t_Dual)
```

```
display("Dual:")
display(status_Dual)
display(MSE(G_DAP, G_Dual))
display(MSE(X_DAP, X_Dual))
display(MSE(A_DAP, A_Dual))

t_DAP, G_DAP, X_DAP, A_DAP = DAPsave(mod, G_0, X_0, A_0, K_g, K_a, v_gmax,
↪  v_amax, Bio_index, Ace_index, Glu_index, t_end, t_PDual)
display("P. Dual:")
display(status_PDual)
display(MSE(G_DAP, G_PDual))
display(MSE(X_DAP, X_PDual))
display(MSE(A_DAP, A_PDual))

t_DAP, G_DAP, X_DAP, A_DAP = DAPsave(mod, G_0, X_0, A_0, K_g, K_a, v_gmax,
↪  v_amax, Bio_index, Ace_index, Glu_index, t_end, t_KKT)
display("KKT:")
display(status_KKT)
display(MSE(G_DAP, G_KKT))
display(MSE(X_DAP, X_KKT))
display(MSE(A_DAP, A_KKT))

t_DAP, G_DAP, X_DAP, A_DAP = DAPsave(mod, G_0, X_0, A_0, K_g, K_a, v_gmax,
↪  v_amax, Bio_index, Ace_index, Glu_index, t_end, t_PKKT)
display("P. KKT:")
display(status_PKKT)
display(MSE(G_DAP, G_PKKT))
display(MSE(X_DAP, X_PKKT))
display(MSE(A_DAP, A_PKKT))

# Direct Approach:
display("DA:")
TimerOutputs.reset_timer!(timer)
t_DAP, G_DAP, X_DAP, A_DAP = DAP(mod, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax,
↪  Bio_index, Ace_index, Glu_index, t_end)
display(timer)

# Ploting the results:
using Plots

t_points_Dual = [t_0]
for n = 4:3:N*3+1
    global t_points_Dual
    t_points_Dual = vcat(t_points_Dual, t_Dual[n])
end

t_points = [t_0]
for n = 4:3:N*3+1
    global t_points
    t_points = vcat(t_points, t_PDual[n])
end

t_points_KKT = [t_0]
for n = 4:3:N*3+1
    global t_points_KKT
    t_points_KKT = vcat(t_points_KKT, t_KKT[n])
end
```

```
t_points_PKKT = [t_0]
for n = 4:3:N*3+1
    global t_points_PKKT
    t_points_PKKT = vcat(t_points_PKKT, t_PKKT[n])
end

pG = plot(t_DAP,G_DAP, title="Glucose", xaxis="Time [h]",yaxis="Concentration
↪  [mmol/L]", label="DA", lw=2, ls=:solid, color=:blue)
pG = plot!(t_Dual,G_Dual, label="Dual", lw=2, ls=:dashdot, color=:orange)
pG = plot!(t_PDual,G_PDual, label="P. Dual", lw=2, ls=:dash, color=:green)
pG = plot!(t_KKT,G_KKT, label="KKT", lw=2, ls=:dashdotdot, color=:purple)
pG = plot!(t_PKKT,G_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)
pG = scatter!(t_points_Dual, zeros(length(t_points_Dual)) ,label="Finite elements
↪  Dual",color=:orange, markershape=:star5)
pG = scatter!(t_points, zeros(length(t_points)) , label="Finite elements P.
↪  Dual", color=:green)
pG = scatter!(t_points_KKT, zeros(length(t_points_KKT)) ,label="Finite elements
↪  KKT",color=:purple, markershape=:square)
pG = scatter!(t_points_PKKT, zeros(length(t_points_PKKT)) ,label="Finite elements
↪  P. KKT",color=:red, markershape=:cross)

pX = plot(t_DAP,X_DAP, title="Biomass", xaxis="Time [h]",yaxis="Concentration
↪  [g/L]", label="DA", lw=2, ls=:solid, color=:blue)
pX = plot!(t_Dual,X_Dual, label="Dual", lw=2, ls=:dashdot, color=:orange)
pX = plot!(t_PDual,X_PDual, label="P. Dual", lw=2, ls=:dash, color=:green)
pX = plot!(t_KKT,X_KKT, label="KKT", lw=2, ls=:dashdotdot, color=:purple)
pX = plot!(t_PKKT,X_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)
pX = scatter!(t_points_Dual, zeros(length(t_points_Dual)) ,label="Finite elements
↪  Dual",color=:orange, markershape=:star5)
pX = scatter!(t_points, zeros(length(t_points)) ,label="Finite elements P. Dual",
↪  color=:green)
pX = scatter!(t_points_KKT, zeros(length(t_points_KKT)) ,label="Finite elements
↪  KKT",color=:purple, markershape=:square)
pX = scatter!(t_points_PKKT, zeros(length(t_points_PKKT)) ,label="Finite elements
↪  P. KKT",color=:red, markershape=:cross)

pA = plot(t_DAP,A_DAP, title="Acetate", xaxis="Time [h]",yaxis="Concentration
↪  [mmol/L]", label="DA", lw=2, ls=:solid, color=:blue, legend=:topleft)
pA = plot!(t_Dual,A_Dual, label="Dual", lw=2, ls=:dashdot, color=:orange)
pA = plot!(t_PDual,A_PDual, label="P. Dual", lw=2, ls=:dash, color=:green)
pA = plot!(t_KKT,A_KKT, label="KKT", lw=2, ls=:dashdotdot, color=:purple)
pA = plot!(t_PKKT,A_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)
pA = scatter!(t_points_Dual, zeros(length(t_points_Dual)) ,label="Finite elements
↪  Dual",color=:orange, markershape=:star5)
pA = scatter!(t_points, zeros(length(t_points)) ,label="Finite elements P. Dual",
↪  color=:green)
pA = scatter!(t_points_KKT, zeros(length(t_points_KKT)) ,label="Finite elements
↪  KKT",color=:purple, markershape=:square)
pA = scatter!(t_points_PKKT, zeros(length(t_points_PKKT)) ,label="Finite elements
↪  P. KKT",color=:red, markershape=:cross)

# Save plots:
savefig(pG,"SGT.pdf")
savefig(pX,"SXT.pdf")
savefig(pA,"SAT.pdf")
```

DAP.jl

DAP.jl is used to simulate the batch dFBA model of *Escherichia coli* core metabolic model using the Direct Approach (DA).

```
using Ipopt, DifferentialEquations, JuMP

# pFBA solver:
function pFBA(mod)

    S = mod.S
    c = mod.c
    UB = mod.UB
    LB = mod.LB

    W = zeros(Float64, size(S)[2], size(S)[2])
    for i = 1:size(S)[2]
        for j = 1:size(S)[2]
            if i == j
                W[i, j] = 1e-6
            end
        end
    end

    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "linear_solver", "ma97")
    JuMP.set_silent(model)

    JuMP.@variable(model, v[i=1:length(c)], start = mod.v[i])

    for i = 1:length(c)
        JuMP.@constraint(model, LB[i] <= v[i] <= UB[i])
    end

    JuMP.@constraint(model, S*v .== 0)

    JuMP.@expression(model, FO, -transpose(c)*v + transpose(v)*W*v)

    JuMP.@objective(model, Min, FO)

    JuMP.optimize!(model)

    v_opt = JuMP.value.(v)

    for i = 1:length(c)
        mod.v[i] = v_opt[i]
    end

    return JuMP.raw_status(model)
end

# Reactor rquations:
function reactor!(dC, C, consts, t)
    global status

    # Gathering constants:
    # ----------------------------------------------------
```

```
    K_g       = consts[1]
    K_a       = consts[2]
    v_gmax    = consts[3]
    v_amax    = consts[4]
    model     = consts[5]
    Bio_index = consts[6]
    Ace_index = consts[7]
    Glu_index = consts[8]

    # Gathering concentrations
    X = C[1]
    G = C[2]
    A = C[3]

    if A < 0
        A = 0
        changeRxnBounds(model, "EX_ac(e)", 0, "l")
    else
        v_a = v_amax*(A/(K_a + A))
        changeRxnBounds(model, "EX_ac(e)", v_a, "l")
    end
    if G < 0
        G = 0
        changeRxnBounds(model, "EX_glc(e)", 0, "l")
    else
        v_g = v_gmax*(G/(K_g + G))
        changeRxnBounds(model, "EX_glc(e)", v_g, "l")
    end

    st = pFBA(model)
    status = vcat(status, st)

    mu, v_g, v_a = model.v[Bio_index], model.v[Glu_index], model.v[Ace_index]

    # Mass balance equations:
    # ------------------------------------------------------
    dXdt = mu*X
    dGdt = v_g*X
    dAdt = v_a*X

    # Return result:
    # ------------------------------------------------------
    dC .= vcat(dXdt, dGdt, dAdt)

    return nothing
end

# Direct Approach:
function DAP(model, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax, Bio_index,
↪   Ace_index, Glu_index, t_end)

    global  status

    # Initialization:
    # ------------------------------------------------------
    init = vcat(X_0, G_0, A_0)
    consts = vcat(K_g, K_a, v_gmax, v_amax, model, Bio_index, Ace_index,
    ↪   Glu_index)
```

```
    status = []

    # Initializing boundary in t-direction:
    #-------------------------------------------------------------------------
    tspan = [0, t_end] # s

    # Solving the ODE-system:
    #-------------------------------------------------------------------------
    prob = ODEProblem(reactor!, init, tspan, consts)
    sol = DifferentialEquations.solve(prob, QNDF(autodiff=false), reltol=1e-6,
    ↪  abstol=1e-6)

    display(sol.destats)
    display(status)
    for i in status
        if i != "Solve_Succeeded"
            display(i)
        end
    end

    # Gathering results:
    #-------------------------------------------------------------------------
    t = Array{Float64}(undef,size(sol.t,1))
    C_X = Array{Float64}(undef,size(sol.t,1))
    C_G = Array{Float64}(undef,size(sol.t,1))
    C_A = Array{Float64}(undef,size(sol.t,1))

    # t2 - tend:
    #-------------------------------------------------------------------------
    for i=1:size(sol.t,1)
        t[i] = sol.t[i]
        C_X[i] = sol[i][1,1]
        C_G[i] = sol[i][2,1]
        C_A[i] = sol[i][3,1]
    end

    # Reset model:
    for i = 1:length(mod.v)
        mod.v[i] = 0.0
    end

    return t, C_G, C_X, C_A
end
```

Duality.jl

Duality.jl is used to simulate the batch dFBA model of *Escherichia coli* core metabolic model using the non-penalized duality theory Non-Linear Programming Approach (NLPA).

```
# Duality theory reformulation:

using Ipopt, JuMP

function Dual(mod, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax, Bio_index, Ace_index,
↪  Glu_index, t_0, t_end)
```

```
t_list = [t_0]
G_list = [G_0]
X_list = [X_0]
A_list = [A_0]

X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
↪
A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
↪
t =  [0.155051, 0.644949, 1.0000] # Collocation points
i = size(mod.S)[1]                # Numer of metabolites
j = size(mod.S)[2]                # Number of reactions

# Creating M-matrix:
M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
       t[2] 1/2*t[2]^2 1/3*t[2]^3
       t[3] 1/2*t[3]^2 1/3*t[3]^3
]
M_2 = [1 t[1] t[1]^2
       1 t[2] t[2]^2
       1 t[3] t[3]^2
]
M = M_1*M_2^(-1)

# Creating W-matrix:
W = zeros(Float64, j, j)
for k = 1:j
    for l = 1:j
        if k == l
            W[k, l] = 1e-6
        end
    end
end

# Initilazing UB:
UB = mod.UB

model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "linear_solver", "ma97")
set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
set_optimizer_attribute(model, "acceptable_tol", 1e-8)
JuMP.set_silent(model)

# Variabels:
JuMP.@variable(model, v[1:j, 1:N])   # Fluxes,  => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, G[1:3, 1:N])   # Glucose, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, X[1:3, 1:N])   # Biomass, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, A[1:3, 1:N])   # Acetate, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,  => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB my,   => Matrix: #reactions*#finite
↪  elements
```

```
JuMP.@variable(model, umy[1:j, 1:N]) # UB my,    => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds       => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, dt[1:N])       # Finite elment size => Vector: #finite
↪  elements

# Time constraints:
JuMP.@constraint(model, sum(dt) == t_end)
for n = 1:N
    JuMP.@constraint(model, 0 <= dt[n] <= 1)
end

# Defining LB:
if Ace_index < Glu_index
    for n = 1:N
        for k = 1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
else
    for n = 1:N
        for k = 1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
        JuMP.@constraint(model, lmy[k,n] >= 0)
        JuMP.@constraint(model, umy[k,n] >= 0)
    end
    for c = 1:3
        JuMP.@constraint(model, G[c,n] >= 0)
```

```
        JuMP.@constraint(model, X[c,n] >= 0)
        JuMP.@constraint(model, A[c,n] >= 0)
    end
end

# Orthogonal collocation constraints:
for c = 1:3
    JuMP.@NLconstraint(model, X[c,1] == X_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Bio_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Bio_index,1]*X[2,1]) + M[c,3]*(v[Bio_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, G[c,1] == G_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Glu_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Glu_index,1]*X[2,1]) + M[c,3]*(v[Glu_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, A[c,1] == A_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Ace_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Ace_index,1]*X[2,1]) + M[c,3]*(v[Ace_index,1]*X[3,1])))
    for n = 2:N
        JuMP.@NLconstraint(model, X[c,n] == X[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Bio_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Bio_index,n]*X[2,n]) + M[c,3]*(v[Bio_index,n]*X[3,n])))
        JuMP.@NLconstraint(model, G[c,n] == G[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Glu_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Glu_index,n]*X[2,n]) + M[c,3]*(v[Glu_index,n]*X[3,n])))
        JuMP.@NLconstraint(model, A[c,n] == A[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Ace_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Ace_index,n]*X[2,n]) + M[c,3]*(v[Ace_index,n]*X[3,n])))
    end
end

# Duality Constraints:
for n = 1:N
    JuMP.@constraint(model, mod.S*v[:,n] .== 0)
    JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] .+
    ↪  transpose(mod.S)*lam[:,n] .+ umy[:,n] .- lmy[:,n] .== 0)
    JuMP.@constraint(model, -transpose(mod.c)*v[:,n] +
    ↪  transpose(v[:,n])*W*v[:,n] == -transpose(v[:,n])*W*v[:,n] +
    ↪  transpose(lmy[:,n])*LB[:,n] - transpose(umy[:,n])*UB)
end

# Objective:
JuMP.@NLexpression(model, FO, sum(dt[n]*(G[1,n]-G[3,n]) for n=1:N))
JuMP.@NLobjective(model, Min, FO)

JuMP.optimize!(model)

# Gather the solution:
for n = 1:N
    t_list = vcat(t_list, [last(t_list)+0.155051*JuMP.value.(dt)[n],
    ↪  last(t_list)+0.644949*JuMP.value.(dt)[n],
    ↪  last(t_list)+1.0000*JuMP.value.(dt)[n]])
    for c = 1:3
        G_list = vcat(G_list, JuMP.value.(G)[c,n])
        X_list = vcat(X_list, JuMP.value.(X)[c,n])
        A_list = vcat(A_list, JuMP.value.(A)[c,n])
    end
end

return t_list, G_list, X_list, A_list, JuMP.raw_status(model)
```

```
end
```

P-Duality.jl

P-Duality.jl is used to simulate the batch dFBA model of *Escherichia coli* core metabolic model using the penalized duality theory NLPA.

```
# Penalized duality theory reformulation:

using Ipopt, JuMP

function PDual(mod, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax, Bio_index,
↪  Ace_index, Glu_index, t_0, t_end)

    t_list = [t_0]
    G_list = [G_0]
    X_list = [X_0]
    A_list = [A_0]

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    ↪
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    ↪
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Numer of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
           1 t[3] t[3]^2
    ]
    M = M_1*M_2^(-1)

    # Creating W-matrix:
    W = zeros(Float64, j, j)
    for k = 1:j
        for l = 1:j
            if k == l
                W[k, l] = 1e-6
            end
        end
    end

    # Initilazing UB:
    UB = mod.UB
```

```
model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "linear_solver", "ma97")
set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
set_optimizer_attribute(model, "acceptable_tol", 1e-8)
JuMP.set_silent(model)

# Variabels:
JuMP.@variable(model, v[1:j, 1:N])    # Fluxes,  => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, G[1:3, 1:N])    # Glucose, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, X[1:3, 1:N])    # Biomass, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, A[1:3, 1:N])    # Acetate, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,  => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB my,   => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB my,   => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds       => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, dt[1:N])       # Finite elment size => Vector: #finite
↪  elements

# Time constraints:
JuMP.@constraint(model, sum(dt) == t_end)
for n = 1:N
    JuMP.@constraint(model, 0 <= dt[n] <= 1)
end

# Defining LB:
if Ace_index < Glu_index
    for n = 1:N
        for k = 1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
else
    for n = 1:N
        for k = 1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
```

```
            end
            JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
            ↪  A[3,n]))
            for k = Ace_index+1:j
                JuMP.@constraint(model, LB[k,n] == mod.LB[k])
            end
        end
    end

    # Constraints for variables:
    for n = 1:N
        for k = 1:j
            JuMP.@constraint(model, v[k,n] <= UB[k])
            JuMP.@constraint(model, LB[k,n] <= v[k,n])
            JuMP.@constraint(model, lmy[k,n] >= 0)
            JuMP.@constraint(model, umy[k,n] >= 0)
        end
        for c = 1:3
            JuMP.@constraint(model, G[c,n] >= 0)
            JuMP.@constraint(model, X[c,n] >= 0)
            JuMP.@constraint(model, A[c,n] >= 0)
        end
    end

    # Orthogonal collocation constraints:
    for c = 1:3
        JuMP.@NLconstraint(model, X[c,1] == X_0[c] +
        ↪  dt[1]*(M[c,1]*(v[Bio_index,1]*X[1,1]) +
        ↪  M[c,2]*(v[Bio_index,1]*X[2,1]) + M[c,3]*(v[Bio_index,1]*X[3,1])))
        JuMP.@NLconstraint(model, G[c,1] == G_0[c] +
        ↪  dt[1]*(M[c,1]*(v[Glu_index,1]*X[1,1]) +
        ↪  M[c,2]*(v[Glu_index,1]*X[2,1]) + M[c,3]*(v[Glu_index,1]*X[3,1])))
        JuMP.@NLconstraint(model, A[c,1] == A_0[c] +
        ↪  dt[1]*(M[c,1]*(v[Ace_index,1]*X[1,1]) +
        ↪  M[c,2]*(v[Ace_index,1]*X[2,1]) + M[c,3]*(v[Ace_index,1]*X[3,1])))
        for n = 2:N
            JuMP.@NLconstraint(model, X[c,n] == X[3,n-1] +
            ↪  dt[n]*(M[c,1]*(v[Bio_index,n]*X[1,n]) +
            ↪  M[c,2]*(v[Bio_index,n]*X[2,n]) + M[c,3]*(v[Bio_index,n]*X[3,n])))
            JuMP.@NLconstraint(model, G[c,n] == G[3,n-1] +
            ↪  dt[n]*(M[c,1]*(v[Glu_index,n]*X[1,n]) +
            ↪  M[c,2]*(v[Glu_index,n]*X[2,n]) + M[c,3]*(v[Glu_index,n]*X[3,n])))
            JuMP.@NLconstraint(model, A[c,n] == A[3,n-1] +
            ↪  dt[n]*(M[c,1]*(v[Ace_index,n]*X[1,n]) +
            ↪  M[c,2]*(v[Ace_index,n]*X[2,n]) + M[c,3]*(v[Ace_index,n]*X[3,n])))
        end
    end

    # Duality Constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] .+
        ↪  transpose(mod.S)*lam[:,n] .+ umy[:,n] .- lmy[:,n] .== 0)
    end

    # Objective:
```

```
    JuMP.@NLexpression(model, FO, sum(sum(-mod.c[l]*v[l,n] +
    ↪  2*v[l,n]*W[l,l]*v[l,n] - lmy[l,n]*LB[l,n] + umy[l,n]*UB[l] for l in 1:j)
    ↪  for n in 1:N) + 0.1*sum(dt[n]*(G[1,n]-G[3,n]) for n=1:N))
    JuMP.@NLobjective(model, Min, FO)

    JuMP.optimize!(model)

    # Gather the solution:
    for n = 1:N
        t_list = vcat(t_list, [last(t_list)+0.155051*JuMP.value.(dt)[n],
        ↪  last(t_list)+0.644949*JuMP.value.(dt)[n],
        ↪  last(t_list)+1.0000*JuMP.value.(dt)[n]])
        for c = 1:3
            G_list = vcat(G_list, JuMP.value.(G)[c,n])
            X_list = vcat(X_list, JuMP.value.(X)[c,n])
            A_list = vcat(A_list, JuMP.value.(A)[c,n])
        end
    end

    return t_list, G_list, X_list, A_list, JuMP.raw_status(model)
end
```

KKT.jl

KKT.jl is used to simulate the batch dFBA model of *Escherichia coli* core metabolic model using the non-penalized Karush–Kuhn–Tucker (KKT) NLPA.

```
# KKT reformulation:

using Ipopt, JuMP

function KKT(mod, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax, Bio_index, Ace_index,
↪  Glu_index, t_0, t_end)

    t_list = [t_0]
    G_list = [G_0]
    X_list = [X_0]
    A_list = [A_0]

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    ↪
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    ↪
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Numer of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
```

```
        1 t[3] t[3]^2
]
M = M_1*M_2^(-1)

# Creating W-matrix:
W = zeros(Float64, j, j)
for k = 1:j
    for l = 1:j
        if k == l
            W[k, l] = 1e-6
        end
    end
end

# Initilazing UB:
UB = mod.UB

model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "linear_solver", "ma97")
set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
set_optimizer_attribute(model, "acceptable_tol", 1e-8)
JuMP.set_silent(model)

# Variabels:
JuMP.@variable(model, v[1:j, 1:N])    # Fluxes,  => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, G[1:3, 1:N])    # Glucose, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, X[1:3, 1:N])    # Biomass, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, A[1:3, 1:N])    # Acetate, => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,  => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB my,   => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB my,   => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds       => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, dt[1:N])       # Finite elment size => Vector: #finite
↪  elements

# Time constraints:
JuMP.@constraint(model, sum(dt) == t_end)
for n = 1:N
    JuMP.@constraint(model, 0 <= dt[n] <= 1)
end

# Defining LB:
if Ace_index < Glu_index
    for n = 1:N
        for k = 1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:Glu_index-1
```

```
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
else
    for n = 1:N
        for k = 1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
        JuMP.@constraint(model, lmy[k,n] >= 0)
        JuMP.@constraint(model, umy[k,n] >= 0)
    end
    for c = 1:3
        JuMP.@constraint(model, G[c,n] >= 0)
        JuMP.@constraint(model, X[c,n] >= 0)
        JuMP.@constraint(model, A[c,n] >= 0)
    end
end

# Orthogonal collocation constraints:
for c = 1:3
    JuMP.@NLconstraint(model, X[c,1] == X_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Bio_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Bio_index,1]*X[2,1]) + M[c,3]*(v[Bio_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, G[c,1] == G_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Glu_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Glu_index,1]*X[2,1]) + M[c,3]*(v[Glu_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, A[c,1] == A_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Ace_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Ace_index,1]*X[2,1]) + M[c,3]*(v[Ace_index,1]*X[3,1])))
    for n = 2:N
        JuMP.@NLconstraint(model, X[c,n] == X[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Bio_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Bio_index,n]*X[2,n]) + M[c,3]*(v[Bio_index,n]*X[3,n])))
```

```
            JuMP.@NLconstraint(model, G[c,n] == G[3,n-1] +
            ↪  dt[n]*(M[c,1]*(v[Glu_index,n]*X[1,n]) +
            ↪  M[c,2]*(v[Glu_index,n]*X[2,n]) + M[c,3]*(v[Glu_index,n]*X[3,n])))
            JuMP.@NLconstraint(model, A[c,n] == A[3,n-1] +
            ↪  dt[n]*(M[c,1]*(v[Ace_index,n]*X[1,n]) +
            ↪  M[c,2]*(v[Ace_index,n]*X[2,n]) + M[c,3]*(v[Ace_index,n]*X[3,n])))
        end
    end

    # Duality Constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] .+
        ↪  transpose(mod.S)*lam[:,n] .+ umy[:,n] .- lmy[:,n] .== 0)
    end

    # CS:
    for n = 1:N
        JuMP.@constraint(model, transpose(lmy[:,n])*(v[:,n] .- LB[:,n]) == 0)
        JuMP.@constraint(model, transpose(umy[:,n])*(UB .- v[:,n]) == 0)
    end

    # Objective:
    JuMP.@NLexpression(model, FO, sum(dt[n]*(G[1,n]-G[3,n]) for n=1:N))
    JuMP.@NLobjective(model, Min, FO)

    JuMP.optimize!(model)

    # Gather the solution:
    for n = 1:N
        t_list = vcat(t_list, [last(t_list)+0.155051*JuMP.value.(dt)[n],
        ↪  last(t_list)+0.644949*JuMP.value.(dt)[n],
        ↪  last(t_list)+1.0000*JuMP.value.(dt)[n]])
        for c = 1:3
            G_list = vcat(G_list, JuMP.value.(G)[c,n])
            X_list = vcat(X_list, JuMP.value.(X)[c,n])
            A_list = vcat(A_list, JuMP.value.(A)[c,n])
        end
    end

    return t_list, G_list, X_list, A_list, JuMP.raw_status(model)
end
```

P-KKT.jl

P-KKT.jl is used to simulate the batch dFBA model of *Escherichia coli* core metabolic model using the penalized KKT NLPA.

```
# Penalized KKT reformulation:

using Ipopt, JuMP
```

```
function PKKT(mod, G_0, X_0, A_0, K_g, K_a, v_gmax, v_amax, Bio_index, Ace_index,
↪  Glu_index, t_0, t_end)

    t_list = [t_0]
    G_list = [G_0]
    X_list = [X_0]
    A_list = [A_0]

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    ↪
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    ↪
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Numer of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
           1 t[3] t[3]^2
    ]
    M = M_1*M_2^(-1)

    # Creating W-matrix:
    W = zeros(Float64, j, j)
    for k = 1:j
        for l = 1:j
            if k == l
                W[k, l] = 1e-6
            end
        end
    end

    # Initilazing UB:
    UB = mod.UB

    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "linear_solver", "ma97")
    set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
    set_optimizer_attribute(model, "acceptable_tol", 1e-8)
    JuMP.set_silent(model)

    # Variabels:
    JuMP.@variable(model, v[1:j, 1:N])    # Fluxes,  => Matrix: #reactions*#finite
    ↪  elements
    JuMP.@variable(model, G[1:3, 1:N])    # Glucose, => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, X[1:3, 1:N])    # Biomass, => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, A[1:3, 1:N])    # Acetate, => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,  => Matrix:
    ↪  #metabolites*#finite elements
```

```
JuMP.@variable(model, lmy[1:j, 1:N]) # LB my,    => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB my,    => Matrix: #reactions*#finite
↪  elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds       => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, dt[1:N])       # Finite elment size => Vector: #finite
↪  elements

# Time constraints:
JuMP.@constraint(model, sum(dt) == t_end)
for n = 1:N
    JuMP.@constraint(model, 0 <= dt[n] <= 1)
end

# Defining LB:
if Ace_index < Glu_index
    for n = 1:N
        for k = 1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
else
    for n = 1:N
        for k = 1:Glu_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Glu_index,n] == v_gmax*G[3,n]/(K_g +
        ↪  G[3,n]))
        for k = Glu_index+1:Ace_index-1
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
        JuMP.@NLconstraint(model, LB[Ace_index,n] == v_amax*A[3,n]/(K_a +
        ↪  A[3,n]))
        for k = Ace_index+1:j
            JuMP.@constraint(model, LB[k,n] == mod.LB[k])
        end
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
        JuMP.@constraint(model, lmy[k,n] >= 0)
        JuMP.@constraint(model, umy[k,n] >= 0)
    end
```

```
    for c = 1:3
        JuMP.@constraint(model, G[c,n] >= 0)
        JuMP.@constraint(model, X[c,n] >= 0)
        JuMP.@constraint(model, A[c,n] >= 0)
    end
end

# Orthogonal collocation constraints:
for c = 1:3
    JuMP.@NLconstraint(model, X[c,1] == X_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Bio_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Bio_index,1]*X[2,1]) + M[c,3]*(v[Bio_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, G[c,1] == G_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Glu_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Glu_index,1]*X[2,1]) + M[c,3]*(v[Glu_index,1]*X[3,1])))
    JuMP.@NLconstraint(model, A[c,1] == A_0[c] +
    ↪  dt[1]*(M[c,1]*(v[Ace_index,1]*X[1,1]) +
    ↪  M[c,2]*(v[Ace_index,1]*X[2,1]) + M[c,3]*(v[Ace_index,1]*X[3,1])))
    for n = 2:N
        JuMP.@NLconstraint(model, X[c,n] == X[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Bio_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Bio_index,n]*X[2,n]) + M[c,3]*(v[Bio_index,n]*X[3,n])))
        JuMP.@NLconstraint(model, G[c,n] == G[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Glu_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Glu_index,n]*X[2,n]) + M[c,3]*(v[Glu_index,n]*X[3,n])))
        JuMP.@NLconstraint(model, A[c,n] == A[3,n-1] +
        ↪  dt[n]*(M[c,1]*(v[Ace_index,n]*X[1,n]) +
        ↪  M[c,2]*(v[Ace_index,n]*X[2,n]) + M[c,3]*(v[Ace_index,n]*X[3,n])))
    end
end

# Duality Constraints:
for n = 1:N
    JuMP.@constraint(model, mod.S*v[:,n] .== 0)
    JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] .+
    ↪  transpose(mod.S)*lam[:,n] .+ umy[:,n] .- lmy[:,n] .== 0)
end

# Objective:
JuMP.@NLexpression(model, FO, sum(sum(((v[l,n] - LB[l,n])*lmy[l,n]) +
↪  (-v[l,n] + UB[l])*umy[l,n] for l in 1:j) for n in 1:N) +
↪  0.1*sum(dt[n]*(G[1,n]-G[3,n]) for n=1:N))
JuMP.@NLobjective(model, Min, FO)

JuMP.optimize!(model)

# Gather the solution:
for n = 1:N
    t_list = vcat(t_list, [last(t_list)+0.155051*JuMP.value.(dt)[n],
    ↪  last(t_list)+0.644949*JuMP.value.(dt)[n],
    ↪  last(t_list)+1.0000*JuMP.value.(dt)[n]])
    for c = 1:3
        G_list = vcat(G_list, JuMP.value.(G)[c,n])
        X_list = vcat(X_list, JuMP.value.(X)[c,n])
        A_list = vcat(A_list, JuMP.value.(A)[c,n])
    end
end
```

```
    return t_list, G_list, X_list, A_list, JuMP.raw_status(model)
end
```

B.4 MPC models

In this subsection we present some the code used to tests the Model Predictive Controller (MPC).

main.jl

main.jl is used to initialize the system, run the MPC loop, and to plot the results.

```
# include("MPC/main.jl")

import Ipopt, JuMP, Plots

include("E. coli model/E_coli_Model.jl")
include("init.jl")
include("DA.jl")
include("pFBA reformulations/Duality.jl")
include("pFBA reformulations/P-Duality.jl")
include("pFBA reformulations/KKT.jl")
include("pFBA reformulations/P-KKT.jl")

# Initialize E. coli core metabolic network:
mod = E_coli_Model_init
changeRxnBounds(mod, "EX_o2(e)", v_omax , "l")

# Initialize MPC:
t_0 = 0    # h
t_end = 35 # h
N = 35
dt = (t_end-t_0)/N
sp  = [0.2*ones(1,6) 0.4*ones(1,15) 0.2*ones(1,15)] # 0.2*ones(1,N+1)
G_f = G_f0*ones(1,N+1) # [G_f0*ones(1,6) 8*ones(1,15) G_f0*ones(1,15)] #
↪  G_f0*ones(1,N+1)
v_gmax = v_gmax_0*ones(1,N+1) #[v_gmax_0*ones(1,6) -5.25*ones(1,15)
↪  v_gmax_0*ones(1,15)]

time = Array{Float64}(undef, N+1)
time[1] = t_0

# Dual:
G_Dual =    Array{Float64}(undef, N+1)
X_Dual =    Array{Float64}(undef, N+1)
A_Dual =    Array{Float64}(undef, N+1)
D_Dual =    Array{Float64}(undef, N+1)
time_Dual = Array{Float64}(undef, N)
status_Dual = Array{Any}(undef, N)
t_list_Dual = [t_0]
G_list_Dual = [G_0]
X_list_Dual = [X_0]
A_list_Dual = [A_0]
G_Dual[1]    = G_0
```

```
X_Dual[1]    = X_0
A_Dual[1]    = A_0
D_Dual[1]    = D_0

# P. Dual:
G_PDual =    Array{Float64}(undef, N+1)
X_PDual =    Array{Float64}(undef, N+1)
A_PDual =    Array{Float64}(undef, N+1)
D_PDual =    Array{Float64}(undef, N+1)
time_PDual = Array{Float64}(undef, N)
status_PDual = Array{Any}(undef, N)
t_list_PDual = [t_0]
G_list_PDual = [G_0]
X_list_PDual = [X_0]
A_list_PDual = [A_0]
G_PDual[1]    = G_0
X_PDual[1]    = X_0
A_PDual[1]    = A_0
D_PDual[1]    = D_0

# KKT:
G_KKT =    Array{Float64}(undef, N+1)
X_KKT =    Array{Float64}(undef, N+1)
A_KKT =    Array{Float64}(undef, N+1)
D_KKT =    Array{Float64}(undef, N+1)
time_KKT = Array{Float64}(undef, N)
status_KKT = Array{Any}(undef, N)
t_list_KKT = [t_0]
G_list_KKT = [G_0]
X_list_KKT = [X_0]
A_list_KKT = [A_0]
G_KKT[1]    = G_0
X_KKT[1]    = X_0
A_KKT[1]    = A_0
D_KKT[1]    = D_0

# P. KKT:
G_PKKT =    Array{Float64}(undef, N+1)
X_PKKT =    Array{Float64}(undef, N+1)
A_PKKT =    Array{Float64}(undef, N+1)
D_PKKT =    Array{Float64}(undef, N+1)
time_PKKT = Array{Float64}(undef, N)
status_PKKT = Array{Any}(undef, N)
t_list_PKKT = [t_0]
G_list_PKKT = [G_0]
X_list_PKKT = [X_0]
A_list_PKKT = [A_0]
G_PKKT[1]    = G_0
X_PKKT[1]    = X_0
A_PKKT[1]    = A_0
D_PKKT[1]    = D_0

# MSE:
MSE_Dual = 0
MSE_PDual = 0
MSE_KKT = 0
MSE_PKKT = 0
```

```
for n = 1:N
    global t_list_Dual, G_list_Dual, X_list_Dual, A_list_Dual, t_list_PDual,
    ↪  G_list_PDual, X_list_PDual, A_list_PDual, t_list_PKKT, G_list_PKKT,
    ↪  X_list_PKKT, A_list_PKKT, MSE_Dual, MSE_PDual, MSE_PKKT, time_Dual,
    ↪  time_PDual, time_PKKT,status_Dual, status_PDual, status_PKKT, status_KKT,
    ↪  t_list_KKT, G_list_KKT, X_list_KKT, A_list_KKT, MSE_KKT
    display(n)

    time[n+1] = time[n]+dt

    # MPC control optimizations:
    D_Dual[n+1], time_Dual[n], status_Dual[n] = Dual(mod, G_Dual[n], X_Dual[n],
    ↪  A_Dual[n], K_g, v_gmax[n], 4, D_Dual[n], G_f[n], sp[n])
    D_PDual[n+1], time_PDual[n], status_PDual[n] = PDual(mod, G_PDual[n],
    ↪  X_PDual[n], A_PDual[n], K_g, v_gmax[n], 4, D_PDual[n], G_f[n], sp[n])
    D_KKT[n+1], time_KKT[n], status_KKT[n] = KKT(mod, G_KKT[n], X_KKT[n],
    ↪  A_KKT[n], K_g, v_gmax[n], 4, D_KKT[n], G_f[n], sp[n])
    D_PKKT[n+1], time_PKKT[n], status_PKKT[n] = PKKT(mod, G_PKKT[n], X_PKKT[n],
    ↪  A_PKKT[n], K_g, v_gmax[n], 4, D_PKKT[n], G_f[n], sp[n])

    # Dual:
    G_Dual[n+1], X_Dual[n+1], A_Dual[n+1], C_t, C_G, C_X, C_A = DAP(mod,
    ↪  G_Dual[n], X_Dual[n], A_Dual[n], K_g, v_gmax[n], v_omax, time[n],
    ↪  time[n+1], D_Dual[n+1], G_f[n])
    t_list_Dual = vcat(t_list_Dual, C_t)
    G_list_Dual = vcat(G_list_Dual, C_G)
    X_list_Dual = vcat(X_list_Dual, C_X)
    A_list_Dual = vcat(A_list_Dual, C_A)

    # P. Dual:
    G_PDual[n+1], X_PDual[n+1], A_PDual[n+1], C_t, C_G, C_X, C_A = DAP(mod,
    ↪  G_PDual[n], X_PDual[n], A_PDual[n], K_g, v_gmax[n], v_omax, time[n],
    ↪  time[n+1], D_PDual[n+1], G_f[n])
    t_list_PDual = vcat(t_list_PDual, C_t)
    G_list_PDual = vcat(G_list_PDual, C_G)
    X_list_PDual = vcat(X_list_PDual, C_X)
    A_list_PDual = vcat(A_list_PDual, C_A)

    # KKT:
    G_KKT[n+1], X_KKT[n+1], A_KKT[n+1], C_t, C_G, C_X, C_A = DAP(mod, G_KKT[n],
    ↪  X_KKT[n], A_KKT[n], K_g, v_gmax[n], v_omax, time[n], time[n+1],
    ↪  D_KKT[n+1], G_f[n])
    t_list_KKT = vcat(t_list_KKT, C_t)
    G_list_KKT = vcat(G_list_KKT, C_G)
    X_list_KKT = vcat(X_list_KKT, C_X)
    A_list_KKT = vcat(A_list_KKT, C_A)

    # P. KKT:
    G_PKKT[n+1], X_PKKT[n+1], A_PKKT[n+1], C_t, C_G, C_X, C_A = DAP(mod,
    ↪  G_PKKT[n], X_PKKT[n], A_PKKT[n], K_g, v_gmax[n], v_omax, time[n],
    ↪  time[n+1], D_PKKT[n+1], G_f[n])
    t_list_PKKT = vcat(t_list_PKKT, C_t)
    G_list_PKKT = vcat(G_list_PKKT, C_G)
    X_list_PKKT = vcat(X_list_PKKT, C_X)
    A_list_PKKT = vcat(A_list_PKKT, C_A)

    # MSE:
    MSE_Dual = MSE_Dual + (X_Dual[n]- sp[n])^2
```

```
    MSE_PDual = MSE_PDual + (X_PDual[n]- sp[n])^2
    MSE_KKT = MSE_KKT + (X_KKT[n]- sp[n])^2
    MSE_PKKT = MSE_PKKT + (X_PKKT[n]- sp[n])^2
end

# Calculate MSE:
MSE_Dual = MSE_Dual + (X_Dual[N+1]- sp[N+1])^2
MSE_PDual = MSE_PDual + (X_PDual[N+1]- sp[N+1])^2
MSE_KKT = MSE_KKT + (X_KKT[N+1]- sp[N+1])^2
MSE_PKKT = MSE_PKKT + (X_PKKT[N+1]- sp[N+1])^2

# Display results:
display("Status:")
display("Dual:")
for i in status_Dual
    if i != "Solve_Succeeded"
        display(i)
    end
end
display("P. Dual:")
for i in status_PDual
    if i != "Solve_Succeeded"
        display(i)
    end
end
display("KKT:")
for i in status_KKT
    if i != "Solve_Succeeded"
        display(i)
    end
end
display("P. KKT:")
for i in status_PKKT
    if i != "Solve_Succeeded"
        display(i)
    end
end

display("MSE:")
display(MSE_Dual/N)
display(MSE_PDual/N)
display(MSE_KKT/N)
display(MSE_PKKT/N)

display("Time:")
display(sum(time_Dual)/N)
display(sum(time_PDual)/N)
display(sum(time_KKT)/N)
display(sum(time_PKKT)/N)

# Plot results:
using Plots

pG =  plot(t_list_Dual,G_list_Dual, title="Glucose", xaxis="Time
↪  [h]",yaxis="Concentration [mmol/L]", label="Dual", lw=2, ls=:solid,
↪  color=:orange)
```

```
pG =  plot!(t_list_PDual,G_list_PDual, title="Glucose", xaxis="Time
↪  [h]",yaxis="Concentration [mmol/L]", label="P. Dual", lw=2, ls=:dash,
↪  color=:green)
pG =  plot!(t_list_KKT,G_list_KKT, label="KKT", lw=2, ls=:dashdotdot,
↪  color=:purple)
pG =  plot!(t_list_PKKT,G_list_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)

pX =  plot(t_list_Dual,X_list_Dual, title="Biomass", xaxis="Time
↪  [h]",yaxis="Concentration [g/L]", label="Dual", lw=2, ls=:solid,
↪  color=:orange)
pX =  plot!(t_list_PDual,X_list_PDual, title="Biomass", xaxis="Time
↪  [h]",yaxis="Concentration [g/L]", label="P. Dual", lw=2, ls=:dash,
↪  color=:green)
pX = plot!(t_list_KKT,X_list_KKT, label="KKT", lw=2, ls=:dashdotdot,
↪  color=:purple)
pX =  plot!(t_list_PKKT,X_list_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)
pX =  plot!(time, sp[1,:], label="Setpoint", lw=2, ls=:dash, color=:blue,
↪  linetype=:steppost)

pA =  plot(t_list_Dual,A_list_Dual, title="Acetate", xaxis="Time
↪  [h]",yaxis="Concentration [mmol/L]", label="Dual", lw=2, ls=:solid,
↪  color=:orange)
pA =  plot!(t_list_PDual,A_list_PDual, title="Acetate", xaxis="Time
↪  [h]",yaxis="Concentration [mmol/L]", label="P. Dual", lw=2, ls=:dash,
↪  color=:green)
pA = plot!(t_list_KKT,A_list_KKT, label="KKT", lw=2, ls=:dashdotdot,
↪  color=:purple)
pA =  plot!(t_list_PKKT,A_list_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red)

pD =  plot(time, D_Dual, title="Dilution rate", xaxis="Time [h]",yaxis="Dilution
↪  rate [h^-1]", label="Dual", lw=2, ls=:solid, color=:orange,
↪  linetype=:steppost)
pD =  plot!(time, D_PDual, title="Dilution rate", xaxis="Time
↪  [h]",yaxis="Dilution rate [h^-1]", label="P. Dual", lw=2, ls=:dash,
↪  color=:green, linetype=:steppost)
pD =  plot!(time, D_KKT, label="KKT", lw=2, ls=:dashdotdot, color=:purple,
↪  linetype=:steppost)
pD =  plot!(time, D_PKKT, label="P. KKT", lw=2, ls=:dot, color=:red,
↪  linetype=:steppost)

pGf =  plot(time, G_f[1,:], title="Glucose feed", xaxis="Time
↪  [h]",yaxis="Concentration [mmol/L]", label="Glucose feed", lw=2, ls=:dash,
↪  color=:blue, linetype=:steppost)

pgmax =  plot(time, v_gmax[1,:], title="Maximal glucose uptake", xaxis="Time
↪  [h]",yaxis="Maximal glucose uptake", label="Maximal glucose uptake", lw=2,
↪  ls=:dash, color=:blue, linetype=:steppost)

# Save plots:
savefig(pG,"MPC-G.pdf")
savefig(pX,"MPC-X.pdf")
savefig(pA,"MPC-A.pdf")
savefig(pD,"MPC-D.pdf")
savefig(pGf,"MPC-Gf.pdf")
savefig(pgmax,"MPC-gmax.pdf")
```

DA.jl

DA.jl is used to simulate the CSTR dFBA model of *Escherichia coli* core metabolic model using the Direct Approach (DA).

```
using Ipopt, DifferentialEquations, JuMP

# pFBA solver:
function pFBA(mod)

    S = mod.S
    c = mod.c
    UB = mod.UB
    LB = mod.LB

    W = zeros(Float64, size(S)[2], size(S)[2])
    for i = 1:size(S)[2]
        for j = 1:size(S)[2]
            if i == j
                W[i, j] = 1e-6
            end
        end
    end

    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "linear_solver", "ma97")
    set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
    set_optimizer_attribute(model, "acceptable_tol", 1e-10)
    set_optimizer_attribute(model, "tol", 1e-10)
    JuMP.set_silent(model)

    JuMP.@variable(model, v[i=1:length(c)], start = mod.v[i])

    for i = 1:length(c)
        JuMP.@constraint(model, LB[i] <= v[i] <= UB[i])
    end

    JuMP.@constraint(model, S*v .== 0)

    JuMP.@expression(model, FO, -transpose(c)*v + transpose(v)*W*v)

    JuMP.@objective(model, Min, FO)

    JuMP.optimize!(model)

    v_opt = JuMP.value.(v)

    for i = 1:length(c)
        mod.v[i] = v_opt[i]
    end

    return nothing
end

# Reactor equations:
function reactor!(dC, C, consts, t)
```

```
    # Gathering constants:
    # -----------------------------------------------------
    K_g      = consts[1]
    v_gmax   = consts[2]
    v_omax   = consts[3]
    model    = consts[4]
    D        = consts[5]
    G_f      = consts[6]

    # Gathering concentrations
    X = C[1]
    G = C[2]
    A = C[3]

    # Calculating variabels
    if A < 0
        changeRxnBounds(model, "EX_ac(e)", 0, "l")
    else
        v_a = -2.5*(A/(0.01 + A))
        changeRxnBounds(model, "EX_ac(e)", v_a, "l")
    end
    if G < 0
        changeRxnBounds(model, "EX_glc(e)", 0, "l")
    else
        v_g = v_gmax*(G/(K_g + G))
        changeRxnBounds(model, "EX_glc(e)", v_g, "l")
    end
    pFBA(model)
    mu, v_g, v_a = model.v[13], model.v[28], model.v[20]

    # Mass balance equations:
    # -----------------------------------------------------
    dXdt = -D*X + mu*X
    dGdt = D*(G_f - G) + v_g*X
    dAdt = -D.*A + v_a*X

    # Return result:
    # -----------------------------------------------------
    dC .= vcat(dXdt, dGdt, dAdt)

    return nothing
end

# Direct Approach
function DAP(model, G_0, X_0, A_0, K_g, v_gmax, v_omax, t_0, t_end, D, G_f)

    # Initialization:
    # -----------------------------------------------------
    init = vcat(X_0, G_0, A_0)
    consts = vcat(K_g, v_gmax, v_omax, model, D, G_f)

    # Initializing boundary in t-direction:
    #-------------------------------------------------------------------
    tspan = [t_0, t_end] # s

    # Solving the ODE-system:
    #-------------------------------------------------------------------
    prob = ODEProblem(reactor!, init, tspan, consts)
```

```
    sol = DifferentialEquations.solve(prob, QNDF(autodiff=false), reltol=1e-6,
    ↪  abstol=1e-6)

    display(sol.destats)

    # Gathering results:
    #-------------------------------------------------------------------
    t = Array{Float64}(undef,size(sol.t,1))
    C_X = Array{Float64}(undef,size(sol.t,1))
    C_G = Array{Float64}(undef,size(sol.t,1))
    C_A = Array{Float64}(undef,size(sol.t,1))

    # t2 - tend:
    #-------------------------------------------------------------------
    for i=1:size(sol.t,1)
        t[i] = sol.t[i]
        C_X[i] = sol[i][1,1]
        C_G[i] = sol[i][2,1]
        C_A[i] = sol[i][3,1]
    end

    # Reset model
    for i = 1:length(mod.v)
        mod.v[i] = 0.0
    end

    # Return results:
    return last(C_G), last(C_X), last(C_A), t, C_G, C_X, C_A
end
```

Duality.jl

Duality.jl is used to solve one MPC optimization with the non-penalized duality theory Non-Linear Programming Approach (NLPA).

```
# Duality theory MPC reformulation:

using Ipopt, JuMP

function Dual(mod, G_0, X_0, A_0, K_g, v_gmax, N, u, G_f, sp)

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Number of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
```

```
        1 t[3] t[3]^2
]
M = M_1*M_2^(-1)

# Creating W-matrix:
W = zeros(Float64, j, j)
for k = 1:j
    for l = 1:j
        if k == l
            W[k, l] = 1e-6
        end
    end
end

# Initilaizing UB:
UB = mod.UB

model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "linear_solver", "ma97")
set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
set_optimizer_attribute(model, "acceptable_tol", 1e-8)
JuMP.set_silent(model)

# Variabels:
JuMP.@variable(model, v[1:j, 1:N])   # Fluxes,       => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, G[1:3, 1:N])   # Glucose,      => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, X[1:3, 1:N])   # Biomass,      => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, A[1:3, 1:N])   # Acetate,      => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,       => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB-my,        => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB-my,        => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds  => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, D[1:N])        # Dilution rate => Vector: #finite
↪  elements

# Defining LB:
for n = 1:N
    for k = 1:20-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[20,n] == -2.5*A[3,n]/(0.01 + A[3,n]))
    for k = 20+1:28-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[28,n] == v_gmax*G[3,n]/(K_g + G[3,n]))
    for k = 28+1:j
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
end
```

```
    # Constraints for variables:
    for n = 1:N
        for k = 1:j
            JuMP.@constraint(model, v[k,n] <= UB[k])
            JuMP.@constraint(model, LB[k,n] <= v[k,n])
            JuMP.@constraint(model, lmy[k,n] >= 0)
            JuMP.@constraint(model, umy[k,n] >= 0)
        end
        for c = 1:3
            JuMP.@constraint(model, G[c,n] >= 0)
            JuMP.@constraint(model, X[c,n] >= 0)
            JuMP.@constraint(model, A[c,n] >= 0)
        end
        JuMP.@constraint(model, 0 <= D[n] <= 1)
    end

    # Orthogonal collocation constraints:
    JuMP.@constraint(model, X[:,1] .== X_0 .+ dt*M*(D[1]*(- X[:,1]) .+
    ↪  v[13,1].*X[:,1]))
    JuMP.@constraint(model, G[:,1] .== G_0 .+ dt*M*(D[1]*(G_f .- G[:,1]) .+
    ↪  v[28,1].*X[:,1]))
    JuMP.@constraint(model, A[:,1] .== A_0 .+ dt*M*(D[1]*(- A[:,1]) .+
    ↪  v[20,1].*X[:,1]))
    for n = 2:N
        JuMP.@constraint(model, X[:,n] .== [X[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(- X[:,n]) .+ v[13,n].*X[:,n]))
        JuMP.@constraint(model, G[:,n] .== [G[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(G_f .- G[:,n]) .+ v[28,n].*X[:,n]))
        JuMP.@constraint(model, A[:,n] .== [A[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(- A[:,n]) .+ v[20,n].*X[:,n]))
    end

    # Duality constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] +
        ↪  transpose(mod.S)*lam[:,n] + umy[:,n] - lmy[:,n] .== 0)
        JuMP.@constraint(model, -transpose(mod.c)*v[:,n] +
        ↪  transpose(v[:,n])*W*v[:,n] == -transpose(v[:,n])*W*v[:,n] +
        ↪  transpose(lmy[:,n])*LB[:,n] - transpose(umy[:,n])*UB)
    end

    # MPC constraints:
    for n in 1:N
        JuMP.@constraint(model, D[n] >= u - 0.2)
        JuMP.@constraint(model, D[n] <= u + 0.2)
    end

    # Objective:
    JuMP.@NLobjective(model, Min, sum(sum((X[c,n] - sp)^2 for n = 1:N) for c =
    ↪  1:3) + 1e-10*(u - D[1])^2)

    JuMP.optimize!(model)

    return JuMP.value.(D)[1], JuMP.solve_time(model), JuMP.raw_status(model)
end
```

P-Duality.jl

P-Duality.jl is used to solve one MPC optimization with the penalized duality theory NLPA.

```
# Penalized duality theory MPC reformulation:

using Ipopt, JuMP

function PDual(mod, G_0, X_0, A_0, K_g, v_gmax, N, u, G_f, sp)

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Number of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
           1 t[3] t[3]^2
    ]
    M = M_1*M_2^(-1)

    # Creating W-matrix:
    W = zeros(Float64, j, j)
    for k = 1:j
        for l = 1:j
            if k == l
                W[k, l] = 1e-6
            end
        end
    end

    # Initilaizing UB:
    UB = mod.UB

    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "linear_solver", "ma97")
    set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
    set_optimizer_attribute(model, "acceptable_tol", 1e-8)
    JuMP.set_silent(model)

    # Variabels:
    JuMP.@variable(model, v[1:j, 1:N])   # Fluxes,       => Matrix:
    ↪  #reactions*#finite elements
    JuMP.@variable(model, G[1:3, 1:N])   # Glucose,      => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, X[1:3, 1:N])   # Biomass,      => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, A[1:3, 1:N])   # Acetate,      => Matrix: #collocation
    ↪  points*#finite elements
```

```
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,        => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds   => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, D[1:N])        # Dilution rate => Vector: #finite
↪  elements

# Defining LB:
for n = 1:N
    for k = 1:20-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[20,n] == -2.5*A[3,n]/(0.01 + A[3,n]))
    for k = 20+1:28-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[28,n] == v_gmax*G[3,n]/(K_g + G[3,n]))
    for k = 28+1:j
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
        JuMP.@constraint(model, lmy[k,n] >= 0)
        JuMP.@constraint(model, umy[k,n] >= 0)
    end
    for c = 1:3
        JuMP.@constraint(model, G[c,n] >= 0)
        JuMP.@constraint(model, X[c,n] >= 0)
        JuMP.@constraint(model, A[c,n] >= 0)
    end
    JuMP.@constraint(model, 0 <= D[n] <= 1)
end

# Orthogonal collocation constraints:
JuMP.@constraint(model, X[:,1] .== X_0 .+ dt*M*(D[1]*(- X[:,1]) .+
↪  v[13,1].*X[:,1]))
JuMP.@constraint(model, G[:,1] .== G_0 .+ dt*M*(D[1]*(G_f .- G[:,1]) .+
↪  v[28,1].*X[:,1]))
JuMP.@constraint(model, A[:,1] .== A_0 .+ dt*M*(D[1]*(- A[:,1]) .+
↪  v[20,1].*X[:,1]))
for n = 2:N
    JuMP.@constraint(model, X[:,n] .== [X[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(- X[:,n]) .+ v[13,n].*X[:,n]))
    JuMP.@constraint(model, G[:,n] .== [G[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(G_f .- G[:,n]) .+ v[28,n].*X[:,n]))
    JuMP.@constraint(model, A[:,n] .== [A[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(- A[:,n]) .+ v[20,n].*X[:,n]))
end
```

```
    # Duality theory Constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] +
        ↪  transpose(mod.S)*lam[:,n] + umy[:,n] - lmy[:,n] .== 0)
    end

    # MPC constraints:
    for n in 1:N
        JuMP.@constraint(model, D[n] >= u - 0.2)
        JuMP.@constraint(model, D[n] <= u + 0.2)
    end

    # Objective:
    JuMP.@NLexpression(model, FO,  sum(sum((X[c,n] - sp)^2 for n = 1:N) for c =
    ↪  1:3) + 1e-10*(u - D[1])^2 + 10*sum(sum(-mod.c[l]*v[l,n] +
    ↪  2*v[l,n]*W[l,l]*v[l,n] - lmy[l,n]*LB[l,n] + umy[l,n]*UB[l] for l in 1:j)
    ↪  for n in 1:N))
    JuMP.@NLobjective(model, Min, FO)

    JuMP.optimize!(model)

    return JuMP.value.(D)[1], JuMP.solve_time(model), JuMP.raw_status(model)
end
```

KKT.jl

KKT.jl is used to solve one MPC optimization with the non-penalized Karush–Kuhn–Tucker (KKT) NLPA.

```
# KKT MPC reformulation:

using Ipopt, JuMP

function KKT(mod, G_0, X_0, A_0, K_g, v_gmax, N, u, G_f, sp)

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Number of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
           1 t[3] t[3]^2
    ]
    M = M_1*M_2^(-1)

    # Creating W-matrix:
```

```
W = zeros(Float64, j, j)
for k = 1:j
    for l = 1:j
        if k == l
            W[k, l] = 1e-6
        end
    end
end

# Initilaizing UB:
UB = mod.UB

model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "linear_solver", "ma97")
set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
set_optimizer_attribute(model, "acceptable_tol", 1e-8)
JuMP.set_silent(model)

# Variabels:
JuMP.@variable(model, v[1:j, 1:N])    # Fluxes,        => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, G[1:3, 1:N])    # Glucose,       => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, X[1:3, 1:N])    # Biomass,       => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, A[1:3, 1:N])    # Acetate,       => Matrix: #collocation
↪  points*#finite elements
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,        => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, LB[1:j, 1:N])   # Lower bounds   => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, D[1:N])         # Dilution rate => Vector: #finite
↪  elements

# Defining LB:
for n = 1:N
    for k = 1:20-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[20,n] == -2.5*A[3,n]/(0.01 + A[3,n]))
    for k = 20+1:28-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[28,n] == v_gmax*G[3,n]/(K_g + G[3,n]))
    for k = 28+1:j
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
```

```
            JuMP.@constraint(model, lmy[k,n] >= 0)
            JuMP.@constraint(model, umy[k,n] >= 0)
        end
        for c = 1:3
            JuMP.@constraint(model, G[c,n] >= 0)
            JuMP.@constraint(model, X[c,n] >= 0)
            JuMP.@constraint(model, A[c,n] >= 0)
        end
        JuMP.@constraint(model, 0 <= D[n] <= 1)
    end

    # Orthogonal collocation constraints:
    JuMP.@constraint(model, X[:,1] .== X_0 .+ dt*M*(D[1]*(- X[:,1]) .+
    ↪  v[13,1].*X[:,1]))
    JuMP.@constraint(model, G[:,1] .== G_0 .+ dt*M*(D[1]*(G_f .- G[:,1]) .+
    ↪  v[28,1].*X[:,1]))
    JuMP.@constraint(model, A[:,1] .== A_0 .+ dt*M*(D[1]*(- A[:,1]) .+
    ↪  v[20,1].*X[:,1]))
    for n = 2:N
        JuMP.@constraint(model, X[:,n] .== [X[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(- X[:,n]) .+ v[13,n].*X[:,n]))
        JuMP.@constraint(model, G[:,n] .== [G[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(G_f .- G[:,n]) .+ v[28,n].*X[:,n]))
        JuMP.@constraint(model, A[:,n] .== [A[3,n-1] for z = 1:3] .+
        ↪  dt*M*(D[n]*(- A[:,n]) .+ v[20,n].*X[:,n]))
    end

    # Duality Constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] +
        ↪  transpose(mod.S)*lam[:,n] + umy[:,n] - lmy[:,n] .== 0)
    end

    # CS:
    for n = 1:N
        JuMP.@constraint(model, transpose(lmy[:,n])*(v[:,n] .- LB[:,n]) == 0)
        JuMP.@constraint(model, transpose(umy[:,n])*(UB .- v[:,n]) == 0)
    end

    # MPC constraints:
    for n in 1:N
        JuMP.@constraint(model, D[n] >= u - 0.2)
        JuMP.@constraint(model, D[n] <= u + 0.2)
    end

    # Objective:
    JuMP.@NLobjective(model, Min, sum(sum((X[c,n] - sp)^2 for n = 1:N) for c =
    ↪  1:3) + 1e-10*(u - D[1])^2)

    JuMP.optimize!(model)

    return JuMP.value.(D)[1], JuMP.solve_time(model), JuMP.raw_status(model)
end
```

P-KKT.jl

P-KKT.jl is used to solve one MPC optimization with the penalized KKT NLPA.

```
# Penalized KKT MPC reformulation:

using Ipopt, JuMP

function PKKT(mod, G_0, X_0, A_0, K_g, v_gmax, N, u, G_f, sp)

    X_0 = [X_0 for i = 1:3]           # Initial concentration of biomass, gDW/L
    G_0 = [G_0 for i = 1:3]           # Initial concentration of glucose, mmol/L
    A_0 = [A_0 for i = 1:3]           # Initial concentration of acetate, mmol/L
    t =  [0.155051, 0.644949, 1.0000] # Collocation points
    i = size(mod.S)[1]                # Number of metabolites
    j = size(mod.S)[2]                # Number of reactions

    # Creating M-matrix:
    M_1 = [t[1] 1/2*t[1]^2 1/3*t[1]^3
           t[2] 1/2*t[2]^2 1/3*t[2]^3
           t[3] 1/2*t[3]^2 1/3*t[3]^3
    ]
    M_2 = [1 t[1] t[1]^2
           1 t[2] t[2]^2
           1 t[3] t[3]^2
    ]
    M = M_1*M_2^(-1)

    # Creating W-matrix:
    W = zeros(Float64, j, j)
    for k = 1:j
        for l = 1:j
            if k == l
                W[k, l] = 1e-6
            end
        end
    end

    # Initilaizing UB:
    UB = mod.UB

    model = Model(Ipopt.Optimizer)
    set_optimizer_attribute(model, "linear_solver", "ma97")
    set_optimizer_attribute(model, "constr_viol_tol", 1e-10)
    set_optimizer_attribute(model, "acceptable_tol", 1e-8)
    JuMP.set_silent(model)

    # Variabels:
    JuMP.@variable(model, v[1:j, 1:N])   # Fluxes,       => Matrix:
    ↪  #reactions*#finite elements
    JuMP.@variable(model, G[1:3, 1:N])   # Glucose,      => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, X[1:3, 1:N])   # Biomass,      => Matrix: #collocation
    ↪  points*#finite elements
    JuMP.@variable(model, A[1:3, 1:N])   # Acetate,      => Matrix: #collocation
    ↪  points*#finite elements
```

```
JuMP.@variable(model, lam[1:i, 1:N]) # Lambda,        => Matrix:
↪  #metabolites*#finite elements
JuMP.@variable(model, lmy[1:j, 1:N]) # LB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, umy[1:j, 1:N]) # UB-my,         => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, LB[1:j, 1:N])  # Lower bounds   => Matrix:
↪  #reactions*#finite elements
JuMP.@variable(model, D[1:N])        # Dilution rate => Vector: #finite
↪  elements

# Defining LB:
for n = 1:N
    for k = 1:20-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[20,n] == -2.5*A[3,n]/(0.01 + A[3,n]))
    for k = 20+1:28-1
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
    JuMP.@NLconstraint(model, LB[28,n] == v_gmax*G[3,n]/(K_g + G[3,n]))
    for k = 28+1:j
        JuMP.@constraint(model, LB[k,n] == mod.LB[k])
    end
end

# Constraints for variables:
for n = 1:N
    for k = 1:j
        JuMP.@constraint(model, v[k,n] <= UB[k])
        JuMP.@constraint(model, LB[k,n] <= v[k,n])
        JuMP.@constraint(model, lmy[k,n] >= 0)
        JuMP.@constraint(model, umy[k,n] >= 0)
    end
    for c = 1:3
        JuMP.@constraint(model, G[c,n] >= 0)
        JuMP.@constraint(model, X[c,n] >= 0)
        JuMP.@constraint(model, A[c,n] >= 0)
    end
    JuMP.@constraint(model, 0 <= D[n] <= 1)
end

# Orthogonal collocation constraints:
JuMP.@constraint(model, X[:,1] .== X_0 .+ dt*M*(D[1]*(- X[:,1]) .+
↪  v[13,1].*X[:,1]))
JuMP.@constraint(model, G[:,1] .== G_0 .+ dt*M*(D[1]*(G_f .- G[:,1]) .+
↪  v[28,1].*X[:,1]))
JuMP.@constraint(model, A[:,1] .== A_0 .+ dt*M*(D[1]*(- A[:,1]) .+
↪  v[20,1].*X[:,1]))
for n = 2:N
    JuMP.@constraint(model, X[:,n] .== [X[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(- X[:,n]) .+ v[13,n].*X[:,n]))
    JuMP.@constraint(model, G[:,n] .== [G[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(G_f .- G[:,n]) .+ v[28,n].*X[:,n]))
    JuMP.@constraint(model, A[:,n] .== [A[3,n-1] for z = 1:3] .+
    ↪  dt*M*(D[n]*(- A[:,n]) .+ v[20,n].*X[:,n]))
end
```

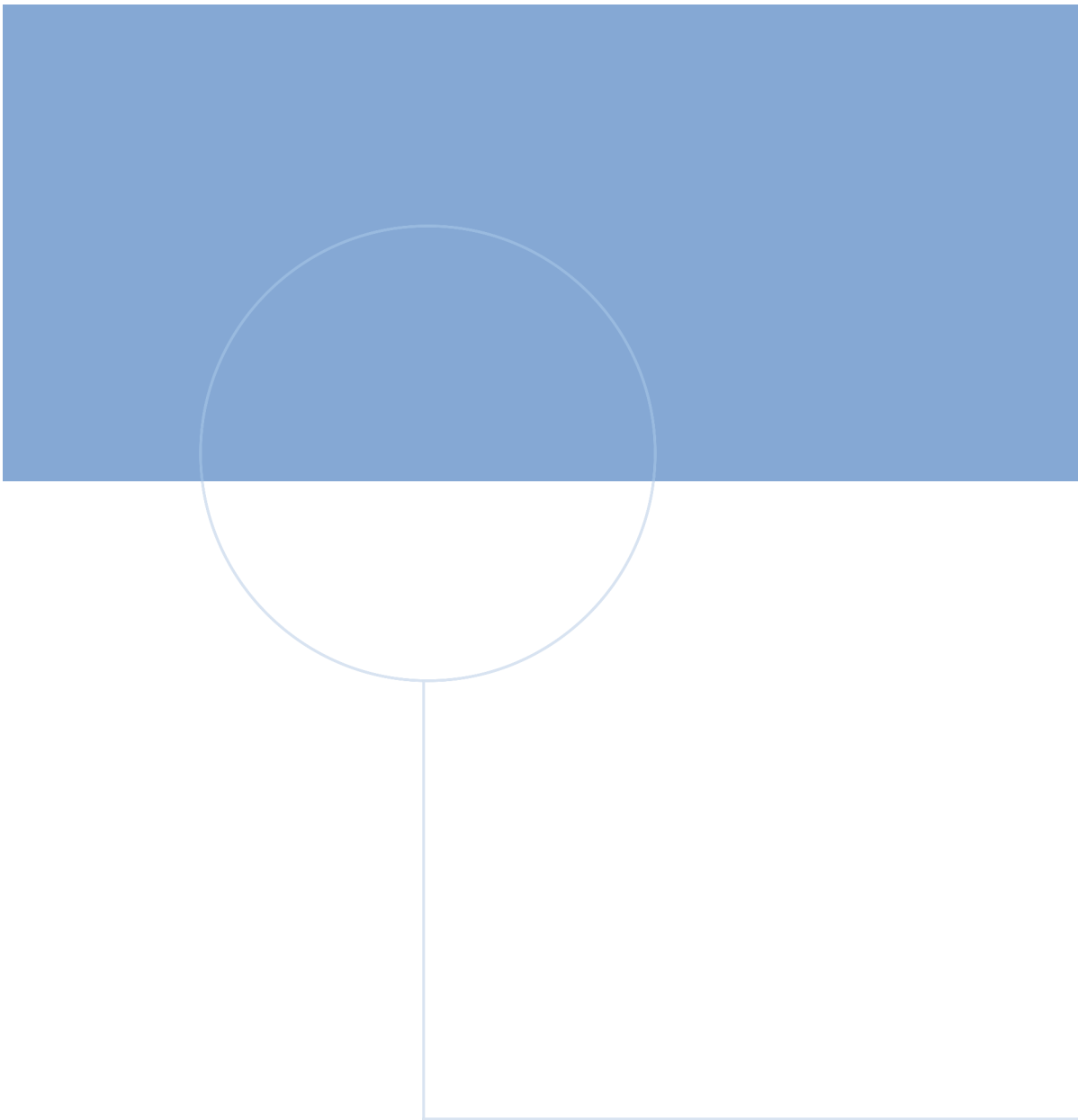
```
    # Duality Constraints:
    for n = 1:N
        JuMP.@constraint(model, mod.S*v[:,n] .== 0)
        JuMP.@constraint(model, -mod.c .+ 2 .*W*v[:,n] +
        ↪  transpose(mod.S)*lam[:,n] + umy[:,n] - lmy[:,n] .== 0)
    end

    # MPC constraints:
    for n in 1:N
        JuMP.@constraint(model, D[n] >= u - 0.2)
        JuMP.@constraint(model, D[n] <= u + 0.2)
    end

    # Objective:
    JuMP.@NLexpression(model, FO,  sum(sum((X[c,n] - sp)^2 for n = 1:N) for c =
    ↪  1:3) + 1e-10*(u - D[1])^2 + 1e-2*sum(sum(((v[l,n] - LB[l,n])*lmy[l,n]) +
    ↪  (-v[l,n] + UB[l])*umy[l,n] for l in 1:j) for n in 1:N))
    JuMP.@NLobjective(model, Min, FO)

    JuMP.optimize!(model)

    return JuMP.value.(D)[1], JuMP.solve_time(model), JuMP.raw_status(model)
end
```



NTNU

Norwegian University of
Science and Technology