CHALMERS

GÖTEBORGS UNIVERSITET

Dynamisk modellering av signalvägar som reglerar kol- och kvävemetabolismen i bagerijäst

Implementering, parameteruppskattning och analys av en dynamisk ODE-modell

Dynamical modeling of signaling pathways regulating carbon and nitrogen metabolism in baker's yeast

Implementation, parameter estimation and analysis of a dynamic ODE-model

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Leo Andrekson
Rakel Hellberg
Emma Johansson
Jesper Olsson

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

Dynamisk modellering av signalvägar som reglerar kol- och kvävemetabolismen i bagerijäst

Implementering, parameteruppskattning och analys av en dynamisk ODE-modell

Kandidatarbete i matematik inom civilingenjörsprogrammet Bioteknik vid Chalmers

Leo Andrekson Jesper Olsson

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Rakel Hellberg

Kandidatarbete i matematik inom civilingenjörsprogrammet Kemiteknik vid Chalmers

Emma Johansson

Handledare: Sebastian Persson

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

Förord

Vi vill tacka vår handledare Sebastian Persson som varit generös med sin tid och hjälpt oss mycket under projektet. Han har inte bara hjälpt oss genom att svara på våra frågor, vägleda projektet och tålmodigt hjälpa oss med våra textutkast, utan även genom att framföra presentationer om hur man genomför ett bra programmeringsprojekt och skapar tydliga figurer. Vi vill också tacka Isidora Kristofersdottir. Hon har främst hjälpt oss med att hitta kvalitativ data, men även genom att svara på frågor om signalvägarna som analyserats i projektet.

Detta kandidatarbete kan i stora drag delas i in i två delar; implementering och analys av ODE-modellen och parameteruppskattning. I början av projektet togs beslutet att dela upp gruppen i två delar, där Emma och Jesper började med implementering av modellen, medan Rakel och Leo istället arbetade med parameteruppskattningen, vilket inleddes med att implementera en optimeringsalgoritm från grunden. Denna uppdelning behölls under resten av projektet.

För att underlätta projektetarbetet har två olika typer av loggböcker förts. Den ena var en tidslogg för individuellt arbete och nedlagd tid som gruppmedlemmarna fyllde i själva. Den andra var en dagbok som användes som verktyg för att beskriva arbetets utveckling i allmänhet, vilka problem som uppstod och hur de löstes. Vi sammanfattade arbetet i dagboken veckovis utifrån tidsloggen och mötesanteckningar. Denna uppgift cirkulerades i gruppen, så att en person var ansvarig varje vecka. Utöver detta förde även gruppen ett eget mötesprotokoll på alla möten där det tydligt noteras vad som beslutats och var i arbete man befann sig vid mötestillfället samt eventuella mål till nästa tillfälle.

Den uppdelning av arbetet som gjordes - med implementering och analys av ODE-modellen och parameteruppskattning som de två huvuddelarna av projektet - speglar även uppdelningen som gjordes vid arbetet med rapporten. Ändå förekommer exempel där alla medlemmar har bidragit till vissa delar av rapporten, framförallt vid de tillfällen där arbetet överlappade.

Under projektet har även programmeringen delats upp mellan gruppmedlemmarna. Leo och Rakel har arbetat med att implementera en optimeringsalgoritm för att genomföra parameteruppskattningen, där de vidare delade upp arbetet i två olika metoder för beräkning av gradienten. Leo använde framåtriktad automatisk differentiering och Rakel känslighetsanalys, varav den förstnämnda användes i projektet. Utöver detta har Rakel också genomfört identifierbarhets- och känslighetsanalys av modellparametrarna. Jesper och Emma har arbetat med att implementera ODE-modellen och illustrera lösningar av ODE-modellen i jämförelse med data, samt genomfört simuleringar av modellen med erhållna parametrar för att analysera modellens giltighet utifrån kvalitativ data. Det samlade arbetet med programmeringen redovisas i appendix E och kan återfinnas i vår GitHub-mapp som nås via länken här. Webbadressen finns även utskriven i appendix E.1. I tabell 0.1 presenteras uppdelningen av arbetet fördelat mellan de olika författarna både för programmeringen och skrivandet i slutrapporten. I de fall där avsnitten återfinns för flera författare innebär detta att författarna kan anses ha bidragit lika mycket till det avsnittet.

Tabell 0.1: Bidragsrapport över arbetet för kandidatarbetsgruppen MVEX01-22-03.

Huvudförfattare	Avsnitt
Emma	Förord, Populärvetenskaplig presentation, 1, 1.1, Inledning till avsnitt 2, 2.1.2, Figur 2.1, 2.2, 3.1, 3.4, 4.1, 4.3, 5, 6, B, D.1, D.2. Kod: E.4.1, E.4.2, E.7.2, E.9, E.11.1
Jesper	1, 2.1.1, 2.1.2, 2.2, 4.3, 5, 6, D.2. Kod: E.4.3, E.5, E.7, E.9, E.11.2
Leo	Sammandrag, Abstract, 2.3.2, 2.3.4 - 2.3.7, 3.2, 3.3, 4.2, 5, C.1 - C.3. Kod: E.2, E.3, E.4.3, E.6
Rakel	2.3.1 - 2.3.4, 2.3.8, 4.2, 5, C.4. Kod: E.8

Populärvetenskaplig presentation

De flesta känner till åldersrelaterade sjukdomar som cancer och Alzheimers. Dessa sjukdomar är mycket obehagliga och många har någon personlig relation till dem; hur de har påverkat vänner, familj eller till och med en själv. De flesta är medvetna om att det pågår intensiv forskning i dessa områden, men färre känner till att kombinationen av matematik, biologi och bagerijäst kan bidra till att bota dessa sjukdomar.

Forskning på sjukdomar som cancer och Alzheimers görs till stor del på människoceller i klassisk labbmiljö, men det finns också andra organismer man kan forska på. Ett exempel är att utföra liknande studier på modellorganismer som bagerijäst. Med modellorganism menas en välstuderad art som används för att bedriva forskning vars slutsatser sedan kan appliceras på andra organismer, exempelvis människor. Precis som namnet antyder används bagerijäst bland annat för att baka bröd men den är också mycket användbar inom forskning.

En av huvudanledningarna till att bagerijäst används inom forskning är att det är en väldigt bra modellorganism med många eftertraktade egenskaper. Till exempel är det möjligt att studera åldersrelaterande sjukdomar eftersom det är möjligt att studera organismen under hela dess livstid. Det finns också vissa likheter mellan däggdjurscellers och bagerijästens metabolism och signalering, vilket gör den extra användbar för att exempelvis förstå hur metabolismen fungerar i människor. I det här sammanhanget innebär signalering hur cellen svarar på någon förändring i dess omgivning. Modellorganismer som bagerijäst behöver inte bara användas inom klassisk laborativ forskning utan kan också användas som modellorganism inom mer tvärvetenskapliga områden. Ett exempel på detta är kombinationen av matematik och biologi som tillsammans kallas systembiologi.

Inom systembiologi är det vanligt att studier genomförs på ett specifikt system i modellorganismens biologi. Ett exempel på ett system som kan studeras är någon eller några delar av ett signalnätverk hos en modellorganism. Ett signalnätverk har i uppgift att reglera hur signaler behandlas och tas emot inom cell vilket är viktigt för att förstå hur biologiska system reagerar på sin omgivning. Eftersom varje organism måste kunna reagera på sin omgivning är det inte förvånande att ett icke funktionellt signalnätverk är kopplat till sjukdomar som cancer eller Alzheimers.

Ett vanligt sätt att studera signalering och signalnätverk är som tidigare nämnt genom systembiologi. Det första steget i den systembiologiska processen är att samla data från någon modellorganism som exempelvis bagerijäst. Därefter används information från litteratur för att skapa en matematisk modell över signalnätverket, som sedan kan studeras under olika förhållanden. Den skapade modellen anpassas sedan till data för att försöka beskriva verkligheten. Analys av modellen genomförs därefter för att ta reda på vilka delar av systemet som stämmer överens med data. Utifrån denna analys görs sedan försök att förändra modellen tills den stämmer väl överens med data över flera olika fall. När detta väl är uppnått kan modellen användas för att förutspå hur bagerijästens signalnätverk beter sig i situationer som kan vara svåra att testa i experiment. Därmed kan man få kunskaper som annars hade varit svåra att få. Sammantaget, genom systembiologi är det möjligt att skapa en digital cell där det är mycket lättare att utforska hur alla komponenter i ett nätverk hänger samman.

I detta arbete har fokus varit på att analysera en modell som redan tagits fram av någon annan för att förstå hur bra modellen är. Detta gjordes genom att jämföra modellen med data som beskriver hur systemet beter sig. Som ett komplement till detta gjordes även parameteruppskattning på modellen. Syftet med den var att hitta värden på olika parametrar som beskriver experimentell data väl. Med parameter menas i detta sammanhang något som har ett konstant värde och spelar en avgörande roll för modellens resultat. Med en parameteruppskattning hittas parametrar som kan användas för att testa hur de påverkar modellen. Därmed får man förståelse om det är modellen eller kanske en samling parametrar som inte följer datan.

Det system som studerades i arbetet var de delar av bagerijästens signalnätverk som reglerar kol- och kvävemetabolismen. Systemet kan delas in i tre huvuddelar som kopplas samman utifrån hur signalmolekyler i dessa delar påverkar varandra när signaler går genom cellen. I ena änden av

signalnäverket påverkas molekylerna av att mängden kol och kväve förändras. Dessa delar tar alltså emot signalen och skickar vidare den till mitten. Där finns delar som utför regleringen av signalen. Dessa delar är mycket komplexa, men också viktiga. Slutligen skickas signalen vidare till de nedersta delarna, som orsakar ett svar på signalen från cellen. De slutliga målen för signalen består ofta av olika gener som förändrar cellen. Denna förändring kan sedan mätas under experiment. Modellen som analyserades i detta arbete försöker beskriva hur signalmolekylerna i dessa tre delar påverkar varandra.

Slutsatsen som drogs från analysen av modellen var att det fanns flera delar av modellen som inte överensstämde med data. Detta gällde för alla uppsättningar av parametervärden som testades. Modellen behöver därför förbättras för att kunna ge en bättre beskrivning av bagerijästens signalnätverk. Under arbetet gavs också exempel på vilka delar av modellen som framförallt behöver förbättras och hur detta kan genomföras. Alltså är det fortfarande oklart om just denna modell kan modifieras så att den kan ge information om signalnätverket som är svåråtkomlig på annat sätt.

Avslutningsvis är det däremot troligt att en bra och tillräckligt avancerad modell över bagerijästens signalnätverk kan användas för att få ny information om systemet. Vad dessa kunskaper i sin tur kan användas till är fortfarande ovisst, men det är troligt att de kan användas som hjälpmedel för att förstå sig på åldersrelaterade sjukdomar som cancer eller Alzheimers och i framtiden kanske till och med bota dessa sjukdomar.

Sammandrag

Förståelse för hur signalering fungerar inom och mellan celler är relevant för att förstå flera sjukdomar. En central process inom alla organismer är hur deras signalnätverk reagerar baserat på ändringar av näringstillgångar som exempelvis tillgången på kol och kväve. Signalvägarna TORC1, SNF1 och cAMP-PKA i bagerijäst är centrala för reglering av kol- och kvävemetabolismen. I detta projekt undersöks dessa signalvägar genom att använda en dynamisk ODE-modell från artikeln skriven av Jalihal m.fl. [1]. Modellen implementerades och ett urval av resultaten från Jalihal m.fl. återskapades för att bekräfta implementeringen. Modellen består av ett flertal parametrar som skattas med syftet att ge en bättre beskrivning av experimentell data. Detta görs genom användning av en kvasi-Newtonalgoritm samt beräkning av identifierbarhet och känslighet för parametrarna.

Två parametervektorer erhölls som ger en bättre beskrivning av experimentell data. Slutsatsen som drogs från identifierbarhets- och känlighetsanalys var att majoriteten av parametrarna har låg känslighet. Kvalitativ data används för att undersöka modellens giltighet för scenarion där ingen kvantitativ data finns tillgänglig. Detta gjordes för de parametervektorer som erhölls från parameteruppskattingen och parametervektorn rapporterad av Jalihal m.fl. [1]. Resultatet från denna analys var att de flesta scenarion som analyserades överensstämde med kvalitativ data, men att det även fanns scenarion där detta inte var fallet. För dessa fall föreslogs experiment som kan förbättra och utöka modellen. Trots de brister modellen uppvisar är den ändå användbar som en grund för vidareutveckling. Åtgärder som utvidgning av modellen kombinerat med nya parameteruppskattningar kan på sikt leda till en modell som beskriver signalvägarna i bagerijäst väl.

Abstract

Understanding how signaling functions within and between cells is relevant to get a better grasp of several diseases. A central signaling process in organisms ranging from everyday baker's yeast to humans is the nutrient signaling network, which is responsible for sensing availability of nutrients, such as carbon and nitrogen. The TORC1, SNF1 and cAMP-PKA signaling pathways are central components of the nutrient signaling network in baker's yeast, which regulates the metabolism based on nutrient levels. In this project, we investigated these pathways by implementing and analyzing an ODE model of said pathways from Jalihal et al. [1]. As a first step, the model was implemented and some results from Jalihal et al. were recreated in order to confirm the implementation. Furthermore, the model consists of several unknown parameters that have to be estimated. This was done using a quasi-Newton algorithm.

Two parameter vectors were obtained which give a better description of data than the parameter vector reported by Jalihal et al. The conclusion drawn from the identifiability and sensitivity analysis is that the majority of the parameters have low sensitivity. Qualitative data was used to examine the model's validity for scenarios where no quantitative data is available. This was done for the parameter vectors obtained from the parameter estimation and Jalihal et al. [1]. Most of the scenarios analyzed were consistent with qualitative data, except for a few cases. For these cases, experiments were proposed that could improve and expand the model. Despite the shortcomings of the model, it is still useful as a basis for further development. Measures such as extending the model, combined with new parameter estimations, may in the long run lead to a well-describing model of the nutrient signaling pathways in baker's yeast.

Innehåll

1 Inledning

För att behandla, bota och förebygga åldersrelaterande sjukdomar som till exempel cancer krävs förståelse om hur biologiska system fungerar på cellnivå. För att få denna ökade förståelse krävs forskning på de grundläggande system som styr hur biologiska celler känner av och svarar på sin omgivning. Generellt sett görs denna forskning på modellorganismer. Detta eftersom det kan vara etiskt problematiskt samt svårt att genomföra forskning på människor, men också för att modellorganismer är välstuderade. Med välstuderad menas här att det finns mycket kunskap om komponenterna inom organismen. Däremot brukar dynamiken mellan dessa komponenter vara mer okänd och intressant att studera vidare.

Saccharomyces cerevisiae, även känd som bagerijäst, är ett exempel på en välstuderad modellorganism. Det är en encellig organism med ett flertal användningsområden, främst i biotekniska processer som produktion av läkemedel och livsmedel [2]. Bagerijäst är även användbar inom forskning om åldersrelaterade sjukdomar, då det är möjligt att studera organismerna under hela dess livscykel [3]. Samtidigt kan mer generella slutsatser dras från studier på bagerijäst eftersom vissa delar av dess metabolism och signalering är evolutionärt bevarad hos andra avancerade organismer, som exempelvis däggdjur [4]. Dessa studier går ofta ut på att man utsätter cellen eller dess omgivning för någon typ av förändring och mäter hur det påverkar cellerna.

Ett exempel på en förändring som kan göras i cellens omgivning är av tillgången till kväve eller kol, vilket orsakar förändringar i cellens beteende. Denna förändring kan beskrivas som en signal till cellen. Reglering av dessa signaler sker via signalvägar som skickar vidare signalen genom cellen till specifika mål, vilka sedan orsakar cellens reaktion. Regleringen av näringssignaler är dock en komplex process där olika signalvägar överlappar med varandra. Detta gör även processen dynamisk.

Tre viktiga komponenter i bagerijästens näringssignalnätverk för reglering av kol- och kvävemetabolism är signalvägarna TORC1, SNF1 och cAMP-PKA [5]. De generella rollerna hos signalvägarna är välkända. Däremot är det mindre känt hur dessa signalvägar interagerar och kommunicerar med varandra samt hur signalnätverket som helhet beter sig under olika förutsättningar [1]. Detta beror främst på att de olika signalvägarnas interaktion ger upphov till en dynamik i systemet vilket gör det svårare att förstå signalnätverket på en detaljerad nivå [6]. Det är dessa delar av processen som är mindre förstådda och kräver ytterligare undersökning för en mer komplett förståelse av näringssignalering.

Matematiskt modellering via ordinära differentialekvationer har i flera fall visat sig vara användbart för att förstå komplexa signalnätverk. Ofta kan komponenterna i nätverken och deras beteende approximeras med ODE:er [7]. Detta innebär att en dynamisk modell i form av ett system av ODE:er är användbar för att få förståelse för hur komplexa biologiska system - såsom interaktion mellan signalvägar - fungerar [8]. Mer specifikt används ODE:er för att simulera de kemiska reaktionerna som sker i signalvägarna och därmed beskriva systemets komplexitet och hur det reagerar på yttre förändringar i form av näringssignaler [8].

Eftersom det system som ska modelleras är mycket komplext och dynamiskt är det svårt att formulera en modell som beskriver systemet väl. Därför är en viktig del i arbete med modeller att analysera dess giltighet under olika förutsättningar och jämföra med kvalitativ data från litteratur och experimentell forskning. Sådant arbete kan ge information om vilka delar av modellen som eventuellt behöver förändras för att överensstämma med experimentell data över systemet.

En förutsättning för att kunna analysera en modell är att alla dess parametervärden är kända. Generellt sett uppskattas parametrar i en ODE-modell från data, genom så kallad parameteruppskattning. Denna uppskattning gör det möjligt att se hur olika uppsättningar av dessa parametervärden påverkar modellen. Således kan man förstå om modellen i sig inte representerar experimentell data eller om just specifika parametervektorer passar datan sämre än andra. Utifrån dessa resultat kan även slutsatser dras om vad ytterligare undersökning av systemet bör fokusera på.

1.1 Syfte och avgränsningar

Syftet med detta projekt var att implementera en dynamisk ODE-modell över signalnätverket hos bagerijäst och anpassa denna till experimentell data. Detta för att analysera modellens giltighet för olika näringsförhållanden och parametervektorer och vid behov föreslå experiment som kan genomföras för att förbättra modellen. De parametrar som analyserades erhölls från parameteruppskattning. Syftet med parameteruppskattningen var därmed att implementera en optimeringsalogritm för att hitta parametervektorer som beskriver experimentell data väl. Modellen som analyserades i arbetet hämtades från Jalihal m.fl. och förändrades inte under arbetet utan implementerades i sin ursprungliga form [1].

Därmed är en av de avgränsningar som gjorts för att genomföra projektet att modellen som användes inte utökades. Den experimentella data som användes vid parameteruppskattingen erhölls inte från egen litteraturstudie eller laborativt arbete, utan hämtades från de referenser som Jalihal m.fl. använt vid sin framställning och analys av modellen [1]. En annan avgränsning som gjordes var att den ODE-lösare som användes inte var egenkonstruerad.

2 Teori

Det forskningsfält där matematik och biologi kombineras för att studera komplexa biologiska system kallas systembiologi [9]. En viktig gren inom systembiologi är analys av signalnätverk. Arbetssättet inom systembiologi utgår från att experimentell data över systemet samlas in och sedan används för att skapa en matematisk modell. Denna modell anpassas sedan till data och ger parametervärden som kan verifieras med experiment. Experimentresultaten informerar eventuella ändringar av modellen och arbetscykeln börjar om tills en modell som korrekt beskriver systemet och dess dynamik uppnås. När modellen beskriver experimentella data väl kan den sedan användas för att förutspå egenskaper av systemet som kan vara svåra att observera experimentellt. På så vis kan man inhämta information om systemet som annars är svår att ta del av.

Den teori som är relevant för att genomföra projektet kan framförallt delas in i tre huvuddelar: biologi över signalvägar hos bagerijäst, matematiken bakom modellering med ordinära differentialekvationer och matematiken bakom optimering av funktioner som beror på en lösning av ett ODE-system. Dessa delar presenteras nedan i separata avsnitt.

2.1 Näringssignalering och signalvägar i bagerijäst

Nedan presenteras biologin över näringssignalering i *Saccharomyces cerevisiae*, även känd som bagerijäst. I det första avsnittet presenteras de generella principerna för hur näringssignaler tas emot och förs vidare genom cellen. Därefter följer en beskrivning av signalvägarna som reglerar kol- och kvävemetabolisen i bagerijäst och som ligger till grund för modellen som används under arbetet.

2.1.1 Generella principer för näringssignalering

Ett signalnätverk beskriver hur den enskilda cellen tar emot och för vidare information från sin omgivning. Ett exempel på vad cellen känner av är näringsförhållanden i form av koncentrationer av näringsämnen. Signalnätverkets behandling av information sker i tre steg [10]. I det första steget binder en signalmolekyl till en receptor, vilket är ett protein på cellens yta. Receptorn blir då aktiverad och skickar vidare signalen inom cellen. Inuti cellen aktiveras en eller flera av de signalvägar som utgör nätverket. Dessa signalvägar kommer i det andra steget att ta emot signalen och skicka vidare den genom ett flertal mellansteg till olika mål, även dessa i form av proteiner. Det tredje steget är en respons som målproteinerna reglerar, till exempel förändring av uttrycksnivån för en gen [10]. Förändringar som sker i omgivningen orsakar alltså en reaktion hos cellen. Det är signalnätverket som reglerar denna process.

2.1.2 Signalvägar som reglerar kol- och kvävemetabolismen i bagerijäst

Med bakgrund av de generella principerna för näringssignalering som beskrivs ovan följer nedan en beskrivning av de tre signalvägarna som utgör signalnätverket som analyseras i detta arbete. De signalvägar som betraktas är SNF1, TORC1 och cAMP-PKA, vilka reglerar kol- och kvävemetabolismen i bagerijäst [5]. Benämningen av signalvägarna kommer från huvudregulatorerna.

SNF1 är en av signalvägarna i bagerijäst och spelar en viktig roll i kolmetabolismen. Den huvudsakliga funktionen sker under förhållanden där det råder brist på glukos, vilken är den främsta kolkällan för cellen. SNF1-signalvägen består främst av ett komplex av proteiner (kallad Snf1) som aktiveras av signalmolekylerna Sak1, Tos3 och Elm3, och reglerar upptaget av alternativa kolkällor [1]. I detta fall är alltså Snf1 huvudregulatorn.

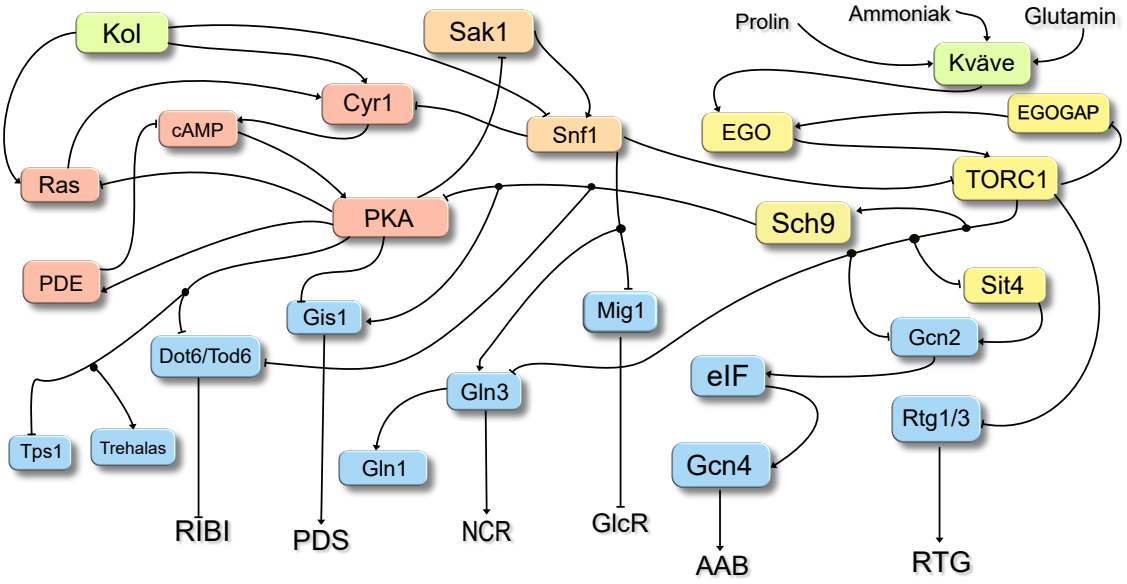
Vid höga nivåer av glukos är signalvägen cAMP-PKA relevant för kolmetabolismens aktivitet. cAMP är det signalämne som aktiverar PKA. PKA reglerar i sin tur hämning av kolupptagning samt främjande av celltillväxt. Bildning av cAMP styrs främst av enzymet Cyr1 medan nedbrytning styrs av PDE, som även aktiveras av PKA. Således inhiberar PKA indirekt sig själv [1].

Den tredje och sista signalvägen som är relevant i detta projekt är TORC1, som har funktionen att reglera kvävemetabolismen. Den består av kärnkomplexet TORC1 som regleras av ett flertal signalämnen, men främst av EGO-komplexet [1].

De tre signalvägarna påverkar varandra. SNF1 och cAMP-PKA har en ömsesidigt hämmande påverkan på varandra vilken härstammar naturligt från att de fyller en motsatt funktion i cellens näringssvar. Detta sker genom att Snf1 (signalmolekylen, inte signalvägen) inhiberar Cyr1, vilket har en inhiberande effekt på PKA. PKA inhiberar i sin tur Sak1, som har en inhiberande effekt på Snf1. Relationen mellan TORC1 och SNF1 är inte helt fastställd. SNF1 har en inhiberande effekt på TORC1[1], men TORC1 tros även påverka SNF1 [11]. Sch9 är ett signalämne som aktiveras av TORC1 och har en inhiberande effekt på PKA [1].

För en överblick av ovan beskrivet signalnätverk presenteras nedan i figur 2.1 en illustration av signalnätverket som reglerar kol- och kvävemetabolismen i bagerijäst. Nätverket består av de ovan beskrivna signalvägarna, nedströmssvaren samt slutmål för signalerna. I figuren presenteras alla komponenter i modellen och där visas hur de samverkar med övriga komponenter och vad som påverkar dem. Distinktionen mellan den inaktiva och aktiva formen av en signalmolekyl har uteslutits i syftet att förenkla figuren. Med aktiv form menas den form av signalmolekylen som utför en funktion i signalvägen, exempelvis genom att aktivera en annan signalmolekyl. Färgerna hos signalmolekylerna indikerar att de hör ihop. De signalmolekyler som tillhör signalvägen SNF1 är orange medan de som tillhör cAMP-PKA är röda och TORC1 är gula. De blå komponenterna representerar nedströmssvar och diskuteras mer utförligt nedan. Nedströmssvaren reglerar komponenterna längst ner i figuren (RIBI, PDS, NCR, GIcR och RTG) som är gener (eller samlingsnamn för en grupp gener) och de slutgiltiga målen i signalnätverket. Signalen kommer leda till en förändring av uttrycket av dessa gener och detta är cellens svar på näringssignalen.

Signalvägarna har i slutändan syftet att reglera det signalsvar som orsakar en förändring i cellens beteende. Cellens svar på en förändring i omgivningen sker oftast genom att ändra uttrycksnivån av en gen. De delar av signalnätverket som reglerar den slutgiltiga responsen kallas nedströmssvar (eng. *downstream responses*). Signalnätverket beskrivet ovan består av ett flertal nedströmssvar där Dot6/Tod6, Gln3, Mig1 och Rtg1/3 är några av de viktigaste. Funktionerna hos de svar som dessa molekyler reglerar är olika. Till exempel aktiverar Gln3 en samling av gener kallade NCR, eftersom de ansvarar för kvävekatabolitrepression (eng. *Nitrogen Catabolite Repression*). Detta innebär att de styr vilka kvävekällor som upptas [12].



Figur 2.1: Illustration av signalnätveket i bagerijäst. I den övre delen av figuren återfinns näringsämnena som utgör kol- och kvävekällor till cellen, representerade av grön färg. Övriga färgade komponenter är signalmolekyler medan de svarta komponenterna nederst i figuren representerar gener eller grupper av gener. cAMP-PKA signalvägen representeras av röd färg, medan SNF1-signalvägen syns i orange och TORC1 i gul. De blå signalmolekylerna är nedströmssvar och ansvarar för att reglera generna, som återfinns nederst i figuren. De två olika formerna av pilar representerar aktivering eller hämning av målet.

2.2 Modellering av signalvägar med ordinära differentialekvationer

För att modellera signalnätverket som beskrivs ovan krävs ett modellverktyg som kan beskriva systemets ändring i tiden under olika förutsättningar. Dessa tillståndsförändringar kan exempelvis vara svältning eller tillsättning av olika kol- och kvävekällor. Hur tillståndsförändringarna påverkar systemet kan modelleras med ordinära differentialekvationer.

De ordinära differentialekvationer (ODE:er) som används för att modellera metabolismen kan utformas på flera olika sätt beroende på vilken del av systemet som modelleras. Gemensamt för ODE:erna är att de byggs upp utifrån kinetiken hos de reaktioner som sker i signalvägen för den del av metabolismen som ska modelleras [13, 14]. En jämviktsreaktion beskrivs vanligtvis med massverkans lag enligt

$$\mathrm{A} \underset{\mathrm{k}_b}{\overset{\mathrm{k}_f}{\rightleftharpoons}} \mathrm{B},$$

där A är reaktanten, som förbrukas med hastighetskonstanten k_f och bildas med hastighetskonstanten k_b. B är en produkt som alltså bildas med hastighetskonstanten k_f och förbrukas med hastighetskonstanten k_b [13]. Med jämviktsreaktion menas att det råder kemisk jämvikt mellan komponenterna A och B. Detta innebär att hastigheten för den framåtgående reaktionen k_f och bakåtgående reaktionen k_b är lika. Därmed är koncentrationerna av A och B konstanta när jämvikten ställt in sig efter en viss tid [15]. I detta exempel ger massverkans lag en beskrivning av hur koncentrationerna förändras över tid enligt

$$\frac{\mathrm{d[B]}}{\mathrm{d}t} = k_f[\mathrm{A}] - k_b[\mathrm{B}], \tag{1}$$

där notationen [B] innebär koncentrationen av ämne B.

Modellen som används i detta arbete inkluderar några ekvationer som följer dynamiken beskriven med massverkans lag. Dessa ekvationer modellerar främst långsamma reaktioner. Det finns däremot andra ekvationer som modellerar snabbare reaktioner, vilket kan vara användbart om

man som i detta fall modellerar flera olika signalmolekyler med olika kinetik. Samlingsnamnet för denna typ av modellering som inkluderar reaktioner med både snabb och långsam kinetik kallas standardkomponentmodellering (eng. *Standard Component Modeling*) och kan förkortas SCM [14]. Fördelen med SCM är att det minskar antalet parametrar som behövs för att beskriva systemet.

I SCM kallas de ODE:er som beskriver långsamma reaktioner *Klass I*, vilka utformas på samma sätt som massverkas lag enligt exemplet i ekvation (1). Utöver dessa reaktioner finns som tidigare nämnt även reaktioner med snabbare kinetik som lämpligen modelleras med ODE:er av *Klass II* eller *Klass III* beroende på reaktionens karaktär. Klass II är en lämplig modell för de flesta reaktioner som modelleras i projektet, men vissa av reaktionerna modelleras med Klass III. I Appendix A återfinns alla ordinära differentialekvationer som används i modellen, generellt kategoriserade efter vilken roll de spelar i signalnätverket.

Ekvationerna som modellerar Klass II formuleras enligt

$$\frac{\mathrm{d}[P]}{\mathrm{d}t} = \gamma([P_T]\mathcal{H}(\sigma, W) - [P]),$$

där $[P_T]$ är den totala koncentrationen av ämne P, dvs. för både inaktiv och aktiv form, medan $[P]$ är koncentrationen av den aktiva formen av ämnet P efter tiden t. γ är en skalningsfaktor för uttrycket, som styr hur snabbt reaktionen går [14]. Funktionen benämnd $\mathcal{H}$ kallas soft-Heaviside-funktion och definieras enligt

$$\mathcal{H}(\sigma, W) = \frac{1}{1 + e^{-\sigma W}},$$

där σ är en skalningsfaktor för uttrycket och W är en viktad summa av koncentrationerna av de ämnen som deltar i reaktionen. Skalningsfaktorn σ kan vara experimentellt eller teoretiskt uppmätt [14]. W beräknas enligt

$$W = \omega_0 + \sum_{i=1}^{N} \omega_i [X_i], \tag{2}$$

där $[X_i]$ är koncentrationer och w_i är vikter för de ingående ämnena i [14].

De reaktioner som sker snabbt och exempelvis modellerar hur proteinkomplex bildas beskrivs av ekvationen för Klass III enligt

$$C = \min([X], [Y]),$$

där X och Y är koncentrationerna av de ingående komponenterna i proteinkomplexet C [14].

2.3 Parameteruppskattning

En ODE-modell som formulerats enligt det ramverk som presenteras i föregående avsnitt ger upphov till ett flertal parametrar, till exempel ω_i i ekvation (2). Det är av intresse att hitta värden på dessa parametrar som ger en bra beskrivning av befintlig data. Den processen kallas parameteruppskattning och är ett optimeringsproblem där kostnadsfunktionen ska indikera hur väl modellen för en given parametervektor beskriver verkligheten. I samband med en parameteruppskattning görs även identifierbarhets- och känslighetsanalys för att avgöra hur noga parametrarna går att bestämma utifrån modell och data.

Det finns flera metoder för att lösa optimeringsproblemet. I detta arbete används en kvasi-Newton-algoritm eftersom problemet inte kan lösas analytiskt. Generellt sett genomförs optimering i flera steg. I det första steget görs en startgissning. Därefter bestäms en sökriktning och sedan steglängd. För den nya parametervektorn kontrolleras termineringskriterium och om detta inte uppfylls så bestäms en ny sökriktning och så vidare.

2.3.1 Kostnadsfunktion

En vanlig kostnadsfunktion för att beskriva avvikelsen mellan modellen och observerad data är likelihood-funktionen. Vi använder fetstilt notation för vektorer. För parametrar $\mathbf{p}$, observationer

$y_i(t_j)$ och prediktioner $\hat{y}_i(\mathbf{p}, t_j)$ (utsignal y_i vid tidpunkt t_j) är likelihood-funktionen $\mathcal{L}(\mathbf{p}) = \mathbb{P}(\hat{\mathbf{y}} = \mathbf{y}|\mathbf{p})$, alltså sannolikheten att observera $\mathbf{y}$ givet parametervärdena $\mathbf{p}$. Antag nu att vår observerade data har oberoende, normal- och likafördelat fel med standardavvikelse $\sigma \in \mathbb{R}$, så att $y_i(t_j) \sim N(\hat{y}_i(t_j), \sigma) \ \ \forall i, j$. Då är täthetsfunktionen

$$f_N(\mathbf{p}, y_i(t_j)) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i(t_j) - \hat{y}_i(\mathbf{p}, t_j)}{\sigma}\right)^2}.$$

Oberoendet implicerar att $\mathcal{L}(\mathbf{p}) = \Pi_i \Pi_j f_N(\mathbf{p}, y_i(t_j))$, vilket i sin tur implicerar att

$$\mathcal{L}(\mathbf{p}) \propto e^{-\frac{1}{2}\left(\frac{y_i(t_j) - \hat{y}_i(\mathbf{p}, t_j)}{\sigma}\right)^2}.$$

För att förenkla högerledet använder vi istället

$$f(\mathbf{p}) = \sum_{i=1}^{I} \sum_{j=1}^{J_i} \left(y_i(t_j) - \hat{y}_i(\mathbf{p}, t_j)\right)^2 \propto -\ln(\mathcal{L}(\mathbf{p})), \tag{3}$$

där I är antalet utsignaler och J_i är antalet observationer för den utsignalen, som kostnadsfunktion.

2.3.2 Kvasi-Newtonmetoder

När kostnadsfunktionen har definierats kan ett optimeringsproblem formuleras där målet är att minimera den valda kostnadsfunktionen $f(\mathbf{p})$. Med Newtons metod förenklar man optimeringsproblemet genom att i varje iteration minimera en kvadratisk approximation av funktionen istället, nämligen andra ordningens Taylorutveckling $T(\mathbf{s}_k)$ [16]. Denna beräknas enligt

$$f(\mathbf{p}_k + \mathbf{s}_k) \approx T(\mathbf{s}_k) = f(\mathbf{p}_k) + \nabla f(\mathbf{p}_k)^T \mathbf{s}_k + \frac{1}{2}\mathbf{s}_k^T \nabla^2 f(\mathbf{p}_k)\mathbf{s}_k,$$

där $\mathbf{s}_k = \mathbf{p}_{k+1} - \mathbf{p}_k$, det vill säga ett steg. Eftersom Taylorutvecklingen är kvadratisk finns extremvärdet där dess gradient är noll. Därför bestäms steget eller sökriktningen $\mathbf{s}_k$ genom att sätta

$$\frac{d}{d\mathbf{s}_k} T(\mathbf{s}_k) = \nabla f(\mathbf{p}_k) + \nabla^2 f(\mathbf{p}_k)\mathbf{s}_k = 0.$$

För enklare notation benämner vi hädanefter hessianen $H_k = \nabla^2 f(\mathbf{p}_k)$. Sökriktningen enligt Newtons metod blir alltså $\mathbf{s}_k = -H_k^{-1} \nabla f(\mathbf{p}_k)$ [16].

I kvasi-Newtonmetoder väljs en steglängd α så att ett steg istället blir $\alpha\mathbf{s}_k$, och hessianen H_k approximeras för att undvika dyra beräkningar av andraderivator. Vi väljer att approximera hessianen med den välanvända BFGS-metoden (Broyden-Fletcher-Goldfarb-Shanno) [16]. Den första approximationen B_0 sätts till identitetsmatrisen och följande B_k beräknas enligt

$$B_{k+1} = B_k - \frac{(B_k\mathbf{s}_k)(B_k\mathbf{s}_k)^T}{\mathbf{s}_k^T B_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}, \tag{4}$$

där $\mathbf{y}_k = \nabla f(\mathbf{p}_{k+1}) - \nabla f(\mathbf{p}_k)$.

2.3.3 Steglängd

När sökriktning bestämts önskar vi hitta en steglängd α. Eftersom målet är att minimera kostnadsfunktionen önskar vi här

$$\min \ f(\mathbf{p}_k + \alpha\mathbf{s}_k),$$

med steglängden $\alpha \geq 0$ [16]. Detta är av intresse då en minimering av värdet på kostnadsfunktionen önskas i den sökriktning som beräknats, det vill säga vi gör en linjesökning. En steglängd sådan att

$$\nabla f(\mathbf{p}_k + \alpha\mathbf{s}_k)^T \mathbf{s}_k \geq 0 \text{ och} \tag{5}$$

$$\alpha \nabla f(\mathbf{p}_k + \alpha\mathbf{s}_k)^T \mathbf{s}_k = 0, \tag{6}$$

vore därför optimal [16]. Om ekvation (5) inte uppfylls så kan man gå längre i samma riktning och få minskad kostnad. Ekvation (6) uppfylls antingen om $\alpha = 0$ eller om sökriktningen är ortogonal mot gradienten i slutet av steget. Då är det dags att ändra riktning.

Å andra sidan är det sällan optimum verkligen ligger längs just den linjen. Därför är det lämpligt att använda en beräkningsmässigt enklare, approximativ metod. Vi väljer att använda Armijos steglängdsregel. För små $\alpha > 0$ kan $f(\mathbf{p}_k + \alpha\mathbf{s}_k)$ approximeras med hjälp av dess gradient i $\mathbf{p}_k$ enligt

$$f(\mathbf{p}_k + \alpha\mathbf{s}_k) \approx f(\mathbf{p}_k) + \alpha\nabla f(\mathbf{p}_k)^T\mathbf{s}_k,$$

vilket medför att

$$f(\mathbf{p}_k + \alpha\mathbf{s}_k) - f(\mathbf{p}_k) \approx \alpha\nabla f(\mathbf{p}_k)^T\mathbf{s}_k. \tag{7}$$

Armijos regel går ut på att välja en steglängd så att vänsterledet i ekvation (7) minskar med minst en bråkdel μ av högerledet, det vill säga att hitta något α så att

$$f(\mathbf{p}_k + \alpha\mathbf{s}_k) - f(\mathbf{p}_k) \leq \mu\alpha\nabla f(\mathbf{p}_k)^T\mathbf{s}_k, \tag{8}$$

uppfylls [16]. Typiskt väljs något litet $\mu \in (0, 1)$ och först $\alpha = 1$ [16]. Om α är för stort så provar man $\frac{\alpha}{2}$, och så vidare.

Så länge $\mathbf{s}_k$ är en nedåtgående riktning, dvs $\nabla f(\mathbf{p}_k)^T\mathbf{s}_k < 0$, är det möjligt att hitta ett tillräckligt litet steg för att regeln ska gälla. Det kan dock ibland gå långsamt, vilket innebär att stegen blir väldigt små med väldigt liten förbättring av målvärdet. För att avhjälpa detta kan man även kräva att

$$|\nabla f(\mathbf{p}_k + \alpha\mathbf{s}_k)^T\mathbf{s}_k| \geq \nu|\nabla f(\mathbf{p}_k)^T\mathbf{s}_k|, \tag{9}$$

för något $\nu \in [0, 1)$. Detta försäkrar att lutningen minskar tillräckligt [16]. Villkoren i ekvation (8) och (9) utgör tillsammans de så kallade Wolfe-villkoren.

2.3.4 Termineringskriterier

Efter varje steg med kvasi-Newtonmetoden bestäms en ny sökriktning, och ett nytt steg tas med förhoppningen att komma nära ett optimalt värde. Dock är det osannolikt att man kommer hitta ett optimum inom en rimlig tidsram. På grund av detta bör man införa termineringskriterier så att iterationen upphör när optimeringen ger en tillräckligt bra approximation. Tre lämpliga kriterier är

$$||\nabla f(\mathbf{p}_k)|| \leq \epsilon_1(1 + |f(\mathbf{p}_k)|), \tag{10}$$

$$f(\mathbf{p}_{k-1}) - f(\mathbf{p}_k) \leq \epsilon_2(1 + |f(\mathbf{p}_k)|), \tag{11}$$

$$||\mathbf{p}_{k-1} - \mathbf{p}_k|| \leq \epsilon_3(1 + ||\mathbf{p}_k||), \tag{12}$$

där $\epsilon_1, \epsilon_2, \epsilon_3 > 0$ är små konstanter [16]. Ekvationerna uppfylls där funktionen är tillräckligt platt. För att terminera iterationsprocessen bör två av ekvationerna (10)-(12) uppfyllas.

2.3.5 Brantaste nedstigningsmetoden

Ibland uppstår situationer där två av termineringskriterierna som definierats i avsnitt 2.3.4 uppfylls innan man kommit tillräckligt nära ett optimum. Om kostnadsfunktionen är väldigt platt i den sökriktning som beräknats med kvasi-Newton så blir steglängden α väldigt liten, eller i värsta fall hittas inget α som uppfyller Wolfe-villkoren (8) och (9). Då är sannolikheten stor att sökningen termineras innan man kommit nära ett optimum eftersom termineringskriteriet i ekvation (12) uppfylls och således behöver bara ytterligare ett kriterium uppfyllas för terminering. För att kringgå detta kan man använda någon annan metod som exempelvis den enkla brantaste nedstigningsmetoden (eng. *steepest descent*) ifall $\alpha < \epsilon_3$. Om steglängden ändå blir mindre än ϵ_3 bör vi vara nära ett optimum och därmed är det helt rätt att gå vidare till termineringskriterierna.

Brantaste nedstigningsmetoden bygger på att hitta ett optimum genom att följa gradientriktningen [16], alltså väljs sökriktningen

$$\mathbf{s}_k = -\nabla f(\mathbf{p}_k).$$

Precis som med kvasi-Newton önskar man

$$\min\ f(\mathbf{p}_k + \alpha \mathbf{s}_k),$$

där α är steglängden.

Ett sätt att beräkna steglängd vid användadet av brantaste nedstigningsmetoden är approximativ exakt linjesökning [17]. Denna metod bygger på att starta med en vald stegländ α och sedan ökar dess värde successivt tills kostnaden börjar öka. I detta arbete får steglängden öka med ett konstant värde $\Delta\alpha$. Om första värdet på stegländen definieras som $\alpha = 0$ så kommer den slutliga steglängden vara $\alpha = n\Delta\alpha$, där n är antalet steg med längden $\Delta\alpha$ som tagits för att hitta en parametervektor som resulterar i en ökad kostnad. Detta innebär att desto mindre värde på $\Delta\alpha$ desto större blir noggrannheten på bekostnad av ökat antal steg n.

Brantaste nedstigningsmetoden kan även användas för att göra första iterationen i kvasi-Newton-algoritmen. Det är fördelaktigt framförallt om man startar med en dålig parametervektor, eftersom det kan resultera i att Wolfe villkoren (8) och (9) uppfylls trots att kostanden är mycket högre än det minsta värdet längs gradientriktningen. Denna åtgärd kan minska antalet iterationer man behöver göra med kvasi-Newtonmetoden för att hitta ett optimum.

2.3.6 Framåtriktad automatisk differentiering

Vid användandet av både brantaste nedstigningsmetoden och kvasi-Newtonmetoden behöver gradienter beräknas. En funktions gradient kan beräknas på ett flertal olika sätt. Den kan exempelvis beräknas numeriskt med finita differenser. Denna metod kan dock ge upphov till signifikanta trunkering- och avrundningsfel [18]. Gradienten kan även beräknas symboliskt med automatisk derivering av matematiska uttryck med hjälp av deriveringsregler. Dock kan även denna metod vara problematisk då den kräver att ingående ekvationer är i sluten form. I vårt fall är detta inte fallet eftersom kostnadsfunktionen är en funktion av ett ODE-system vi inte kan lösa analytiskt. På grund av detta används framåtriktad automatisk differentiering (eng. *Forward Automatic Differentiation*), som förkortas FAD, för beräkning av gradienten på kostnadsfunktionen [18].

Vid FAD beräknas gradienten $\frac{\partial y}{\partial \mathbf{p}}$ av en funktion y. Nedan visas hur detta går till med avsende på en parameter. Derivatan beräknas genom att bygga upp y stegvis och spara dessa steg som intermediärvariabler v. Därefter kan dessa intermediärvariabler deriveras och kombineras för att slutligen få derivatan av funktionen med avsende på parameter p_i. För att få $\frac{\partial y}{\partial p_i}$ sätter man $\dot{p_i} = 1$ och $\dot{p_j} = 0\ \forall\ j \neq i$.

Denna procedur kan exempelvis göras genom att beräkna derivatan med avseende på parameter p_1 för ekvationen nedan

$$y = \sin(p_1) + p_2.$$

Vi börjar med att definiera två intermediärvariabler som p_1 och p_2 enligt

$$v_1 = p_1,\ \dot{v_1} = 1,$$
$$v_2 = p_2,\ \dot{v_2} = 0.$$

Vi noterar att funktionen y är uppbyggd av två termer, nämligen $\sin(p_1)$ och p_2. Vi har redan definierat en intermediärvariabel som p_2 och behöver därför bara definiera $\sin(p_1)$ med hjälp av v_1. Sedan kan vi bygga upp y genom att addera dessa två enligt nedan.

$$v_3 = \sin(v_1),\ \dot{v_3} = cos(v_1),$$
$$v_4 = v_3 + v_2,\ \dot{v_4} = \dot{v_3} + \dot{v_2},$$
$$y = v_4,\ \dot{y} = \dot{v_4}.$$

Här ser vi hur man kan få derivatan av y med avseende på p_1 genom att beräkna $\dot{v}_4$ som i sin tur kan beräknas med $\dot{v}_3$ och $\dot{v}_2$ och så vidare. Om exempelvis $p_1 = \pi$ så blir $\dot{y} = cos(\pi) = -1$.

I FAD används dualtal (eng. *dual numbers*) då man expanderar intermediärvariablerna v till formen $v + \dot{v}\varepsilon$, där $v, \dot{v} \in \mathbb{R}$, $\varepsilon^2 = 0$ och $\varepsilon \neq 0$ [18]. Detta gör så att man kan bevara både informationen om v och $\dot{v}$ genom hela processen. Exempelvis gör denna omskrivning så att produktregeln uppfylls

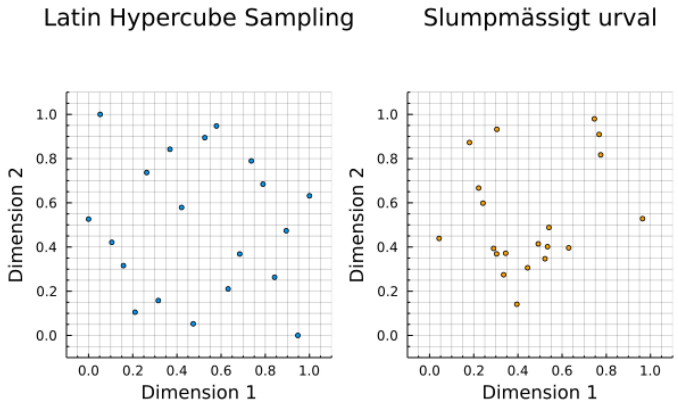
$$(v + \dot{v}\varepsilon)(\nu + \dot{\nu}\varepsilon) = v\nu + (v\dot{\nu} + \dot{v}\nu)\varepsilon,$$

vilket kan ses i högerledet innanför parentesen framför ε.

Allt detta medför att man kan få ut derivatan med avsende på ett v genom att behandla alla icke dualtal v som $v + 0\varepsilon$ medans det v man vill derivera med avsende på definieras som $v + 1\varepsilon$.

2.3.7 Latin Hypercube Sampling

Problemet med metoder som brantaste nedstigningsmetoden och kvasi-Newtonmetoden är att dessa endast kan ta hänsyn till den lokala gradienten och kurvaturen. Detta innebär att majoriteten av parameteruppskattningarna kommer endast hitta lokala optimum. För att kringgå detta kan man generera ett flertal startpunkter till parameteruppskattningen så att man tar hänsyn till ett större område. Ett exempel på en sådan metod är *Latin Hypercube Sampling* (LHS) [19]. Vid LHS skapas ett rutnät där det endast får finnas en punkt längs varje kolumn och rad. Varje punkt placeras sedan på ett slumpvist sätt inom den kvadrat den tilldelats. På så vis undviker man klusterbildning vilket är önskvärt vid generering av startpunkter till parameteruppskattningen.



Figur 2.2: 20 punkter i planet genererade med LHS respektive slumpmässigt urval.

Det syns tydligt i figur 2.2 att slumpmässigt urval har en starkare tendens att skapa kluster än LHS.

2.3.8 Känslighet och identifierbarhet

Vid en parameteruppskattning uppstår frågan om hur säker man kan vara på de funna parametrarna. Därför undersöker vi modellens strukturella och numeriska identifierbarhet. Strukturell identifierbarhet avser bara modellen och är oberoende av parametrarnas värde. Numerisk identifierbarhet bestäms i en viss punkt (det vill säga för en viss parametervektor) och beror alltså på både modellen och lösningen. För att undersöka identifierbarheten krävs känslighetsmatrisen

$$\frac{\partial \mathbf{y}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial y_1(t_1)}{\partial p_1} & \frac{\partial y_1(t_1)}{\partial p_2} & \cdots & \frac{\partial y_1(t_1)}{\partial p_e} \\ \frac{\partial y_1(t_2)}{\partial p_1} & \frac{\partial y_1(t_2)}{\partial p_2} & \cdots & \frac{\partial y_1(t_2)}{\partial p_e} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial y_1(t_N)}{\partial p_1} & \frac{\partial y_1(t_N)}{\partial p_2} & \cdots & \frac{\partial y_1(t_N)}{\partial p_e} \\ \frac{\partial y_2(t_1)}{\partial p_1} & \frac{\partial y_2(t_1)}{\partial p_2} & \cdots & \frac{\partial y_2(t_1)}{\partial p_e} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial y_m(t_N)}{\partial p_1} & \frac{\partial y_m(t_N)}{\partial p_2} & \cdots & \frac{\partial y_m(t_N)}{\partial p_e} \end{bmatrix}, \tag{13}$$

där y_i är funktionsvärden för $i \in [1, m]$, t_j är tiden för $j \in [1, N]$ och p_l är parametern för $l \in [1, e]$ där $m, N, e \in Z^+$ [20].

Rangen hos en matris är antalet linjärt oberoende kolonner eller rader. En matris sägs ha full rang om alla dess kolonner eller rader, beroende på dimension, är linjärt oberoende. Om $\frac{\partial \mathbf{y}}{\partial \mathbf{p}}$ har full rang är parametrarna $\mathbf{p}$ strukturellt identifierbara.

Konditionstalet för en matris A är ett mått på hur fel i indata $\mathbf{b}$ fortplantas till lösningen $\mathbf{x}$ genom ekvationssystemet $A\mathbf{x} = \mathbf{b}$. Är matrisen för vald parametervektor välkonditionerad, det vill säga har litet konditionstal, så är parametrarna numeriskt identifierbara [20].

Varje kolonn i känslighetsmatrisen är den absoluta känslighetsvektorn motsvarande en parameter. För två parametrar kan vi bedöma den relativa identifierbarheten genom beroendet mellan deras respektive känslighetsvektorer. Är de oberoende så är är de numeriskt identifierbara och om de är beroende så är de inte identifierbara. För vinkeln θ_{ij} mellan två vektorer $\frac{\partial \mathbf{y}}{\partial p_i}$ och $\frac{\partial \mathbf{y}}{\partial p_j}$ gäller alltså att $\cos(\theta_{ij})$ nära 1 för icke numeriskt identifierbara parametrar och nära 0 för de som är numeriskt identifierbara relativt varandra [20].

Intimt kopplat till identifierbarhetsanalys är känslighetsanalys. Ett robust (okänsligt) system har litet beroende på små ändringar i indata (här parametrarna) och är därför svårare att göra en korrekt parameteruppskattning för. Liksom för identifierbarhetsanalysen studerar vi här parametrarnas känsligheter $\frac{\partial \mathbf{y}}{\partial \mathbf{p}}$. Normen av en parameters känslighetsvektor ger ett mått på hur känslig modellen är för förändringar i den parametern [20].

3 Metod

För att kunna lösa alla ODE:er och genomföra optimeringen, identifierbarhetsanalysen samt känslighetsanalysen utifrån den teori som presenterats i ovanstående avsnitt krävs programmering. För detta arbete användes Julia v1.7.2 [21].

Under detta projekt användes ett flertal paket i Julia. ODE-systemet i appendix A implementerades med hjälp av paketet *ModelingToolkit* v8.9 eftersom det genomför symboliska beräkningar på ODE-modellen för att representera modellen på ett optimalt sätt för numeriska lösningsmetoder. *DifferentialEquations* v7.1 innehåller ett brett sortiment av högpresterande ODE-lösare.

Eftersom denna ODE-modell innehåller differentialekvationer som modellerar både snabba och långsamma kemiska reaktioner var det förväntat att många av dessa ODE:er skulle vara styva [22]. Detta bekräftades genom att lösa ODE-systemet under identiska förhållanden med både en implicit (för styva ekvationer) och en explicit lösare. I arbetet användes därför en implicit Rosenbrock-lösare, eftersom den är effektiv på att lösa ODE-system som innehåller färre än 50 ODE:er [23].

Paketet *DiffEqSensitivity* v6.72 användes eftersom den har en funktion för beräkning av känslighetsmatrisen. Paketet *ForwardDiff* v0.10.27 innehåller en funktion för beräkning av gradienten med FAD. Resterande paket kan ses i koden som finns i appendix E och på GitHub som nås via länk här. Webbadressen till GitHub finns även i appendix E.1.

Den modellering som genomfördes i projektet använde en modell utvecklad av Jalihal m.fl. [1] som återfinns i appendix A. Modellen implementerades först med den parametervektor som rapporterades av Jalihal m.fl. och som redovisas i appendix B.1. Modellens struktur utgår från den teori över signalnätverket som beskrevs i avsnitt 2.1.2 samt teorin över ODE-modellering i avsnitt 2.2. Genom att använda modellen reproducerades resultat från Jalihal m.fl. i form av ett urval av de figurer som presenterades i originalartikeln [1]. Detta gjordes i syfte att verifiera att modellen implementerades korrekt. Resultaten bestod av tidsserieexperiment och störningsexperiment. För tidsserieexperimenten simulerades modellen för näringsskiften över tid. I störningsexperimenten beräknades däremot endast differensen mellan stationärvärden före och efter ett skifte. Denna differens jämfördes sedan mellan celler av vildtyp och störda celler. Störningar innebär främst mutationer av cellen men kan även vara en behandling med en kemisk förening.

Den experimentella data som återfinns i figurerna användes sedan under parameteruppskattningen i ett försök att hitta paramtervektorer som beskriver data bättre än den från Jalihal m.fl. [1]. När nya parametervektorer erhållits användes dessa för att analysera modellens giltighet genom att jämföra simulerade resultat i olika situationer med kvalitativ data över hur systemet bör bete sig.

Modellen simulerar koncentrationer hos ämnen på en relativ skala från 0 till 1. För näringsämnen innebär 0 brist på näringsämnet medan 1 innebär överflöd. För signalämnen innebär 0 minimal aktivitet och 1 maximal aktivitet. På grund av denna relativa skala behövde experimentell data normaliseras för att kunna jämföras med modellresultat. Detta innebär att det maximala uppmätta värdet hos ett signalämne representerar 1 i den relativa skalan, och den lägsta representerar 0. De resterande experimentella värdena normaliserades enligt ekvation (B.1) som presenteras i appendix B.1.

3.1 Återskapande av figurer för att verifiera modellimplementering

Figurerna av den typ som nämns ovan skapades genom att den implementerade modellen av systemet simulerades till ett stationärtillstånd. Med stationärtillstånd menas här att koncentrationerna hos de ingående signalmolekylerna är konstanta över tid. Detta innebär att systemet i ett stationärtillstånd inte förändras över tid. Efter att stationärvärden har erhållits användes de som initialvärden hos variablerna när systemet simulerades med förändrade parametervärden. Förändringarna var i detta fall skiften av näringskoncentrationer som motsvarar svältning eller tillsättning av källor till glukos eller kväve.

En typ av förändring av modellen är då mutationer av cellen ska simuleras. I detta sammanhang motsvarar förändringen att en signalmolekyl tas bort, vilket i modellen innebär att signalämnet inaktiveras. Detta gjordes oftast genom att sätta värdet för total mängd av ämnet till noll, det vill säga $[\mathrm{P}_T] = 0$, men för vissa mutationer krävdes även andra typer av parameterförändringar. Simuleringar av mutationer gjordes för att jämföra resultaten från olika näringsskiften med experimentell data över hur andra signalmolekyler beter sig i mutanten. Detta kan ge information om vilka roller den raderade signalmolekylen spelar i signalnätverket.

För att återskapa figurerna från Jalihal m.fl. [1] simulerades systemet först under förhållandena redovisade i tabell B.1 och B.4, där resultat från simuleringarna producerades för de olika signalmolekylerna. Dessa resultat presenteras i figur B.1 tillsammans med tillhörande experimentell data från tabell B.2 respektive tabell B.5.

3.2 Implementering av parameteruppskattningsalgoritm

Den data som återfinns i de återskapade figurerna användes sedan som utgångspunkt för parameteruppskattningen, det vill säga motsvarande $\mathbf{y}$ i kostnadsfunktionen (3). Startgissningar av parametervektorer genererades med LHS. Innan evaluering av kostnadsfunktionen i varje iteration simulerades modellen först till ett stationärtillstånd, varpå ett skifte av näringskoncentrationer gjordes. Vid evaluering av kostnadsfunktionen användes de prediktioner $\hat{\mathbf{y}}$ som beräknats under de nya förhållandena.

Innan en parameteruppskattningsalgortim implementerades för ODE-systemet i appendix A gjordes en optimeringsalgoritm för ett enklare exempel. Detta optimeringsexempel kan ses i appendix C.1. Efter att en fungerande optimeringskod utvecklats för optimeringsexemplet anpassades denna till modellen.

3.3 Applicering av parameteruppskattningsalgoritmen på modellen

För att optimeringskoden från exemplet i avsnitt C.1 skulle fungera ihop med modellen behövde ett flertal ändringar göras. I parameteruppskattningen är dimensionen betydligt större än 2, och diverse andra anpassningar krävdes för att få resultat över huvud taget.

ODE-systemet i appendix A innehåller 133 parametrar, vilka presenteras tillsammans med värdena givna av Jalihal m.fl. i tabell B.3. Av dessa skattades 81 stycken. De som inte behandlades i parameteruppskattningen är total mängd protein ($[P_T]$), vars värde sattes till 1.0, och σ-parametrarna, som sattes till de värden erhållna från Jalihal m.fl. [1]. Vi använder de engelska namnen (som exempelvis återfinns i appendix B.1) på parametrar och näringsämnen för att undvika tvetydigheter.

Vid simulering till stationärtillstånd för Carbon = 1, ATP = 1 och $\text{Glutamine}_{\text{ext}} = 1$ uppstod oscillationer i ett flertal av ODE:erna, främst de som tillhör cAMP-PKA-signalvägen. Av denna anledning implementerades ett kriterium där värdet på alla element i gradienten vid stationärtillståndet måste vara mindre än 0,03. Cyr1, Ras, cAMP, Sak och PKA gav upphov till gradientvärden större än 0,03 för parametervärdena från Jalihal m.fl. och således läggs dessa fem till som undantag och påverkas därmed inte av kriteriet.

Val av steglängd för brantaste nedstigningsmetoden modifierades för att minska tiden som gick åt för att hitta optimum längs gradienten. Ändringen var att vart tionde steg multiplicera steglängden med tio. Detta gjorde att α växte snabbare ju fler steg som tagits, vilket främst var användbart då optimum låg långt bort.

Först genererades parametervektorer med LHS på logaritmisk skala mellan -3 och 3 för varje parameter. Sedan användes LHS i närheten av parametervärdena från Jalihal m.fl. [1] för att undersöka om det fanns parametervärden i närheten som gav en bättre beskrivning av datan. Detta gjordes genom en iterativ process där först LHS gjordes i närheten av parametervärdena från Jalihal m.fl. Sedan användes två olika metoder för att finna parametervektorer med lägre kostnad. Den ena metoden gick ut på att göra LHS kring den bästa parametervektorn som skapats från tidigare LHS och repetera denna procedur tills kostnaden minskade mindre än ett valt värde för varje iteration. Här används alltså inte optimeringsalgoritmen alls. Den andra metoden fungerar på ett liknande sätt, förutom att efter varje LHS så används några av de parametervektorer med lägst kostnad som startgissningar till optimeringsalgoritmen. Den bästa vektorn därifrån användes sedan som utgångspunkt för nästa LHS.

Därefter modifierades tillvägagångsättet ovan så att några av de parametervektorer från LHS som givit lägst kostnad optimerades. Därefter genererades nya parametervektorer med LHS kring den bästa optimerade vektorn. Tillvägagångsättet upprepades tills kostnaden inte längre minskade tillräckligt mycket efter varje iteration.

Inför optimeringen valdes följande värden på de konstanter som måste väljas manuellt. För termineringskriterierna valdes att $\epsilon_1, \epsilon_2, \epsilon_3 = 10^{-6}$. För Wolfe-villkoren (se ekvation (8) och (9)) användes $\mu = 10^{-4}$ och $\nu = 0,95$. Slutligen sattes den initiala steglängden på α i brantaste nedstigningsmetoden till 10^{-9}.

Efter parameteruppskattningen gjordes känslighets- och identifierbarhetsanalys. Med hjälp av paketet *DiffEqSensitivity* i Julia genererades en känslighetsmatris i vardera av de föreslagna parametervektorerna, inklusive den från Jalihal m.fl. [1]. Analys av känslighet och identifierbarhet gjordes utifrån dessa matriser.

3.4 Metod för att analysera modellens giltighet

För att förstå den biologiska innebörden av resultaten från parameteruppskattningen genomfördes ytterligare analys av modellen. Systemet simulerades under olika förhållanden med de framtagna parametervektorerna och prediktionerna jämfördes med kvalitativ data över hur systemet bör bete sig. Denna data insamlades genom en litteraturstudie och redovisas i appendix D.1. De fullständiga resultaten av dessa simuleringar redovisas i tabell D.4.

Då kvalitativ data användes innebär detta att den kvantitativa aspekten av simuleringen inte togs i beaktning. Det som bedömdes var endast ifall beteendet hos den observerade signalmolekylen överensstämde med förväntningar. Två exempel på vad kvalitativ data kan innebära i detta sammanhang är att aktiviteten hos en signalmolekyl kommer öka under en specifik förändring i näringsförhållanden, eller att en specifik mutation ändrar aktiviteten hos en signalmolekyl. I modellresultatet bedöms då endast ifall denna ökning sker (på en signifikant nivå), eller om en implementation av mutationen i modellen ger den förväntade förändringen på aktiviteten. Nivåskillnaderna på dessa förändringar tas inte beaktning mer än att de bedöms vara signifikanta.

De resultat som inte överensstämde med kvalitativ data samt de simuleringar där de *kvantitativa* skillnaderna mellan parametervektorernas resultat skiljde sig åt märkvärdigt undersöktes närmare, och presenteras individuellt i avsnitt 4.3. Utifrån dessa resultat kunde en diskussion föras om hur väl modellen beskriver signalnätverket och på vilka sätta modellen kan förbättras. Det kunde även presenteras experiment som skulle kunna informera dessa förbättringar och undersöka resultaten närmare.

4 Resultat

Resultat från genomförandets olika delar presenteras nedan i olika avsnitt. I första delen presenteras resultat från återskapande av figurer från Jalihal m.fl. [1]. Tillvägagångssättet för att återskapa figurerna finns beskrivet i avsnitt 3.1. Data tillhörande dessa figurer används sedan för parameteruppskattningen, beskriven i avsnitt 3.2 och avsnitt 3.3. Resultat från parameteruppskattningen presenteras därför i nästkommande resultatavsnitt. I detta avsnitt sammanställs också resultatet från identifierbarhets- och känslighetsanalysen. Slutligen presenteras resultat från proceduren beskriven i avsnitt 3.4 för att analysera modellens giltighet i det sista resultatavsnittet.

4.1 Återskapade figurer för verifiering av modellens implementering

Enligt tillvägagångssättet beskrivet i avsnitt 3.1, simulerades ODE-modellen i appendix A med de parametervärden rapporterade av Jalihal m.fl. [1]. Detta genomfördes under olika näringsförhållanden för att verifiera att ODE-modellen implementerats korrekt. Parametervärdena, näringsförhållanden före och efter skifte samt tillhörande experimentell data redovisas i appendix B.1. De återskapade figurerna presenteras i appendix B.2. Resultat från alla förutom två simuleringar i störningsexperimenten överensstämde med de resultat presenterade av Jalihal m.fl. [1]. Dessa simuleringar observerade Sch9 under kvävetillsättning för mutanterna lst4Δlst7Δ och gtr1Δgtr2Δ, och kan ses i figur B.1.

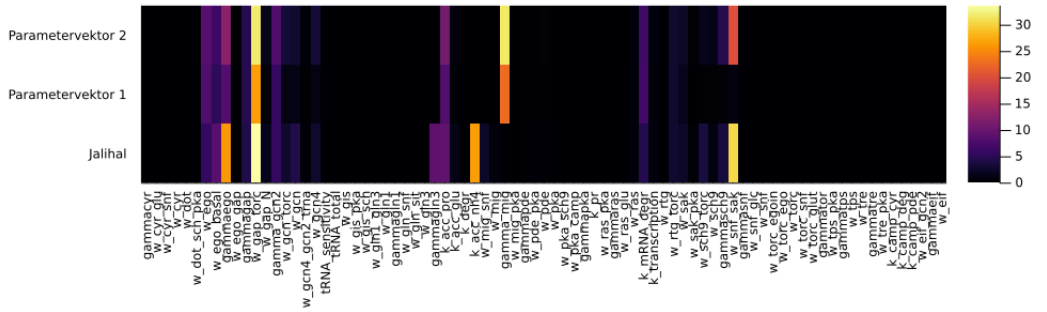
4.2 Resultat av parameteruppskattning

Efter att figurerna återskapades användes tillhörande experimentella data från dessa till att genomföra parameteruppskattningen. LHS med 10 000 parametervektorer på logaritmisk skala mellan -3 och 3 gav 569 parametervektorer med beräkningsbar kostnad och dessa optimerades. Dock hittades inga parametervärden som gav en bra beskrivning av datan.

Genom den iterativa process där både LHS och optimering görs i närheten av parametervärdena från Jalihal m.fl. hittades två parametervektorer som hade ungefär 66% lägre kostnad jämfört med parametervektorn från Jalihal m.fl. [1]. Dessa två parametervektorer redovisas i appendix C.3. En visualisering av resultatet presenteras även i appendix C.2.

Efter att parametervektorer hittats analyserades deras identifierbarhet. Både den strukturella och den numeriska identifierbarheten är låg för systemet. Parametrarna är strukturellt identifierbara om känslighetsmatrisen har full rang, i detta fall 81 (antalet parametrar), men rangen är här 22. Den inbyggda funktionen i Julia för att beräkna konditionstal returnerade oändligheten (∞) då den tillämpades på känslighetsmatriserna för såväl parametervektorn från Jalihal m.fl. som de två parametervektorer vi hittat.

I figur 4.1 presenteras en värmekarta över modellens känsligheter som visar att de flesta parametrarna går att ändra utan större påverkan på kostnaden. Låga känsligheter (mörka i bilden) medför lägre identifierbarhet. Vi kan exempelvis se att modellen är mycket mer känslig för ändringar i $\omega_{gap-torc}$ än i $\omega_{ego-gap}$.



Figur 4.1: Värmekarta över modellens känslighet med avseende på de skattade parametrarna.

I appendix C.4 redovisas värmekartor över parametrarnas parvisa beroenden av varandra i de tre parametervektorer som analyserats. Dessa presenteras i form av cosinus av vinkeln mellan parametrarnas respektive känslighetsvektorer. Vi ser att beroendena är höga mellan de allra flesta paren av parametrar.

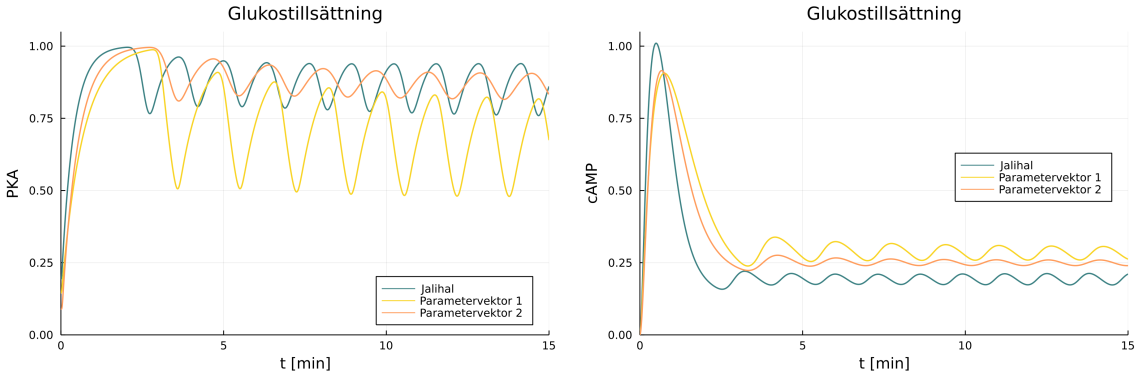
4.3 Resultat från analys av modellens giltighet

ODE-modellens giltighet undersöktes enligt utförandet beskrivet i avsnitt 3.4. 16 simuleringar genomfördes utifrån kvalitativ data över systemet och presenteras i appendix D.1. Utifrån dessa simuleringar kunde slutsatser dras om hur väl modellen beskriver kvalitativ data under olika förhållanden. I denna analys simulerades modellen med både parametervektorn rapporterad av Jalihal m.fl. [1] och de erhållna parametervektorerna som redovisas i tabell C.1 respektive C.2. Resultat från simuleringarna presenteras i sin fullständiga form i tabell D.4 och visualiseras i figur D.1.

Resultaten redovisade i tabellen visar att de flesta simuleringarna överensstämmer med den kvalitativa datan. Generellt sett är resultaten lika för de olika parametervektorerna, även om de kan skilja sig åt kvantitativt. Ett exempel på detta är när Gln3 simuleras under kvävesvältning, vilket redovisas nedan. Av de 16 simuleringar som genomfördes var det 4 som inte stämde överens med kvalitativ data för åtminstone någon av parametervektorerna. Ett urval av simuleringsresultaten från analysen av modellens giltighet presenteras nedan. Dessa resultat syftar till att möjliggöra en djupare diskussion om modellens giltighet.

4.3.1 Oscillationer i cAMP-PKA-signalvägen

Under analys av modellens giltighet upptäcktes att flera ämnen i cAMP-PKA-signalvägen övergår i ett oscillerande tillstånd under glukostillsättning efter cirka 3-4 minuter. I figur 4.2 nedan presenteras två centrala signalmolekyler i signalvägen i varsin panel. Resultaten visar att alla tre parametervektorer följer en liknande dynamik fram till 3-4 minuter. Därefter ger alla upphov till oscillationer för både cAMP och PKA, dock med olika amplitud. Det observeras även att de oscillationer som uppstår verkar vara konstanta över tid.



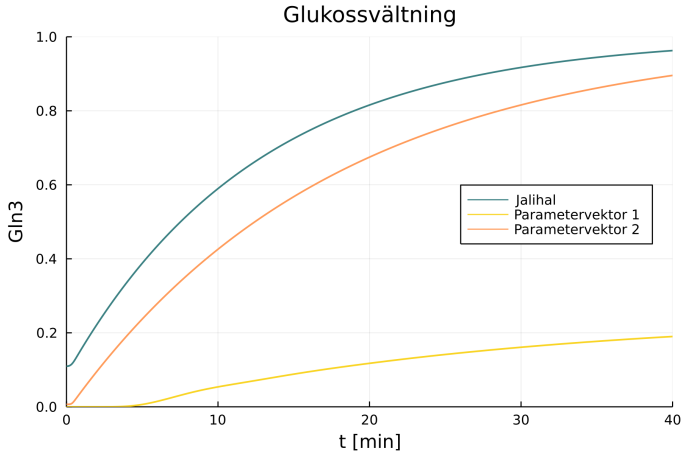
(a) PKA **(b)** cAMP

Figur 4.2: Oscillationer i cAMP-PKA-signalvägen vid glukostillsättning.

4.3.2 Kvantitativa skillnader mellan parametervektorerna för Gln3

Vidare i analysen av modellens giltighet betraktades Gln3 under glukossvältning. Resultat av denna simulering var att parametervektorerna fick samma kvalitativa resultat. I detta fall är dock den *kvantitativa* skillnaden mellan parametervektorernas resultat betydligt större, speciellt mellan parametervektor 1 och de två övriga. Resultatet för denna simulering redovisas nedan i figur 4.3.

De tre resultaten är tämligen likadana ur en kvalitativ synvinkel. Aktiviteten hos Gln3 ökar under glukossvältning, vilket överensstämmer med data [24]. Däremot är skillnaden i storleken hos ökningen stor. För parametervektor 2 och parametervektor från Jalihal m.fl. planar simuleringen ut mot ett stationärvärde runt 0,9, medan motsvarande värde för parametervektor 1 blir 0,2.

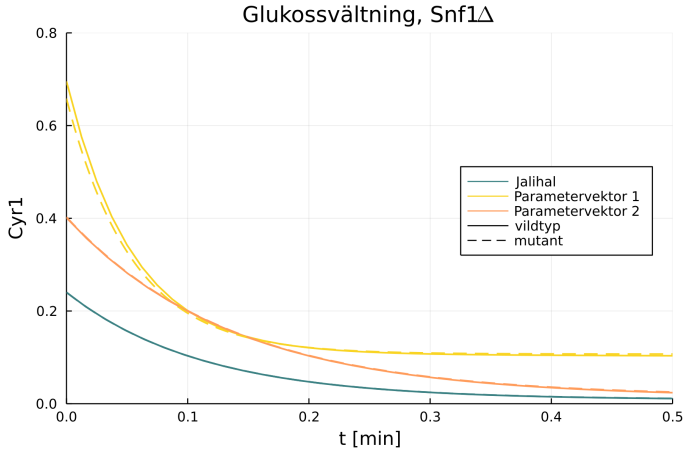


Figur 4.3: Stora kvantitativa skillnader i ökningens storlek mellan parametervektorers resultat för Gln3 under glukossvältning.

4.3.3 Ingen observerbar skillnad mellan mutation och vildtyp för Cyr1

En simulering där resultatet inte överensstämmer med kvalitativ data är då Cyr1 simuleras under glukossvältning. Denna simulering undersöker modellresultatet för en cell av vildtyp och en muterad cell där signalmolekylen Snf1 har inaktiverats (kallad Snf1Δ). Kvalitativ data visar att aktiviteten hos Cyr1 bör minska jämfört med vildtyp under glukossvältning [25]. Resultat från simuleringen visar dock ingen märkbar skillnad mellan resultat från cell av vildtyp och mutant, vilket kan ses i figur 4.4. Detta gäller för alla parametervektorer.

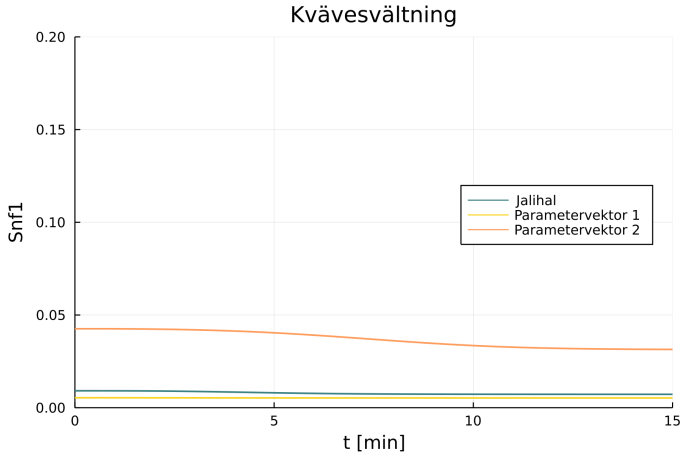
Figur 4.4 illustrerar resultat från fallet som beskrivs ovan genom att simulering från både vildtyp och Snf1Δ ritas i samma figur. Det observeras att skillnaden mellan vildtyp och Snf1Δ är minimala eller obefintliga.



Figur 4.4: Ingen observerad skillnad i minskning av Cyr1-aktivitet mellan mutant och vildtyp under glukossvältning.

4.3.4 Ingen observerbar effekt på Snf1 under kvävesvältning

Ett annat fall där simuleringsresultatet inte överensstämmer med kvalitativ data är då Snf1 simuleras under kvävesvältning. Aktiviteten hos Snf1 borde öka under dessa förhållanden [26, 27]. Resultat från simuleringen, som redovisas i figur 4.5 nedan, visar dock en minimal eller obefintlig effekt på Snf1. Detta gäller främst för parametervektor 1 och parametervektorn från Jalihal m.fl., vilka båda ligger på en konstant låg nivå. Parametervektor 2 ger däremot en kurva med ett något högre stationärvärde före skiftet och en minimal minskning av aktiviteten efter skiftet.



Figur 4.5: Ingen observerbar effekt på Snf1 under kvävesvältning.

5 Diskussion

Innan modellens giltighet analyserades implementerades ODE-modellen från Jalihal m.fl. och resultat återskapades för att verifiera implementeringen [1]. Resultaten visade endast två simuleringar av störningsexperiment som inte överensstämde med resultaten presenterade av Jalihal m.fl. [1]. Det kunde inte fastställas vad orsaken är att resultaten skiljer sig. Alla andra resultat, inklusive resten av störningsexperimenten, påvisar en korrekt implementering av modellen. Eftersom de felaktiga resultaten gäller just mutanterna (som är specifika för den simuleringen) gjordes bedömningen att dessa resultat inte implicerar att andra simuleringsresultat är fel. Därmed anses modellen hädanefter ge rättvisande resultat.

Generellt sett överensstämmer resultaten av simuleringar som presenterats väl med kvalitativ data, som återfinns i tabell D.4. Detta innebär att modellen producerar rimliga resultat i flera olika sammanhang och för flera olika signalmolekyler. Därför kan modellen anses vara användbar.

I tabell D.4 redovisas även ett flertal simuleringar vars resultat inte överensstämmer med kvalitativ data. Enligt den iterativa processen som tillhör systembiologins arbetssätt (tidigare beskriven i den inledande texten till avsnitt 2) är det viktigt att analysera dessa resultat närmare. Utifrån modellen, resultaten och dess relation till data kan man avgöra vilka delar av modellen som behöver förbättras samt hur man kan gå till väga för att göra dessa förbättringar. Därav följer nedan en diskussion kring några av de simuleringar som inte överensstämde med kvalitativ data. I denna diskussion presenteras även förslag på experiment som kan genomföras för att eventuellt få tillgång till mer data och information om signalnätverket. Detta kan sedan användas för att förbättra modellen.

Vid simulering av systemet under glukostillsättning observerades oscillationer hos alla centrala ämnen i cAMP-PKA-signalvägen. Resultaten för cAMP och PKA presenteras i figur 4.2. Oscillationerna hade olika amplitud för de olika parametervektorerna, men alla var konstanta över tid. Den troliga anledningen till att oscillationerna uppstår är kopplat till signalvägarnas struktur, som presenteras i figur 2.1 och beskrivs översiktligt i avsnitt 2.1.2. I figuren illustreras hur de olika signalmolekylerna påverkar varandras aktivitet. För cAMP-PKA-signalvägen syns tydliga återkopplingsslingor (eng. *feedback loops*) där flera signalmolekylers aktivitet påverkar varandra och slutligen indirekt sig själva i två olika slingor. Den ena slingan startar vid PKA som hämmar Ras. Ras aktiverar Cyr1 som i sin tur aktiverar cAMP, som slutligen aktiverar PKA. Den andra slingan startar vid PKA som aktiverar PDE, vilken hämmar cAMP som aktiverar PKA.

Det finns tecken på att oscillering faktiskt förekommer för cAMP under glukostillsättning, men att dessa är avtagande över tid [28]. Detta observerades inte i resultatet från simuleringen där oscillationerna istället var konstanta över tid. Eftersom detta gäller för tre parametervektorer som beskriver tillgänglig kvantitativ data väl kan det vara möjligt att modellen har en otillräcklig beskrivning av denna del av signalnätverket. För att ta reda på ifall detta är fallet bör noggranna experiment göras på cAMP under en längre tidsperiod för att säkerställa oscillationernas existens och karaktär. Det kan också vara användbart att utföra experiment för att ta reda på om även PKA oscillerar. Sammantagna resultat från olika signalmolekyler i cAMP-PKA-signalvägen kan förbättra bilden av hur dess komponenter hör ihop och därefter kan modellen förbättras utifrån denna kunskap.

Då glukossvältning simulerades syntes en ökning av Gln3-aktivitet för alla parametervektorer, vilket visas i figur 4.3. Detta överensstämmer med kvalitativ data [24]. Däremot var det stor *kvantitativ* skillnad i ökningen mellan de tre olika parametervektorerna. Det var framförallt parametervektor 1 som gav en betydligt mindre ökning än de två övriga parametervektorerna. Eftersom skillnaden mellan simuleringsresultaten var såpass stor är det rimligt att experimentell data på ökningens storlek skulle kunna användas för att ge inblick i vilken av de tre parametervektorerna som beskriver systemet bäst. Resultatet skulle även kunna tyda på att modellens beskrivning av Gln3 är felaktig eller inte komplett. Därför behövs mer experimentell data på Gln3 för att svara på dessa frågor.

När Cyr1 analyserades under glukossvältning genomfördes två simuleringar, en för vildtyp och en för mutanten Snf1Δ. Resultaten från dessa simuleringar redovisas tillsammans i figur 4.4. Från re-

sultatet observerades ingen märkbar skillnad mellan mutation och vildtyp, vilket enligt kvalitativ data bör varit fallet [25]. Att ingen skillnad observerats mellan mutant och vildtyp i simuleringen är inte rimligt eftersom signifikanta skillnader i experiment innebär att vi bör se stora skillnader i simuleringsresultat, eftersom brus i experimentell data generellt sett försvårar sådant arbete [29]. Anledning till skillnaden mellan kvalitativ data och modellresultat skulle kunna bero på modellens struktur. Exempelvis kan det saknas komponenter som påverkar Cyr1 så att den inte inaktiveras helt för vildtyp, utan istället stabiliseras på ett lägre värde efter en tid. För mutationen skulle det i så fall kunna innebära en större minskning jämfört med vildtyp. För att hitta eventuella komponenter som skulle kunna utöka modellen och beskriva denna dynamik krävs vidare litteraturstudie och experiment på vad som påverkar Cyr1 under glukossvältning.

Ytterligare värt att nämna är att det var svårt att genomföra simuleringar för att testa modellens giltighet i många av de fall där det finns kvalitativ data över hur mutationer påverkar systemet. Det är just därför inte många av dessa simuleringar har genomförts. Anledningen till denna svårighet är att modellen är relativt avskalad i jämförelse med det verkliga biologiska systemet. Därför saknas det i modellen flera komponenter som undersökts i litteratur. Det är möjligt att försöka finna motsvarigheter i modellen men resultaten av sådana simuleringar är osäkrare och svårare att tolka. Därför skulle modellen gynnas av att innehålla fler komponenter i signalnätverket och fler relationer mellan dessa komponenter. En sådan utökning möjliggör simulering av fler mutationer, vilket förenklar arbetet att analysera modellen.

Ett sätt att förbättra modellen ytterligare kan därför vara att efter vidare litteraturstudie lägga till flera komponenter för att få en mer välutvecklad modell. Då skulle simuleringar av mutationer kunna vara enklare att genomföra och på så vis underlätta analys och vidareutveckling av modellen.

Resultatet från simulering av Snf1 under kvävesvältning, som redovisas i figur 4.5, visar att ingen ökning av dess aktivitet sker. Detta gäller för alla parametervektorer. Enligt kvalitativ data ska denna ökning ske [26, 27] och om detta resultat observeras i experiment bör ökningen vara relativt tydlig i simuleringen [29]. Därav beror denna skillnad mellan simulering och data troligtvis på att något saknas i modellens struktur.

Då signalvägen TORC1 aktiveras under kvävesvältning, vilket beskrevs i avsnitt 2.1.2, skulle en aktivering av Snf1 under kvävesvältning kräva en koppling mellan dessa två signalvägar. Genom att betrakta figur 2.1 observerar man att Snf1 inhiberar TORC1 men att ingen koppling sker i andra riktningen, det vill säga ingen komponent i TORC1-signalvägen förändrar aktiviteten hos en komponent i SNF1-signalvägen. Det finns dock bevis för att Snf1 är involverad i kvävesignalering och påverkas av TORC1 [11]. Därför behövs ytterligare undersökning för att fastställa denna koppling och bestämma dess mekanism. Därefter kan modellen utökas för att inkludera kopplingen. Förslagsvis kan detta göras genom att experiment utförs för att bestämma vilken eller vilka komponent(er) i TORC1 som aktiverar Snf1. Det bör även utföras experiment för att samla in *kvantitativ* data på Snf1 under kvävesvältning, så värden på relevanta parametrar kan uppskattas.

Från parameteruppskattningen erhölls två parametervektorer med relativt låg kostnad. Majoriteten av parametrarna har dock låg känslighet, vilket syns i figur 4.1. Den låga identifierbarheten är inte särskilt förvånande eftersom systemet är gravt överparametriserat. Det är alltså väldigt svårt att bestämma parametrar med någorlunda säkerhet. Detta kan avhjälpas genom att hämta mer data, från flera olika signalämnen och under fler olika situationer, som parameteruppskattningen kan baseras på. Dessutom kan man höja identifierbarheten genom att formulera om modellen så att parametrar som är väldigt beroende av varandra istället representeras av en enda parameter. Allteftersom modellen utvecklas och förbättras kan fler parametrar bli identifierbara.

Parametervektorerna som tagits fram beskriver alla situationer det finns data på till synes lika väl, men med vår kostnadsfunktion finns en risk att vissa fall skulle favoriseras. I kostnadsfunktionen användes oviktad data så att varje enskild observation vägde lika tungt. Det medför att de experiment som innehåller fler observationer (mätpunkter) väger tyngre än de med få observationer. För att få mer rättvisande parametervektorer borde man vikta datan så att varje mätserie väger lika tungt.

En annan förbättring hade varit att beräkna hessianen med FAD istället för att approximera den med BFGS-metoden i ekvation (4). Det är dock beräkningsmässigt dyrt och rekommenderas därför om man har tillgång till en större beräkningsbudget.

Vi noterar att modellen som används har en tendens att ge upphov till en mycket fluktuerande kostnad. För att kunna hitta förhållandevis bra parametervärden utan att utgå ifrån resultatet från Jalihal m.fl. [1] behövs ett stort antal startvärden, så att man minskar risken för terminering vid lokala optimum med hög kostnad. Att optimera ett stort antal parametervektorer kräver en stor beräkningsbudget. Om man har tillgång till detta kan det rekommenderas en parameteruppskattning som använder sig av ett mycket större antal startgissningar än de 569 som använts i detta arbete.

Det finns ett flertal kodrelaterade åtgärder som skulle kunna leda till en bättre parameteruppskattning. Ökad hastighet på koden skulle kunna ge bättre resultat, eftersom parameteruppskattningen då kan göras med fler startgissningar. Att använda en kommersiell optimeringsalgorim som *Ipopt* kan vara både bättre och snabbare jämfört med den egenimplementerade optimeringsalgorimen i detta projekt [30].

En annan åtgärd som kan ge ökad hastighet är att undersöka möjliga beräkningsalternativ för evaluering av gradienten. Under optimeringen användes FAD, en metod som är relativt enkel och snabb att tillämpa och därför kan vara fördelaktig för ODE-system med ett mindre antal parametrar. Eftersom ODE-systemet här (presenteras i appendix A) innefattar fler än 100 parametrar skulle beräkningen av gradienten eventuellt kunnat genomföras snabbare med en metod som har det engelska namnet *adjoint sensitivity analysis* [31]. Vi rekommenderar därför att undersöka denna metod ifall en ny parameteruppskattning skulle göras.

Det går även att göra en granskning av simulering till stationärtillstånd vid oscillationer eftersom dessa situationer egentligen inte har något stationärtillstånd och därför tar lång tid att beräkna. I detta arbete implementerades ett kriterium där alla element i gradienten vid stationärtillstånd måste vara mindre än 0,03 innan beräkningen fortsätter. Detta ökade hastigheten på optimeringen eftersom den går vidare till nästa iteration ifall ODE:erna oscillerar för mycket, istället för att fortsätta försöka beräkna stationärtillstånd som inte finns. Stationärtillstånd med värden på gradientelementen under 0,03 kan också oscillera. Detta kan leda till att lösaren fortsätter att leta efter ett stationärtillstånd tills den nått en gräns på antalet försök och till slut ger snittvärdet på oscillationen som stationärvärde. Denna process kan ta lång tid. En lösning på detta problem hade lett till en snabbare parameteruppskattningskod som hade gjort det möjligt att hinna med fler startgissningar.

6 Slutsatser

Från diskussion i föregående avsnitt kan flera slutsatser dras om modellen framtagen av Jalihal m.fl. [1]. De två parametervektorerna erhållna genom parameteruppskattning beskriver data bättre än parametervektorn rapporterad av Jalihal m.fl. Trots detta identifierades ett flertal modellresultat som inte överensstämmer med data. Utifrån detta kan slutsatsen dras att modellen i vissa sammanhang verkar sakna komponenter, eller interaktioner mellan komponenter, som hade kunnat ge en mer komplett beskrivning av systemets beteende. Dessutom har parametrarna låg identifierbarhet.

Sammantaget innebär detta att modellen skulle behöva utökas. För att genomföra en sådan utökning krävs ytterligare experiment. Data från dessa experiment kan underlätta utökningen både genom att fastställa vad som saknas i modellen och genom att ge bättre förutsättningar för parameteruppskattningen. Utöver experiment kan även en utförligare litteraturstudie genomföras för att samla in data och information som kan användas för att lägga till ytterligare komponenter i modellen. Dessa två vidareutvecklingar hade troligtvis varit nästa steg i detta arbete.

Referenser

1. Jalihal AP, Kraikivski P, Murali T och Tyson JJ. Modeling and analysis of the macronutrient signaling network in budding yeast. Molecular Biology of the Cell 2021; 32:ar20. DOI: https://doi.org/10.1091/mbc.E20-02-0117
2. Mattanovich D, Sauer M och Gasser B. Yeast biotechnology: teaching the old dog new tricks. Microbial cell factories 2014; 13:1–5. DOI: https://doi.org/10.1186/1475-2859-13-34
3. Kaeberlein M. Lessons on longevity from budding yeast. Nature 2010; 464:513–9. DOI: https://doi.org/10.1038/nature08981
4. Coccetti P, Nicastro R och Tripodi F. Conventional and emerging roles of the energy sensor Snf1/AMPK in Saccharomyces cerevisiae. Microbial Cell 2018; 5:482. DOI: https://dx.doi.org/10.15698%2Fmic2018.11.655
5. Broach JR. Nutritional control of growth and development in yeast. Genetics 2012; 192:73–105. DOI: https://doi.org/10.1534/genetics.111.135731
6. Wolkenhauer O, Wellstead P, Cho KH, Rangamani P och Iyengar R. Modelling cellular signalling systems. Essays in Biochemistry 2008; 45:83–94. DOI: https://doi.org/10.1042/BSE0450083
7. Gillespie DT. Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. 2007; 58:35–55. DOI: https://doi.org/10.1146/annurev.physchem.58.032806.104637
8. Klipp E och Liebermeister W. Mathematical modeling of intracellular pathways. BMC Neuroscience. 2006; 7. DOI: https://doi.org/10.1186/1471-2202-7-S1-S10
9. Klipp E, Herwig R, Kowald A, Wierling C och Lehrach H. Systems Biology in Practice. Concepts, Implementation and Application. Wiley-VCH, 2005. Kap. 1
10. Alberts B, Johnson A, Lewis J, Morgan D, Raff M, Roberts K och Walter P. Molecular Biology of the Cell. 6. utg. Garland Science, 2015. Kap. 15:813–4
11. Orlova M, Kanter E, Krakovich D och Kuchin S. Nitrogen availability and TOR regulate the Snf1 protein kinase in Saccharomyces cerevisiae. Eukaryotic cell 2006; 5:1831–7. DOI: https://doi.org/10.1128/EC.00110-06
12. Fayyad-Kazan M, Feller A, Bodo E, Boeckstaens M, Marini AM, Dubois E och Georis I. Yeast nitrogen catabolite repression is sustained by signals distinct from glutamine and glutamate reservoirs. Molecular microbiology 2016; 99:360–79. DOI: https://doi.org/10.1111/mmi.13236
13. Klipp E och Liebermeister W. Mathematical modeling of intracellular signaling pathways. BMC neuroscience 2006; 7:1–16. DOI: https://doi.org/10.1186/1471-2202-7-S1-S10
14. Laomettachit T, Chen KC, Baumann WT och Tyson JJ. A model of yeast cell-cycle regulation based on a standard component modeling strategy for protein regulatory networks. PloS one 2016; 11:e0153738. DOI: https://doi.org/10.1371/journal.pone.0153738
15. Rock P och Vanderzee C. Chemical equilibrium. AccessScience 2020. DOI: https://doi.org/10.1036/1097-8542.127300
16. Andréasson N, Evgrafov A, Patriksson M, Gustavsson E, Nedělková Z, Sou KC och Önnheim M. An Introduction to Continuous Optimization. Foundations and Fundamental Algorithms. Studentlitteratur, 2016. Kap. 11
17. Fridovich-Keil S och Recht B. Approximately Exact Line Search. arXiv preprint arXiv:2011.04721 2020. DOI: https://doi.org/10.48550/ARXIV.2011.04721
18. Baydin AG, Pearlmutter BA, Radul AA och Siskind JM. Automatic differentiation in machine learning: a survey. Journal of Marchine Learning Research 2018; 18:1–43. DOI: https://doi.org/10.48550/ARXIV.1502.05767
19. Raue A, Schilling M, Bachmann J, Matteson A, Schelke M, Kaschek D, Hug S, Kreutz C, Harms BD, Theis FJ m. fl. Lessons learned from quantitative dynamical modeling in systems biology. PloS one 2013; 8:e74335. DOI: https://doi.org/10.1371/annotation/ea0193d8-1f7f-492a-b0b7-d877629fdcee
20. DiStefano J. Dynamic Systems Biology Modeling and Simulation. Elsevier, 2015. Kap. 10-12
21. Bezanson J, Edelman A, Karpinski S och Shah VB. Julia: A fresh approach to numerical computing. SIAM review 2017; 59:65–98. DOI: https://doi.org/10.1137/141000671
22. Städter P, Schälte Y, Schmiester L, Hasenauer J och Stapor PL. Benchmarking of numerical integration methods for ODE models of biological systems. Scientific reports 2021; 11:1–11. DOI: https://doi.org/10.1038/s41598-021-82196-2

23. Rackauckas C och Nie Q. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. Journal of Open Research Software 2017; 5. DOI: http://doi.org/10.5334/jors.151
24. Bertram PG, Choi JH, Carvalho J, Chan TF, Ai W och Zheng XS. Convergence of TOR-nitrogen and Snf1-glucose signaling pathways onto Gln3. Molecular and cellular biology 2002; 22:1246–52. DOI: https://doi.org/10.1128/MCB.22.4.1246-1252.2002
25. Nicastro R, Tripodi F, Gaggini M, Castoldi A, Reghellin V, Nonnis S, Tedeschi G och Coccetti P. Snf1 phosphorylates adenylate cyclase and negatively regulates protein kinase A-dependent transcription in Saccharomyces cerevisiae. Journal of Biological Chemistry 2015; 290:24715–26. DOI: https://doi.org/10.1074/jbc.M115.658005
26. Kuchin S, Vyas VK och Carlson M. Snf1 protein kinase and the repressors Nrg1 and Nrg2 regulate FLO11, haploid invasive growth, and diploid pseudohyphal differentiation. Molecular and cellular biology 2002; 22:3994–4000. DOI: https://doi.org/10.1128/MCB.22.12.3994-4000.2002
27. Orlova M, Ozcetin H, Barrett L och Kuchin S. Roles of the Snf1-activating kinases during nitrogen limitation and pseudohyphal differentiation in Saccharomyces cerevisiae. Eukaryotic cell 2010; 9:208–14. DOI: https://doi.org/10.1128/EC.00216-09
28. Gonzales K, Kayıkçı Ö, Schaeffer DG och Magwene PM. Modeling mutant phenotypes and oscillatory dynamics in the Saccharomyces cerevisiae cAMP-PKA pathway. BMC systems biology 2013; 7:1–16. DOI: https://doi.org/10.1186/1752-0509-7-40
29. Clark D och Pazdernik N. Biotechnology. 2. utg. Academic Cell, 2015
30. Wächter A och Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming 2006; 106:25–57. DOI: https://doi.org/10.1007/s10107-004-0559-y
31. Ma Y, Dixit V, Innes MJ, Guo X och Rackauckas C. A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions. *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 2021 :1–9. DOI: https://doi.org/10.1109/HPEC49654.2021.9622796
32. Wilson WA, Hawley SA och Hardie DG. Glucose repression/derepression in budding yeast: SNF1 protein kinase is activated by phosphorylation under derepressing conditions, and this correlates with a high AMP: ATP ratio. Current Biology 1996; 6:1426–34. DOI: https://doi.org/10.1016/S0960-9822
33. Crauwels M, Donaton MC, Pernambuco MB, Winderickx J, Winde JH de och Thevelein JM. The Sch9 protein kinase in the yeast Saccharomyces cerevisiae controls cAPK activity and is required for nitrogen activation of the fermentable-growth-medium-induced (FGM) pathway. Microbiology 1997; 143:2627–37. DOI: https://doi.org/10.1099/00221287-143-8-2627
34. Stracka D, Jozefczuk S, Rudroff F, Sauer U och Hall MN. Nitrogen source activates TOR (target of rapamycin) complex 1 via glutamine and independently of Gtr/Rag proteins. Journal of Biological Chemistry 2014; 289:25010–20. DOI: https://doi.org/10.1074/jbc.M114.574335
35. Prouteau M, Desfosses A, Sieben C, Bourgoint C, Lydia Mozaffari N, Demurtas D, Mitra AK, Guichard P, Manley S och Loewith R. TORC1 organized in inhibited domains (TOROIDs) regulate TORC1 activity. Nature 2017; 550:265–9. DOI: https://doi.org/10.1038/nature24021
36. Welkenhuysen N, Borgqvist J, Backman M, Bendrioua L, Goksör M, Adiels CB, Cvijovic M och Hohmann S. Single-cell study links metabolism with nutrient signaling and reveals sources of variability. BMC systems biology 2017; 11:1–10. DOI: https://doi.org/10.1186/s12918-017-0435-z
37. Powers T och Walter P. Regulation of ribosome biogenesis by the rapamycin-sensitive TOR-signaling pathway in Saccharomyces cerevisiae. Molecular biology of the cell 1999; 10:987–1000. DOI: https://doi.org/10.1091/mbc.10.4.987
38. Hong SP, Leiper FC, Woods A, Carling D och Carlson M. Activation of yeast Snf1 and mammalian AMP-activated protein kinase by upstream kinases. Proceedings of the National Academy of Sciences 2003; 100:8839–43. DOI: https://doi.org/10.1073/pnas.1533136100
39. Mbonyi K, Van Aelst L, Argüelles J, Jans A och Thevelein J. Glucose-induced hyperaccumulation of cyclic AMP and defective glucose repression in yeast strains with reduced activity of cyclic AMP-dependent protein kinase. Molecular and Cellular Biology 1990; 10:4518–23. DOI: https://doi.org/10.1128/mcb.10.9.4518-4523.1990

40. Ma P, Wera S, Van Dijck P och Thevelein JM. The PDE1-encoded low-affinity phosphodiesterase in the yeast Saccharomyces cerevisiae has a specific function in controlling agonist-induced cAMP signaling. Molecular biology of the cell 1999; 10:91–104. DOI: https://doi.org/10.1091/mbc.10.1.91
41. Mbonyi K och Thevelein JM. The high-affinity glucose uptake system is not required for induction of the RAS-mediated cAMP signal by glucose in cells of the yeast Saccharomyces cerevisiae. Biochimica et Biophysica Acta (BBA)-Molecular Cell Research 1988; 971:223–6. DOI: https://doi.org/10.1016/0167-4889(88)90195-4
42. Péli-Gulli MP, Sardu A, Panchaud N, Raucci S och De Virgilio C. Amino acids stimulate TORC1 through Lst4-Lst7, a GTPase-activating protein complex for the Rag family GTPase Gtr2. Cell reports 2015; 13:1–7. DOI: https://doi.org/10.1016/j.celrep.2015.08.059
43. Rolfes RJ och Hinnebusch AG. Translation of the yeast transcriptional activator GCN4 is stimulated by purine limitation: implications for activation of the protein kinase GCN2. Molecular and cellular biology 1993; 13:5099–111. DOI: https://doi.org/10.1128/mcb.13.8.5099-5111.1993
44. Beck T och Hall MN. The TOR signalling pathway controls nuclear localization of nutrient-regulated transcription factors. Nature 1999; 402:689–92. DOI: https://doi.org/10.1038/45287
45. Garcıa-Salcedo R, Lubitz T, Beltran G, Elbing K, Tian Y, Frey S, Wolkenhauer O, Krantz M, Klipp E och Hohmann S. Glucose de-repression by yeast AMP-activated protein kinase SNF 1 is controlled via at least two independent steps. The FEBS journal 2014; 281:1901–17. DOI: https://doi.org/10.1111/febs.12753
46. Welkenhuysen N, Schnitzer B, Österberg L och Cvijovic M. Robustness of nutrient signaling is maintained by interconnectivity between signal transduction pathways. Frontiers in physiology 2019; 9:1964. DOI: https://doi.org/10.3389/fphys.2018.01964
47. Orzechowski Westholm J, Tronnersjö S, Nordberg N, Olsson I, Komorowski J och Ronne H. Gis1 and Rph1 regulate glycerol and acetate metabolism in glucose depleted yeast cells. PloS one 2012; 7:e31577. DOI: https://doi.org/10.1371/journal.pone.0031577
48. Pedruzzi I, Bürckert N, Egger P och De Virgilio C. Saccharomyces cerevisiae Ras/cAMP pathway controls post-diauxic shift element-dependent transcription through the zinc finger protein Gis1. The EMBO journal 2000; 19:2569–79. DOI: https://doi.org/10.1093/emboj/19.11.2569
49. Wei M, Fabrizio P, Hu J, Ge H, Cheng C, Li L och Longo VD. Life span extension by calorie restriction depends on Rim15 and transcription factors downstream of Ras/PKA, Tor, and Sch9. PLoS genetics 2008; 4:e13. DOI: https://doi.org/10.1371/journal.pgen.0040013
50. Kunkel J, Luo X och Capaldi AP. Integrated TORC1 and PKA signaling control the temporal activation of glucose-induced gene expression in yeast. Nature communications 2019; 10:1–11. DOI: https://doi.org/10.1038/s41467-019-11540-y
51. Rai R, Tate JJ, Nelson DR och Cooper TG. gln3 mutations dissociate responses to nitrogen limitation (nitrogen catabolite repression) and rapamycin inhibition of TorC1. Journal of Biological Chemistry 2013; 288:2789–804. DOI: https://doi.org/10.1074/jbc.M112.421826
52. Beck T och Hall MN. The TOR signalling pathway controls nuclear localization of nutrient-regulated transcription factors. Nature 1999; 402:689–92. DOI: https://doi.org/10.1038/45287
53. Kulkarni A, Buford TD, Rai R och Cooper TG. Differing responses of Gat1 and Gln3 phosphorylation and localization to rapamycin and methionine sulfoximine treatment in Saccharomyces cerevisiae. FEMS yeast research 2006; 6:218–29. DOI: https://doi.org/10.1111/j.1567-1364.2006.00031.x
54. Conrad M, Schothorst J, Kankipati HN, Van Zeebroeck G, Rubio-Texeira M och Thevelein JM. Nutrient sensing and signaling in the yeast Saccharomyces cerevisiae. FEMS microbiology reviews 2014; 38:254–99. DOI: https://doi.org/10.1111/1574-6976.12065
55. Marion RM, Regev A, Segal E, Barash Y, Koller D, Friedman N och O'Shea EK. Sfp1 is a stress-and nutrient-sensitive regulator of ribosomal protein gene expression. Proceedings of the National Academy of Sciences 2004; 101:14315–22. DOI: https://doi.org/10.1073/pnas.0405353101
56. Düvel K, Santhanam A, Garrett S, Schneper L och Broach JR. Multiple roles of Tap42 in mediating rapamycin-induced transcriptional changes in yeast. Molecular cell 2003; 11:1467–78. DOI: https://doi.org/10.1016/S1097-2765(03)00228-4

57. Huber A, French SL, Tekotte H, Yerlikaya S, Stahl M, Perepelkina MP, Tyers M, Rougemont J, Beyer AL och Loewith R. Sch9 regulates ribosome biogenesis via Stb3, Dot6 and Tod6 and the histone deacetylase complex RPD3L. The EMBO journal 2011; 30:3052–64. DOI: https://doi.org/10.1038/emboj.2011.221
58. Garcıa-Salcedo R, Lubitz T, Beltran G, Elbing K, Tian Y, Frey S, Wolkenhauer O, Krantz M, Klipp E och Hohmann S. Glucose de-repression by yeast AMP-activated protein kinase SNF 1 is controlled via at least two independent steps. The FEBS journal 2014; 281:1901–17. DOI: https://doi.org/10.1111/febs.12753
59. Molinet J, Salinas F, Guillamón JM och Martınez C. GTR1 affects nitrogen consumption and TORC1 activity in Saccharomyces cerevisiae under fermentation conditions. Frontiers in genetics 2020; 11:519. DOI: https://doi.org/10.3389/fgene.2020.00519

Appendix

A ODE-modell över signalvägarna hos bagerijäst

Nedan visas alla ODE:er som utgör den dynamiska ODE-modell som använts till att beskriva datan. De olika ekvationerna är utformade på olika sätt utifrån de tre ekvationstyper som presenterats i sektion 2.2. Klass II ur SCM är den vanligaste. ODE:erna nedan är indelade i grupper utifrån det syfte molekylerna har. De tre grupper är näringsavkänning och signaltransduktion, huvudregulatorer och nedströmssvar [1].

Näringsavkänning och transduktion

$$\begin{aligned}
\frac{\mathrm{d}[\mathrm{Glutamin}]}{\mathrm{d}t} &= (k_{acc-glu}[\mathrm{Glutamin}_{ext}] + k_{acc-pro}\mathrm{Prolin} + k_{acc-nh4}\mathrm{NH_4}[\mathrm{Gln1}]\mathrm{Carbon}) - \\
&\quad k_{degr}[\mathrm{Glutamin}] \\
\frac{\mathrm{d}[\mathrm{Cyr1}]}{\mathrm{d}t} &= \gamma_{cyr}([\mathrm{Cyr1}_T]\mathcal{H}(\sigma_{cyr}, \omega_{cyr-glu}\mathrm{Carbon}[\mathrm{Ras}] - \omega_{cyr} - \omega_{cyr-snf}[\mathrm{Snf1}]) - [\mathrm{Cyr1}]) \\
\frac{\mathrm{d}[\mathrm{Ras}]}{\mathrm{d}t} &= \gamma_{ras}([\mathrm{Ras}_T]\mathcal{H}(\sigma_{ras}, -\omega_{ras-pka}[\mathrm{PKA}] + \omega_{ras-glu}\mathrm{Carbon} + \omega_{ras}) - [\mathrm{Ras}]) \\
\frac{\mathrm{d}[\mathrm{EGO}]}{\mathrm{d}t} &= \gamma_{ego}([\mathrm{EGO}_T]\mathcal{H}(\sigma_{ego}, \omega_{ego-gap}[\mathrm{EGOGAP}]([\mathrm{Glutamin}_{ext}] + 0.5\mathrm{NH_4} + \\
&\quad 0.01\mathrm{Prolin}) - \omega_{ego}(1 - [\mathrm{Glutamin}]) - \omega_{ego-basal}) - [\mathrm{EGO}]) \\
\frac{\mathrm{d}[\mathrm{EGOGAP}]}{\mathrm{d}t} &= \gamma_{gap}([\mathrm{EGOGAP}_T]\mathcal{H}(\sigma_{gap}, \omega_{gap-N}(1 - [\mathrm{Glutamin}]) - \omega_{gap-torc}[\mathrm{TORC1}]) - [\mathrm{EGOGAP}]) \\
\frac{\mathrm{d}[\mathrm{cAMP}]}{\mathrm{d}t} &= k_{camp-cyr}[\mathrm{Cyr1}]\mathrm{ATP} - k_{camp-pde}[\mathrm{PDE}][\mathrm{cAMP}] - k_{camp-deg}[\mathrm{cAMP}] \\
\frac{\mathrm{d}[\mathrm{PDE}]}{\mathrm{d}t} &= \gamma_{pde}([\mathrm{PDE}_T]\mathcal{H}(\sigma_{pde}, \omega_{pde-pka}[\mathrm{PKA}] - \omega_{pde}) - [\mathrm{PDE}]) \\
\frac{\mathrm{d}[\mathrm{Sak}]}{\mathrm{d}t} &= [\mathrm{Sak}_T]\mathcal{H}(\sigma_{sak}, \omega_{sak} - \omega_{sak-pka}[\mathrm{PKA}]) - [\mathrm{Sak}]
\end{aligned}$$

Huvudregulatorer

$$\begin{aligned}
\frac{\mathrm{d}[\mathrm{TORC1}]}{\mathrm{d}t} &= \gamma_{tor}([\mathrm{TORC1}_T]\mathcal{H}(\sigma_{tor}, \omega_{torc-glut}[\mathrm{Glutamin}] + \omega_{torc-ego}[\mathrm{EGO}] - \omega_{torc-egoin}(1 - [\mathrm{EGO}]) - \\
&\quad \omega_{torc} - \omega_{torc-snf}[\mathrm{Snf1}]) - [\mathrm{TORC1}]) \\
\frac{\mathrm{d}[\mathrm{Snf1}]}{\mathrm{d}t} &= \gamma_{snf}([\mathrm{Snf1}_T]\mathcal{H}(\sigma_{snf}, -\omega_{snf-glc}\mathrm{Carbon} + \omega_{snf-sak}[\mathrm{Sak}] - \omega_{snf}) - [\mathrm{Snf1}]) \\
\frac{\mathrm{d}[\mathrm{PKA}]}{\mathrm{d}t} &= \gamma_{pka}([\mathrm{PKA}_T]\mathcal{H}(\sigma_{pka}, \omega_{pka-camp}[\mathrm{cAMP}] - \omega_{pka} - \omega_{pka-sch9}[\mathrm{Sch9}]) - [\mathrm{PKA}]) \\
\frac{\mathrm{d}[\mathrm{Sch9}]}{\mathrm{d}t} &= \gamma_{sch9}([\mathrm{Sch9}_T]\mathcal{H}(\sigma_{sch9}, \omega_{sch9-torc}[\mathrm{TORC1}] - \omega_{sch9}) - [\mathrm{Sch9}])
\end{aligned}$$

Nedströmssvar

$$\begin{aligned}
\frac{\mathrm{d}[\mathrm{Gcn2}]}{\mathrm{d}t} &= \gamma_{gcn2}([\mathrm{Gcn2}_T]\mathcal{H}(\sigma_{gcn2}, \omega_{gcn} - \omega_{gcn-torc}[\mathrm{Sch9}]) - [\mathrm{Gcn2}]) \\
\frac{\mathrm{d}[\mathrm{Gcn4}]}{\mathrm{d}t} &= \gamma_{gcn4}([\mathrm{Gcn4}_T]\mathcal{H}(\sigma_{gcn4}, \omega_{gcn4-gcn2-trna}\min([\mathrm{Gcn2}], \mathrm{tRNA}_{sensitivity}([\mathrm{tRNA}_{total}] - \\
&\quad \min([\mathrm{tRNA}_{total}], [\mathrm{Glutamin}]))) - \omega_{gcn4}) - [\mathrm{Gcn4}]) \\
\frac{\mathrm{d}[\mathrm{eIF}]}{\mathrm{d}t} &= \gamma_{eif}([\mathrm{eIF}_T]\mathcal{H}(\sigma_{eif}, \omega_{eif} - \omega_{eif-gcn2}[\mathrm{Gcn2}]) - [\mathrm{eIF}]) \\
\frac{\mathrm{d}[\mathrm{Gln3}]}{\mathrm{d}t} &= \gamma_{gln3}([\mathrm{Gln3}_T]\mathcal{H}(\sigma_{gln}, -\omega_{gln3} + \omega_{gln-snf}[\mathrm{Snf1}] + \omega_{gln-sit}(1 - [\mathrm{TORC1}])) - [\mathrm{Gln3}])
\end{aligned}$$

$$\frac{\mathrm{d}[\mathrm{Gln1}]}{\mathrm{d}t} = \gamma_{gln1}([\mathrm{Gln1}_T]\mathcal{H}(\sigma_{gln1}, \omega_{gln1-gln3}[\mathrm{Gln3}] - \omega_{gln1}) - [\mathrm{Gln1}])$$

$$\frac{\mathrm{d}[\mathrm{Rtg13}]}{\mathrm{d}t} = \gamma_{rtg13}([\mathrm{Rtg13}_T]\mathcal{H}(\sigma_{rtg}, -\omega_{rtg-torc}[\mathrm{TORC1}] + \omega_{rtg}) - [\mathrm{Rtg13}])$$

$$\frac{\mathrm{d}[\mathrm{Gis1}]}{\mathrm{d}t} = \gamma_{gis1}([\mathrm{Gis1}_T]\mathcal{H}(\sigma_{gis1}, -\omega_{gis-pka}[\mathrm{PKA}] - \omega_{gis-sch}[\mathrm{Sch9}] + \omega_{gis}) - [\mathrm{Gis1}])$$

$$\frac{\mathrm{d}[\mathrm{Mig1}]}{\mathrm{d}t} = \gamma_{mig}([\mathrm{Mig1}_T]\mathcal{H}(\sigma_{mig1}, \omega_{mig-pka}[\mathrm{PKA}] - \omega_{mig-snf}[\mathrm{Snf1}] + \omega_{mig}) - [\mathrm{Mig1}])$$

$$\frac{\mathrm{d}[\mathrm{Dot6}]}{\mathrm{d}t} = \gamma_{dot6}([\mathrm{Dot6}_T]\mathcal{H}(\sigma_{dot}, -\omega_{dot-sch-pka}[\mathrm{Sch9}][\mathrm{PKA}] + \omega_{dot}) - [\mathrm{Dot6}])$$

$$\frac{\mathrm{d}[\mathrm{Tps1}]}{\mathrm{d}t} = \gamma_{tps}([\mathrm{Tps1}_T]\mathcal{H}(\sigma_{tps}, \omega_{tps-pka}([\mathrm{PKA}_T] - [\mathrm{PKA}]) - \omega_{tps}) - [\mathrm{Tps1}])$$

$$\frac{\mathrm{d}[\mathrm{Trehalase}]}{\mathrm{d}t} = \gamma_{tre}([\mathrm{Trehalase}_T]\mathcal{H}(\sigma_{trehalase}, \omega_{tre-pka}[\mathrm{PKA}] - \omega_{tre}) - [\mathrm{Trehalase}])$$

$$\frac{\mathrm{d}[\mathrm{Protein}]}{\mathrm{d}t} = k_{pr}[\mathrm{ATP}]\min(\min([\mathrm{Rib}], [\mathrm{eIF}]), \min([\mathrm{tRNA}_{total}], [\mathrm{Glutamin}]))[\mathrm{Protein}]$$

$$\frac{\mathrm{d}[\mathrm{Rib}]}{\mathrm{d}t} = k_{transcription}(1 - [Dot6]) - k_{mRNA-degr}[\mathrm{Rib}]$$

B Återskapande av figurer för att verifiera implementering

Efter att modellen från appendix A har implementerats återskapas ett antal resultat från Jalihal m.fl. för att verifiera korrekt implementering [1]. Modellresultaten återskapas och jämförs med samma experimentella data. Informationen som behövs för att återskapa resultaten presenteras i första avsnittet nedan. Därefter presenteras resultaten av simuleringarna i avsnittet efteråt. Se även avsnitt 3.1 för en översiktlig beskrivning över hur återskapandet av figurena genomförts.

B.1 Data för återskapande av figurer

För att återskapa figurerna används förändringar i tillgångar på näringsämnen, eller näringsskiften. Hur modellen förändras under dessa näringsskiften presenteras nedan i tabell B.1 för vardera simulering. Beteckningarna är på engelska eftersom dessa användes vid programmeringen och i orginalartikeln för modellen [1].

Tabell B.1: Värden på de variabler som förändras före och efter skifte för de olika simuleringarna. Carbon, ATP och Glutamine$_{ext}$ motsvarar näringsförhållanden hos cellen.

Simulering	Före skifte	Efter skifte
Snf1: Glukossvältning	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0	Carbon = 0.0, ATP = 0.0, Glutamnine$_{ext}$ = 1.0
cAMP: Glukostillsättning	Carbon = 0.0, ATP = 0.0, Glutamine$_{ext}$ = 1.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0
cAMP: Sch9Δ	Carbon = 0.0, ATP = 0.0, Glutamine$_{ext}$ = 0.0, Sch9$_T$ = 0.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0, Sch9$_T$ = 0.0
Sch9: Kvävetillsättning (hög)	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 0.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0
Sch9: Kvävetillsättning (låg)	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 0.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 0.3
Sch9: gtr1Δ	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 0.0, EGO$_T$ = 0.0, $w_{torc-ego}$ = 0.0, $w_{torc-egoin}$ = 0.0, EGO = 0.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0
Sch9: Glukossvältning	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0	Carbon = 0.0, ATP = 0.0, Glutamine$_{ext}$ = 1.0
Sch9: Glukostillsättning	Carbon = 0.0, ATP = 0.0, Glutamine$_{ext}$ = 1.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0
Mig1: Glukostillsättning	Carbon = 0.0, ATP = 0.0, Glutamine$_{ext}$ = 1.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0
Rib: Rapamycin behandling	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0	Carbon = 1.0, ATP = 1.0, Glutamine$_{ext}$ = 1.0, TORC1$_T$ = 0.0

I de återskapade figurerna är även experimentell data inkluderad. Denna data har hämtats från Jalihal m.fl., då författarna genomförde literaturstudien, men de referenser som återfinns i kolumnen ”experiment” är till originalkällan för den data som presenteras i samma rad. I avsnitt 3.1 återfinns en utförlig beskrivning över hur återskapandet av figurena genomförts. I avsnitt 4.1 ges en kort beskrivning av resultaten från återskapandet av figurerna och i appendix B.2 presenteras de figurer som återskapats.

Tabell B.2: Experimentell data över tid med respektive värde.

Experiment	Tidpunkter [min]	Värden	Enhet
Snf1: Glukossvältning [32]	0.39, 5.55, 30.56, 60.64	0.03, 0.26, 0.32, 0.302	$\frac{\text{nmol}}{\text{min}\cdot\text{g}}$
cAMP: Glukostillsättning [33]	0.003, 0.255, 0.511, 0.748, 0.999, 1.247, 1.495, 1.737, 1.996, 2.490, 3.001	0.079, 0.632, 0.822, 1.197, 1.227, 0.928, 0.617, 0.617, 0.632, 0.209, 0.282	$\frac{\text{nmol}}{\text{g}}$
cAMP: Sch9Δ Glukostillsättning [33]	0.0025, 0.2495, 0.5015, 0.746, 0.9952, 1.2457, 1.4918, 1.7423, 1.9932, 2.4965, 3.0047	0.052, 0.112, 0.67, 0.259, 0.484, 0.375, 0.264, 0.279, 0.301, 0.275, 0.259	$\frac{\text{nmol}}{\text{g}}$
Sch9: Kvävetilsättning (hög) [34]	0.06, 0.36, 0.73, 1.05, 3.07, 4.12, 5.06, 8.05, 11.03, 15.07, 30.06	5.92, 10.23, 34.29, 41.14, 40.59, 39.82, 32.99, 13.09, 17.25, 35.4, 52.57	% Sch9 fosforylering
Sch9: Kvävetillsättning (låg) [34]	0.46, 1.09, 1.57, 2.04, 3.01, 4.05, 5.03, 8.05, 11.04, 15.01, 29.98	10.23, 49.55, 45.84, 27.66, 6.55, 5.49, 5.22, 5.27, 4.82, 4.89, 5.23	% Sch9 fosforylering
Sch9: gtr1Δ[34]	0.15, 0.7, 1.04, 1.51, 2.06, 3.05, 4.04, 5.06, 8.06, 11.0, 15.04, 30.02	7.96, 8.48, 8.48, 7.44, 7.44, 7.31, 7.44, 7.96, 9.64, 12.49, 25.7, 51.72	% Sch9 fosforylering
Sch9: Glukossvältning [35]	0.0,2.5,5.0,15.0,30.0	1.0,0.34,0.13,0.15,0.20	Rel. Sch9 fosforylering
Sch9: Glukostillsättning [35]	0.0,2.0,5.0,15.0,30.0	0.20,0.63,0.92,0.91,1.0	Rel. Sch9 fosforylering
Mig1: Glukostillsättning [36]	0, 1.1167, 1.7333, 2.1667, 3.8833, 5.5667, 7.7667, 10.1000, 12.8500, 15.3167, 18.1333	1.070, 1.233, 1.363, 1.428, 1.490, 1.465, 1.432, 1.425, 1.416, 1.400, 1.402	$\log\left(\frac{\text{nMig1}}{\text{cMig1}}\right)$
Rib: Rapamycinbehandling [37]	0.0,15.0,30.0,60.0,90.0	1.00,0.43,0.19,0.09,0.07	Rel. RPL32 mRNA

Den experimentella data som används för att återskapa figurerna normaliseras med en minimimaximum funktion för att sedan jämföras med modellresultatet. Detta görs för att anpassa data till den relativa skala från 0 till 1 som används i modelleringen och figurerna. Denna normaliseringsfunktion formuleras enligt

$$x_{\text{norm}} = \frac{x - \min}{\max - \min}, \tag{B.1}$$

där x är det ursprungliga värdet, x_{norm} är det normaliserade värdet, min är minimivärdet i den mätserie som ska normaliseras, och max är det motsvarande maximala värdet. För de signalmolekyler (cAMP och Sch9) som har data från flera experiment (dvs. flera mätserier) kommer värdena på min och max tas från alla uppmätta värden för den signalmolekylen.

Data för Mig1 under glukostillsättning normaliseras inte på grund av den logaritmiska skalan och för att kvoten, som motsvarar andelen Mig1 i cellkärnan (n för *nucleus*) relativt i cytosolen (utanför kärnan), eliminerar enheten för mätvärdena. Motsvarande beräkning görs i modellen, där Mig1 i modellen motsvarar lokalisering i cellkärnan och cMig1 fås genom $1 - \text{Mig1}$.

På grund av att data normaliseras och att modellresultatet inte har någon enhet kommer de enheter som presenteras i tabell B.2 och tabell B.5 inte vara av särskilt intresse. När modellen använder sig av den relativa skalan 0 till 1 kan den antas vara proportionell mot skalan hos experimentella data (såsom koncentration eller massa) när vi normaliserar enligt ekvation (B.1).

Utöver den data som presenterats ovan användes även den rapporterade parametervektorn från Jalihal m.fl. för att återskapa simuleringarna som redovisas i figur B.1. Parametervärdena redovisas nedan i tabell B.3 [1]. Notera att parameterarnas namn är på engelska eftersom dessa beteckningar användes i programmeringen och i orginalartikeln.

Tabell B.3: Rapporterad parametervektor från orginalartikeln [1].

Parameter	Värde	Parameter	Värde	Parameter	Värde
gammacyr	8.96	$Cyr1_T$	1.0	$w_{cyr-glu}$	5.13
$w_{cyr-snf}$	0.12	$sigma_{cyr}$	3.5	w_{cyr}	1.35
w_{dot}	0.29	$sigma_{dot}$	20.0	$w_{dot-sch-pka}$	0.16
$Dot6_T$	1.0	w_{ego}	0.28	$sigma_{ego}$	5.0
$w_{ego-basal}$	0.01	EGO_T	1.0	gammaego	50.66
$w_{ego-gap}$	2.21	gammagap	0.56	$sigma_{gap}$	1.0
$w_{gap-torc}$	88.33	w_{gap-N}	7.76	$EGOGAP_T$	1.0
$gamma_{gcn2}$	4.71	$Gcn2_T$	1.0	$w_{gcn-torc}$	1.29
$sigma_{gcn2}$	20.0	w_{gcn}	0.12	$w_{gcn4-gcn2-trna}$	1.53
w_{gcn4}	0.74	$tRNA_{sensitivity}$	74.51	$sigma_{gcn4}$	5.0
$tRNA_{total}$	2.47	$Gcn4_T$	1.0	$sigma_{gis1}$	10.0
w_{gis}	1.3	$Gis1_T$	1.0	$w_{gis-pka}$	3.3
$w_{gln1-gln3}$	0.52	$Gln1_T$	1.0	w_{gln1}	0.22
gammagln1	0.06	$sigma_{gln1}$	1.0	$w_{gln-snf}$	3.9
$sigma_{gln}$	10.0	$w_{gln-sit}$	0.86	$Gln3_T$	1.0
w_{gln3}	0.64	gammagln3	0.08	$k_{acc-pro}$	0.0
$k_{acc-glu}$	0.05	k_{degr}	0.09	k_{acc_nh4}	0.0
NH4	0.0	Proline	0.0	$w_{mig-snf}$	1.21
w_{mig}	10.64	$sigma_{mig1}$	0.27	$Mig1_T$	1.0
$w_{gis-sch}$	0.84	$gamma_{mig}$	0.66	$w_{mig-pka}$	2.31
$sigma_{pde}$	1.9	gammapde	0.28	$w_{pde-pka}$	2.89
PDE_T	1.0	w_{pde}	0.38	w_{pka}	0.06
$w_{pka-sch9}$	17.5	PKA_T	1,0	$w_{pka-camp}$	102.11
$sigma_{pka}$	1.0	gammapka	2.68	k_{pr}	0.02
$w_{ras-pka}$	1.87	$sigma_{ras}$	1.0	Ras_T	1.0
gammaras	1.82	$w_{ras-glu}$	0.21	w_{ras}	0.02
$k_{mRNA-degr}$	0.07	$k_{transcription}$	0.24	w_{rtg}	0.19
$sigma_{rtg}$	10.0	$w_{rtg-torc}$	0.88	$Rtg13_T$	1.0
w_{sak}	0.21	Sak_T	1.0	$w_{sak-pka}$	0.38
$sigma_{sak}$	20.0	$w_{sch9-torc}$	1.96	w_{sch9}	0.57
gammasch9	4.63	$sigma_{sch9}$	8.0	$Sch9_T$	1.0
$w_{snf-sak}$	1.52	gammasnf	0.82	$w_{snf-glc}$	1.15
$sigma_{snf}$	3.0	w_{snf}	0.54	$Snf1_T$	1.0
$w_{torc-egoin}$	0.3	$TORC1_T$	1.0	$w_{torc-ego}$	0.88
w_{torc}	0.54	$w_{torc-snf}$	0.44	$w_{torc-glut}$	0.86
$sigma_{tor}$	5.0	gammator	7.55	$w_{tps-pka}$	0.57
$sigma_{tps}$	5.0	gammatps	0.47	w_{tps}	0.05
PKA_T	1.0	$Tps1_T$	1.0	w_{tre}	1.07
gammatre	0.34	$w_{tre-pka}$	3.07	$Trehalase_T$	1.0
$sigma_{trehalase}$	10.0	$k_{camp-cyr}$	10.87	$k_{camp-deg}$	0.08
$k_{camp-pde}$	14.12	eIF_T	1.0	$w_{eif-gcn2}$	0.28
$sigma_{eif}$	1.0	gammaeif	0.47	w_{eif}	3.73
Carbon	1.0	ATP	1.0	$Glutamine_{ext}$	1.0
$gamma_{gcn4}$	1.0	$gamma_{rtg13}$	1.0	$gamma_{gis1}$	1.0
$gamma_{dot6}$	1.0				

Tabellerna B.1 och B.2 ovan hör till tidsserieexperimenten, medan de tabeller som presenteras nedan hör till störningsexperimenten. I detta sammanhang innebär en störning antingen en mutation hos cellen eller en behandling av cellerna med ett ämne som inaktiverar någon del av systemet, exempelvis rapamycin. De simuleringar som används för att skapa störningsexperiment presenteras nedan i tabell B.4.

Tabell B.4: Simuleringar som används vid störningsexperimentet.

Simulerings namn	Före skifte	Efter skifte
Glukossvältning	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 1.0$	Carbon = 0.0, ATP = 0.0, $\text{Glutamine}_{ext} = 1.0$
Glukostillsättning	Carbon = 0.0, ATP = 0.0, $\text{Glutamine}_{ext} = 1.0$	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 1.0$
Kvävetillsättning (hög)	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 0.0$	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 1.0$
Kvävesvältning	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 1.0$	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext}0.0$

Information om näringsskiften är inte det enda som krävs för att simulera störningsexperimenten. Det krävs även experimentell data för simuleringarna. Denna data presenteras nedan i en ytterligare tabell. På samma sätt som i den tidigare tabellen med experimentell data återfinns även här referenser till varje experiment i kolumnen ”experiment”. Referenserna är till originalkällan för data, medan värdena hämtades från Jalihal m.fl. [1], som gjorde litteraturstudien.

Tabell B.5: Data för att genomföra simulering av störningsexperimenten.

Experiment	Simulering	Mutation	mutant (pre-, postskifte)	vildtyp (pre-, postskifte)	Enhet
$\text{Snf1}_{sak1tos3elm1}$ [38]	Glukossvältning	$\text{Sak}_T = 0.0$	0.1, 0.1	0.05, 2.25	$\frac{\text{nmol}}{\text{min·mg}}$
Gis1_{sch9} [33]	Glukossvältning	$\text{Sch9}_T = 0.0$	100.0, 10.0	700.0, 35.0	% CTT1 gen uttryck
Trehalase_{tpk3} [39]	Glukostillsättning	$\text{PKA}_T = 0.3$	50.0, 25.0	100.0, 25.0	$\frac{\text{nmol}}{\text{min·mg}}$
cAMP_{pde} [40]	Glukostillsättning	$\text{PDE}_T = 0.0$	0.25, 3.8	0.25, 1.1	$\frac{\text{nmol}}{\text{g}}$
cAMP_{ras} [41]	Glukostillsättning	$\text{Ras}_T = 0.0, w_{pka-camp} = 0.0$	0.25, 0.25	0.25, 1.1	$\frac{\text{nmol}}{\text{g}}$
Rib_{sch9} [33]	Kvävetillsättning (hög)	$\text{Sch9}_T = 0.0$	100.0, 100.0	150.0, 600.0	% RPL25-uttryck
$\text{Sch9}_{lst4-lst7}$ [42]	Kvävetillsättning (hög)	$\text{EGOGAP}_T = 0.0$	1.0, 19.0	2.0, 73.0	% TORC1 aktivitet som Sch9P
$\text{Sch9}_{gtr1-gtr2}$ [42]	Kvävetillsättning (hög)	$\text{EGO}_T = 0.0, w_{torc-ego} = 0.0, w_{torc-egoin} = 0.0, w_{torc-glut} = 0.5$	22.0, 12.0	2.0, 73.0	% TORC1 aktivitet som Sch9P
Sch9_{gtr1-2} [34]	Kvävetillsättning (hög)	$\text{EGO}_T = 0.0, w_{torc-ego} = 0.0, w_{torc-egoin} = 0.0$	5.0, 55.0	5.0, 55.0	%TORC1 aktivitet som Sch9P
Gcn4_{gcn2} [43]	Kvävesvältning	$\text{Gcn2}_T = 0.0$	11.0, 38.0	7.3, 100.0	GCN4-lacZ pull-down
Gln3_{sit4} [44]	Glukossvältning	$\text{w}_{gln-sit} = 0.0, TORC1_T = 0.0$	0.0, 789.0	461.0, 6248.0	GST-URE2 pull-down

B.2 Visualisering av återskapade figurer

Nedan, i figur B.1, presenteras de resultat som återskapats från Jalihal m.fl. [1] genom att använda den information och data som presenteras i appendix B.1 och det genomförande som beskrivs i avsnitt 3.1. Syftet är att validera att modellen från Jalihal m.fl. implementerats korrekt.

Figuren består av 10 simuleringar som jämförs med den tidsseriedata som presenteras i tabell B.2

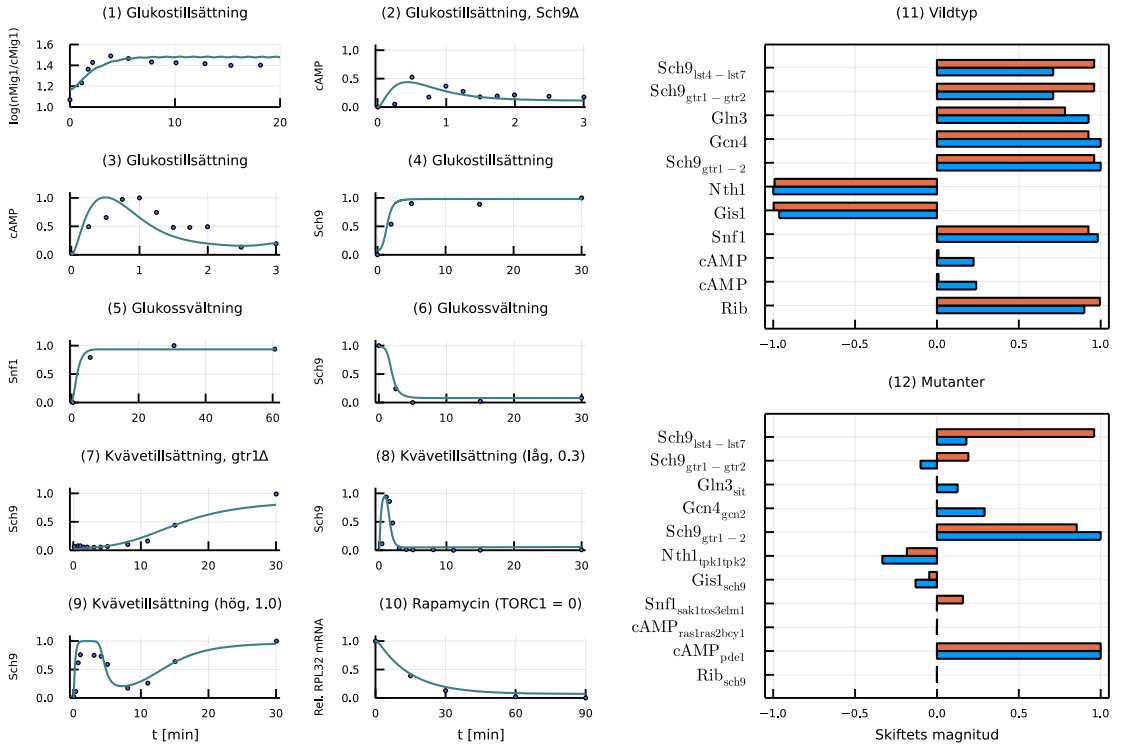
och resultat från simulering av störningsexperimenten. Totalt blir detta 12 paneler, där paneler 1-10 visar resultat från simulering av tidsserieexperimenten och panelerna (11) och (12) motsvarar resultat från simulering av störningsexperimenten, där panel (11) är för celler av vildtyp och panel (12) är för mutanter.

Cirklarna i panelerna 1-10 visar experimentell data för vardera simulering tillsammans med lösningen för den specifika signalmolekylen. X-axeln visar tidskalan för simuleringen mätt i minuter. Y-axeln visar mängden av den specifika signalmolekylen enligt den relativa skalan från 0 till 1, där 1 motsvarar den maximala uppmätta mängden av molekylen och 0 den minsta uppmätta. Normalisering av experimentell data för att anpassa den till denna relativa skala genomförs med ekvation (B.1) och enligt den beskrivning som presenteras i appendix B.1. Ekvationen normaliserar alltså data över signalmolekylen utifrån dess maximala och minimala uppmätta värde, så att den passar in i y-axelns skala från 0 till 1.

I panel (1) visas lokalisering av Mig1 i cellen under glukostillsättning. I panel (2) simuleras samma fall, men istället observeras nivåer av cAMP för mutanten Sch9Δ. Panel (3) i figuren visar scenariot med glukostillsättning, återigen för cAMP men nu för en cell av vildtyp. I panel (4) simuleras glukostillsättning återigen, men nu för Sch9. Snf1 och Sch9 observeras i panel (5) respektive (6) under glukossvältning.

I panel (7) simuleras kvävetillsättning, där Sch9 observeras för mutanten gtr1Δ. I panel (8) och (9) observeras återigen Sch9 under kvävetillsättning, men här för en vildtypcell, och där det i panel (8) har tillsatts en låg mängd kväve (0.3 på den relativa skalan). Slutligen observeras i panel (10) relativt uttryck av RPL32 mRNA (motsvarar Rib i modellen) under näringsrik miljö då rapamycin har tillsatts.

I panel (11) och (12) visas resultat från simuleringar av störningsexperimenten i form av stapeldiagram. De två figurerna möjliggör en jämförelse av hur störningar av olika typer (främst mutationer) påverkar resultatet för de olika signalmolekylerna. Panel (11) visar resultat för vildtyp medan panel (12) visar för störda celler (här kallade ”mutanter”). Längden på staplarna motsvarar differensen mellan stationärvärden hos signalämnet före och efter skiftet i näringsförhållanden. Om stationärvärdet är lägre efter skiftet än före kommer differensen alltså bli negativ, vilket gör att staplarna pekar åt vänster.



Figur B.1: Återskapade resultat från Jalihal m.fl. i form av de figurer som presenterades i figur 2 i originalartikeln. Till vänster syns simulering av tidsserieexperimenten, som redovisas i tabell B.2. Till höger redovisas simulering av störningsexperiment, som redovisas i tabell B.5. Längden på staplarna motsvarar differensen mellan stationärvärdena hos signalämnet före och efter ett skifte. De orangea staplarna visar resultat från simuleringen, medan de blå staplarna visar experimentell data. I panel (12) visas resultat för de störda cellerna (här förenklade som ”mutanter”) och panel (11) visar resultat från celler av vildtyp. De skiften som görs för de olika fallen återfinns i tabell B.5. Det observeras att simuleringsresultaten för mutanterna lst4Δlst7Δ och gtr1Δgtr2Δ skiljer sig från experimentell data. Dessa två simuleringsresultat överensstämmer inte med resultatet från Jalihal m.fl. och är de enda resultaten som inte kunde återskapas.

Vid en jämförelse mellan de återksapde figurerna presenterat ovan och figurerna i artikeln av Jalihal m.fl. konstateras att två simuleringar i störningsexperimentet panel (12) inte överensstämmer [1]. Dessa simuleringar var av mutanterna lst4Δlst7Δ och gtr1Δgtr2Δ för signalämnet Sch9. Detta resultat diskuteras i avsnitt 5.

C Parameteruppskattningen

Innan parameteruppskattningen genomförs för parametrarna i ODE-systemet i avsnitt A utvecklas en optimeringskod för ett enklare exempel. Denna process beskrivs i avsnitt C.1. Från parameteruppskattningen hittades två parametervektorer som gav en relativt låg kostnad och därmed ger en bra beskrivning av data. Parametervärdena för vardera parametervektor kan hittas i avsnitt C.3. En visualisering över hur parametervektorerna påverkar kostnaden för vardera dataset samt hur ODE-lösningen ser ut för dessa kan ses i avsnitt C.2. En värmekarta över de parvisa beroendena mellan parametrarna i de två erhålla parametevektorerna och den från Jalihal m.fl. visualiseras därefter i avsnitt C.4.

C.1 Optimeringsexempel

Vi beslutade att först implementera optimeringsalgoritmen för ett enklare exempel. Vi börjar med att definiera ett ODE system som beskriver följande reaktion

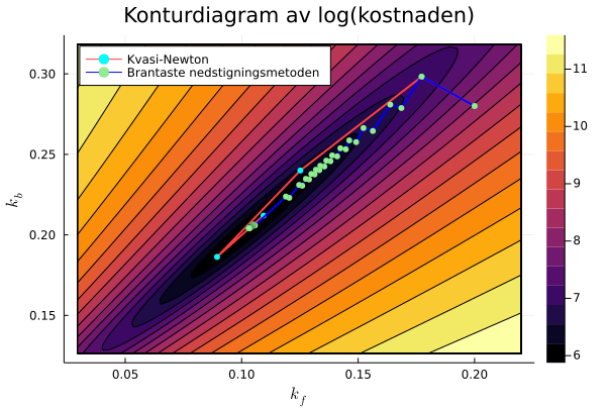
$$\mathrm{A} \underset{\mathrm{k}_b}{\overset{\mathrm{k}_f}{\rightleftharpoons}} \mathrm{B},$$

som kan beskrivas med ODE-systemet

$$\begin{aligned} \frac{\mathrm{d}[A]}{\mathrm{d}t} &= k_b[B] - k_f[A], \\ \frac{\mathrm{d}[B]}{\mathrm{d}t} &= k_f[A] - k_b[B]. \end{aligned}$$

Vi väljer att $k_f = 0{,}1$, $k_b = 0{,}2$, tidsintervallet mellan 0 och 10 tidsenheter samt $A_0 = 50$ och $B_0 = 100$. För att kunna göra parameteruppsakattning på detta system behöver vi generera datapunkter längs lösningen som kan användas till att beräkna en kostnad. Vi väljer att applicera normalfördelat brus på datan. 50 datapunkter jämnt fördelade över tid simuleras för vardera ODE, vilket ger totalt 100 datapunkter. Detta säkerställer att den optimala lösningen är relativt nära de parametervärden som valts.

Vi väljer att starta optimeringsproceduren i punkten där $k_f = 0{,}2$ och $k_b = 0{,}28$. Vi väljer även $\mu = 10^{-4}$, $\nu = 0{,}9$, $\Delta\alpha = 10^{-7}$ samt $\varepsilon_1, \varepsilon_2, \varepsilon_3 = 10^{-6}$. Resultatet från optimeringen kan illustreras i ett konturdiagram där konturlinjerna motsvarar olika värden på logaritmen av kostnaden och parametrarna motsvarar x- och y-axeln. Optimeringsexemplet genomfördes för både brantaste nedstigningsmetoden och kvasi-Newtonmetoden även om brantaste nedstigningsmetoden teoretiskt sett ska vara sämre eftersom den inte tar hänsyn till kurvatur. Nedan i figur C.1 visas båda lösningar av optimeringen.



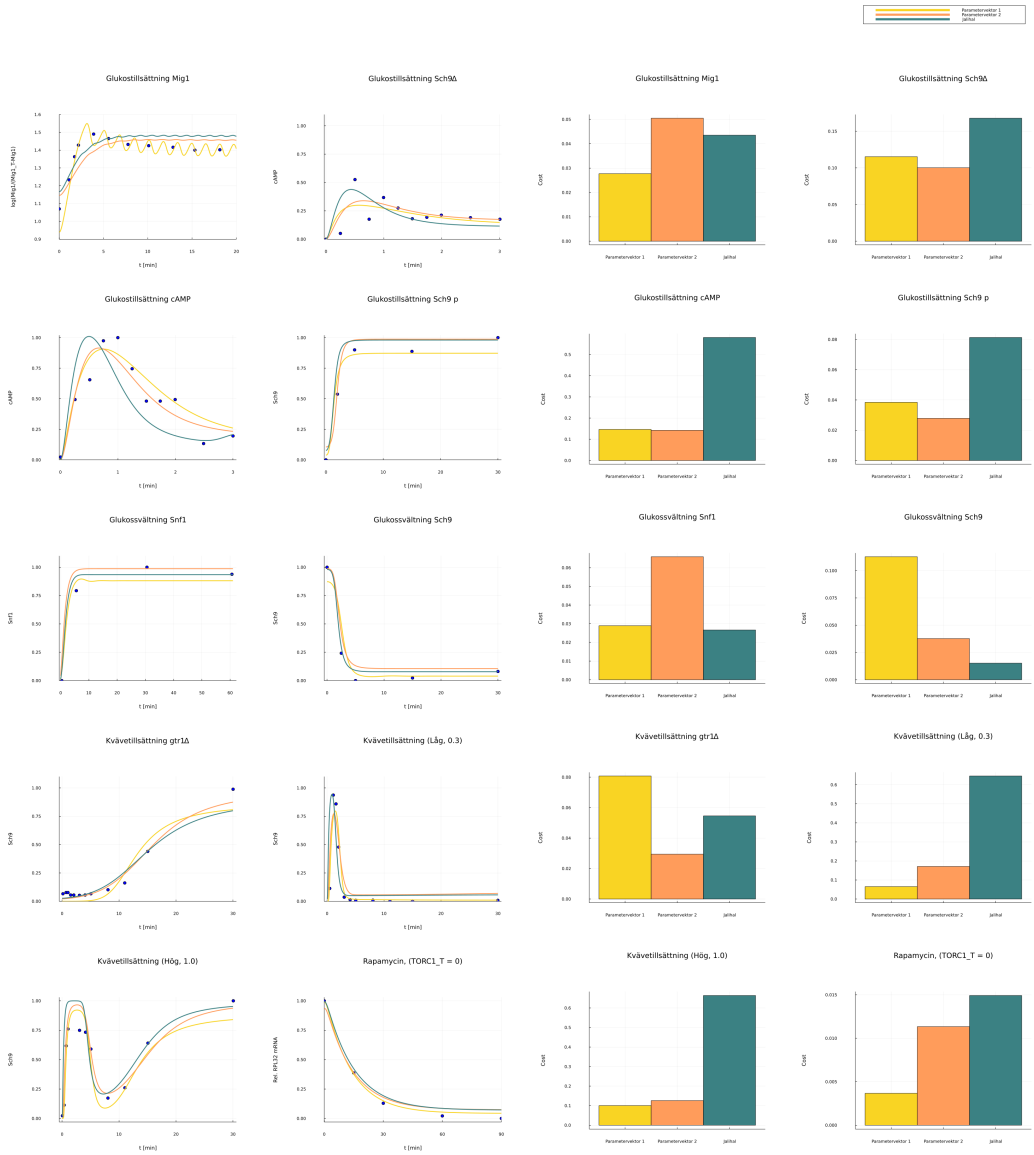
Figur C.1: Hur kvasi-Newtonmetoden och brantaste nedstigningsmetoden närmar sig de optimala parametervärdena.

Det visade sig att kvasi-Newtonmetoden hittade parametervärden som uppfyllde två av de tre termineringskriterierna i avsnitt 2.3.4 efter 10 iterationer medans brantaste nedstigningsmetoden

tog 175 iterationer. Detta indikerar att optimeringskoden verkar fungera då den gick snabbare mot ett optimum än brantaste nedstigningsmetoden.

C.2 Visualisering av resultaten från parameteruppskattningen

På vänster sida i figur C.2 kan vi se hur lösningen för några olika variabler i ODE-systemet i appendix A ser ut för båda parametervektorerna från parameteruppskattningen och parametervektorn från Jalihal m.fl. För att underlätta avläsning av vilken parametervektor som beskriver datan bäst för vardera dataset finns även histogram som illustrerar kostnaderna på höger sida i figuren.



Figur C.2: Till vänster visas lösningen för tre parametervekroter illustrerat mot tio dataset och till höger visas vad kostnaden är för varje figur för vardera parametervektor.

C.3 Erhållna parametrar från parameteruppskattningen

Nedan i tabell C.1 och C.2 visas värdena på alla parametrar i parametervektor 1 och 2.

Tabell C.1: Parametervektor 1.

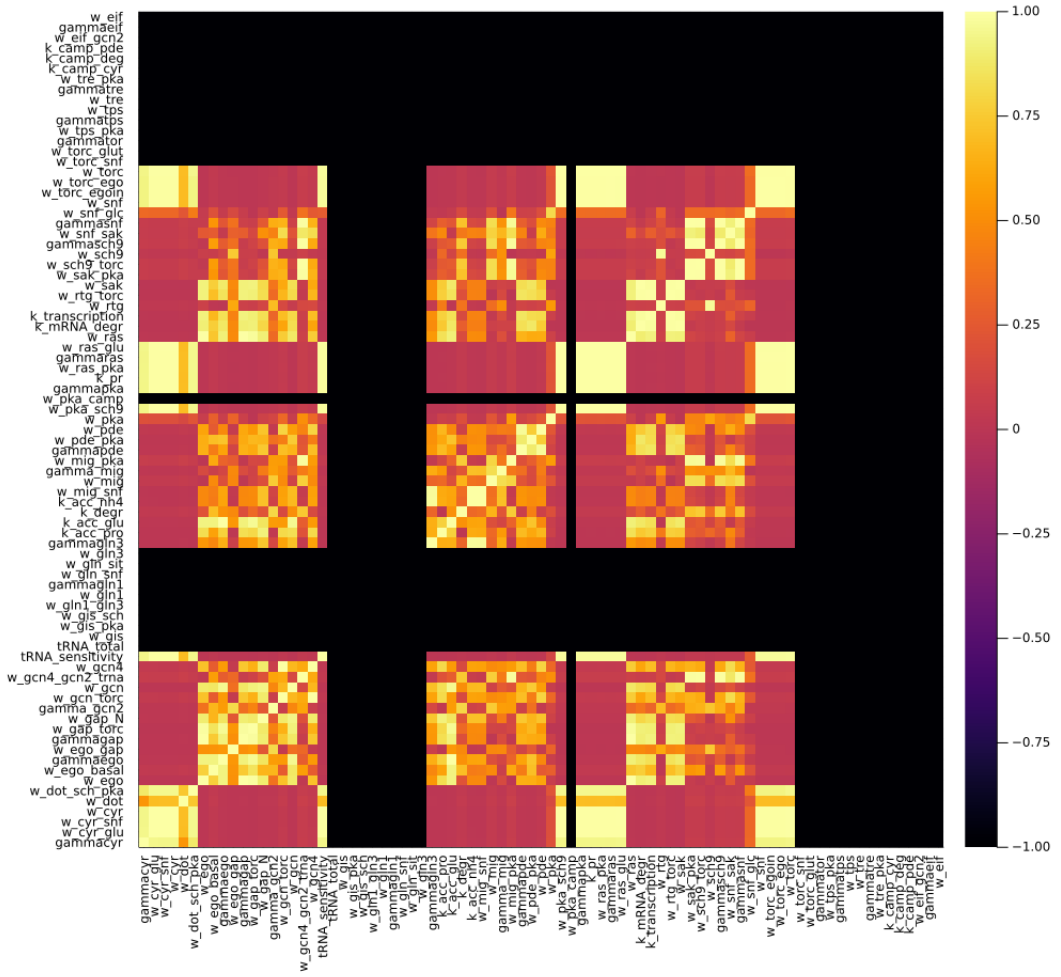
Parameter	Värde	Parameter	Värde	Parameter	Värde
gammacyr	18.3229	$Cyr1_T$	1.0	$w_{cyr-glu}$	3.6634
$w_{cyr-snf}$	0.1285	$sigma_{cyr}$	3.5	w_{cyr}	0.6059
w_{dot}	0.2986	$sigma_{dot}$	20.0	$w_{dot-sch-pka}$	0.2698
$Dot6_T$	1.0	w_{ego}	0.1443	$sigma_{ego}$	5.0
$w_{ego-basal}$	0.006814	EGO_T	1.0	gammaego	14.7367
$w_{ego-gap}$	3.0843	gammagap	0.6663	$sigma_{gap}$	1.0
$w_{gap-torc}$	65.8599	w_{gap-N}	34.2767	$EGOGAP_T$	1.0
$gamma_{gcn2}$	3.4453	$Gcn2_T$	1.0	$w_{gcn-torc}$	3.3064
$sigma_{gcn2}$	20.0	w_{gcn}	0.2345	$w_{gcn4-gcn2-trna}$	0.4912
w_{gcn4}	1.0432	$tRNA_{sensitivity}$	114.6681	$sigma_{gcn4}$	5.0
$tRNA_{total}$	1.2406	$Gcn4_T$	1.0	$sigma_{gis1}$	10.0
w_{gis}	2.0505	$Gis1_T$	1.0	$w_{gis-pka}$	7.2179
$w_{gln1-gln3}$	1.3958	$Gln1_T$	1.0	w_{gln1}	0.4671
gammagln1	0.06914	$sigma_{gln1}$	1.0	$w_{gln-snf}$	1.5788
$sigma_{gln}$	10.0	$w_{gln-sit}$	0.2405	$Gln3_T$	1.0
w_{gln3}	1.6030	gammagln3	0.04016	$k_{acc-pro}$	0.001312
$k_{acc-glu}$	0.05230	k_{degr}	0.09600	k_{acc_nh4}	0.002486
NH4	0.0	Proline	0.0	$w_{mig-snf}$	0.8162
w_{mig}	7.8159	$sigma_{mig1}$	0.27	$Mig1_T$	1.0
$w_{gis-sch}$	0.4426	$gamma_{mig}$	1.1644	$w_{mig-pka}$	6.3368
$sigma_{pde}$	1.9	gammapde	0.2057	$w_{pde-pka}$	3.9406
PDE_T	1.0	w_{pde}	0.9996	w_{pka}	0.1397
$w_{pka-sch9}$	41.4820	PKA_T	1,0	$w_{pka-camp}$	132.7603
$sigma_{pka}$	1.0	gammapka	1.5341	k_{pr}	0.08021
$w_{ras-pka}$	2.7963	$sigma_{ras}$	1.0	Ras_T	1.0
gammaras	1.1148	$w_{ras-glu}$	0.5930	w_{ras}	0.01297
$k_{mRNA-degr}$	0.07545	$k_{transcription}$	0.1527	w_{rtg}	0.06984
$sigma_{rtg}$	10.0	$w_{rtg-torc}$	0.6111	$Rtg13_T$	1.0
w_{sak}	0.2152	Sak_T	1.0	$w_{sak-pka}$	0.8124
$sigma_{sak}$	20.0	$w_{sch9-torc}$	1.7535	w_{sch9}	1.4099
gammasch9	2.9517	$sigma_{sch9}$	8.0	$Sch9_T$	1.0
$w_{snf-sak}$	1.1326	gammasnf	0.7244	$w_{snf-glc}$	1.4228
$sigma_{snf}$	3.0	w_{snf}	0.3246	$Snf1_T$	1.0
$w_{torc-egoin}$	0.6320	$TORC1_T$	1.0	$w_{torc-ego}$	1.0293
w_{torc}	0.3391	$w_{torc-snf}$	0.5627	$w_{torc-glut}$	1.5540
$sigma_{tor}$	5.0	gammator	4.7678	$w_{tps-pka}$	1.1822
$sigma_{tps}$	5.0	gammatps	1.0502	w_{tps}	0.01421
PKA_T	1.0	$Tps1_T$	1.0	w_{tre}	0.7170
gammatre	0.7687	$w_{tre-pka}$	8.4127	$Trehalase_T$	1.0
$sigma_{trehalase}$	10.0	$k_{camp-cyr}$	3.8239	$k_{camp-deg}$	0.02463
$k_{camp-pde}$	10.0626	eIF_T	1.0	$w_{eif-gcn2}$	0.4189
$sigma_{eif}$	1.0	gammaeif	1.3511	w_{eif}	2.9039
Carbon	1.0	ATP	1.0	$Glutamine_{ext}$	1.0
$gamma_{gcn4}$	1.0	$gamma_{rtg13}$	1.0	$gamma_gis1$	1.0
$gamma_{dot6}$	1.0				

Tabell C.2: Parametervektor 2.

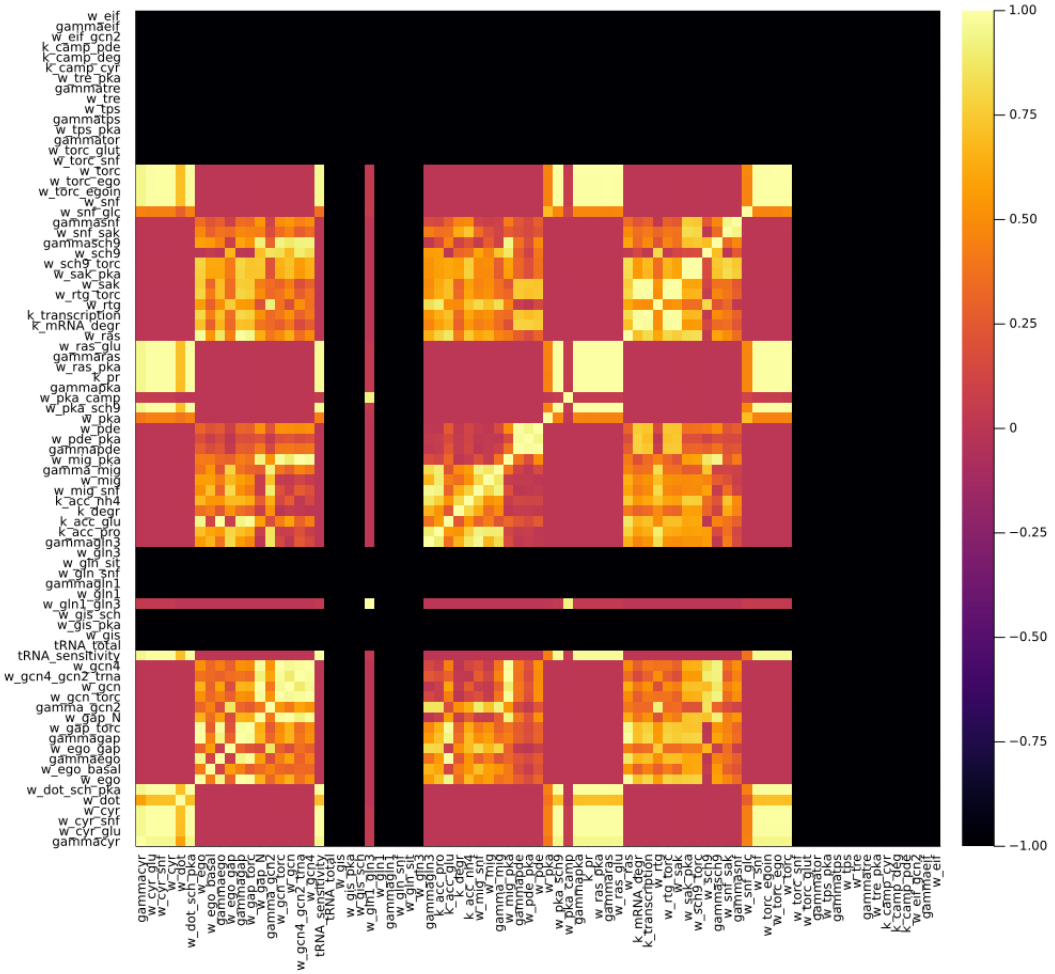
Parameter	Värde	Parameter	Värde	Parameter	Värde
gammacyr	7.3781	$Cyr1_T$	1.0	$w_{cyr-glu}$	4.0997
$w_{cyr-snf}$	0.1562	$sigma_{cyr}$	3.5	w_{cyr}	1.1855
w_{dot}	0.4112	$sigma_{dot}$	20.0	$w_{dot-sch-pka}$	0.1503
$Dot6_T$	1.0	w_{ego}	0.1764	$sigma_{ego}$	5.0
$w_{ego-basal}$	0.01210	EGO_T	1.0	gammaego	28.1264
$w_{ego-gap}$	3.2379	gammagap	0.5612	$sigma_{gap}$	1.0
$w_{gap-torc}$	74.8387	w_{gap-N}	11.8748	$EGOGAP_T$	1.0
$gamma_{gcn2}$	4.0903	$Gcn2_T$	1.0	$w_{gcn-torc}$	0.9896
$sigma_{gcn2}$	20.0	w_{gcn}	0.1145	$w_{gcn4-gcn2-trna}$	1.0379
w_{gcn4}	0.8080	$tRNA_{sensitivity}$	51.4753	$sigma_{gcn4}$	5.0
$tRNA_{total}$	1.4166	$Gcn4_T$	1.0	$sigma_{gis1}$	10.0
w_{gis}	0.9599	$Gis1_T$	1.0	$w_{gis-pka}$	2.2532
$w_{gln1-gln3}$	0.4266	$Gln1_T$	1.0	w_{gln1}	0.2321
gammagln1	0.07487	$sigma_{gln1}$	1.0	$w_{gln-snf}$	3.8833
$sigma_{gln}$	10.0	$w_{gln-sit}$	0.4999	$Gln3_T$	1.0
w_{gln3}	0.8292	gammagln3	0.05684	$k_{acc-pro}$	0.0009580
$k_{acc-glu}$	0.03827	k_{degr}	0.06110	k_{acc_nh4}	0.001063
NH4	0.0	Proline	0.0	$w_{mig-snf}$	1.1045
w_{mig}	10.6827	$sigma_{mig1}$	0.27	$Mig1_T$	1.0
$w_{gis-sch}$	0.3996	$gamma_{mig}$	0.63018	$w_{mig-pka}$	2.0703
$sigma_{pde}$	1.9	gammapde	0.2456	$w_{pde-pka}$	2.7405
PDE_T	1.0	w_{pde}	0.2457	w_{pka}	0.07652
$w_{pka-sch9}$	21.2618	PKA_T	1,0	$w_{pka-camp}$	92.3357
$sigma_{pka}$	1.0	gammapka	2.0363	k_{pr}	0.03686
$w_{ras-pka}$	1.7060	$sigma_{ras}$	1.0	Ras_T	1.0
gammaras	1.5457	$w_{ras-glu}$	0.4174	w_{ras}	0.02040
$k_{mRNA-degr}$	0.07321	$k_{transcription}$	0.3059	w_{rtg}	0.1148
$sigma_{rtg}$	10.0	$w_{rtg-torc}$	0.5694	$Rtg13_T$	1.0
w_{sak}	0.2138	Sak_T	1.0	$w_{sak-pka}$	0.3734
$sigma_{sak}$	20.0	$w_{sch9-torc}$	1.8292	w_{sch9}	0.6810
gammasch9	2.8481	$sigma_{sch9}$	8.0	$Sch9_T$	1.0
$w_{snf-sak}$	1.6515	gammasnf	0.7872	$w_{snf-glc}$	1.0491
$sigma_{snf}$	3.0	w_{snf}	0.1598	$Snf1_T$	1.0
$w_{torc-egoin}$	0.2107	$TORC1_T$	1.0	$w_{torc-ego}$	0.4533
w_{torc}	0.3985	$w_{torc-snf}$	0.4108	$w_{torc-glut}$	0.8002
$sigma_{tor}$	5.0	gammator	5.2170	$w_{tps-pka}$	0.4340
$sigma_{tps}$	5.0	gammatps	0.3365	w_{tps}	0.02768
PKA_T	1.0	$Tps1_T$	1.0	w_{tre}	1.0428
gammatre	0.2491	$w_{tre-pka}$	2.8906	$Trehalase_T$	1.0
$sigma_{trehalase}$	10.0	$k_{camp-cyr}$	6.8348	$k_{camp-deg}$	0.05288
$k_{camp-pde}$	11.1986	eIF_T	1.0	$w_{eif-gcn2}$	0.2139
$sigma_{eif}$	1.0	gammaeif	0.5727	w_{eif}	3.7183
Carbon	1.0	ATP	1.0	$Glutamine_{ext}$	1.0
$gamma_{gcn4}$	1.0	$gamma_{rtg13}$	1.0	$gamma_gis1$	1.0
$gamma_{dot6}$	1.0				

C.4 Parvisa beroenden mellan parametrarna

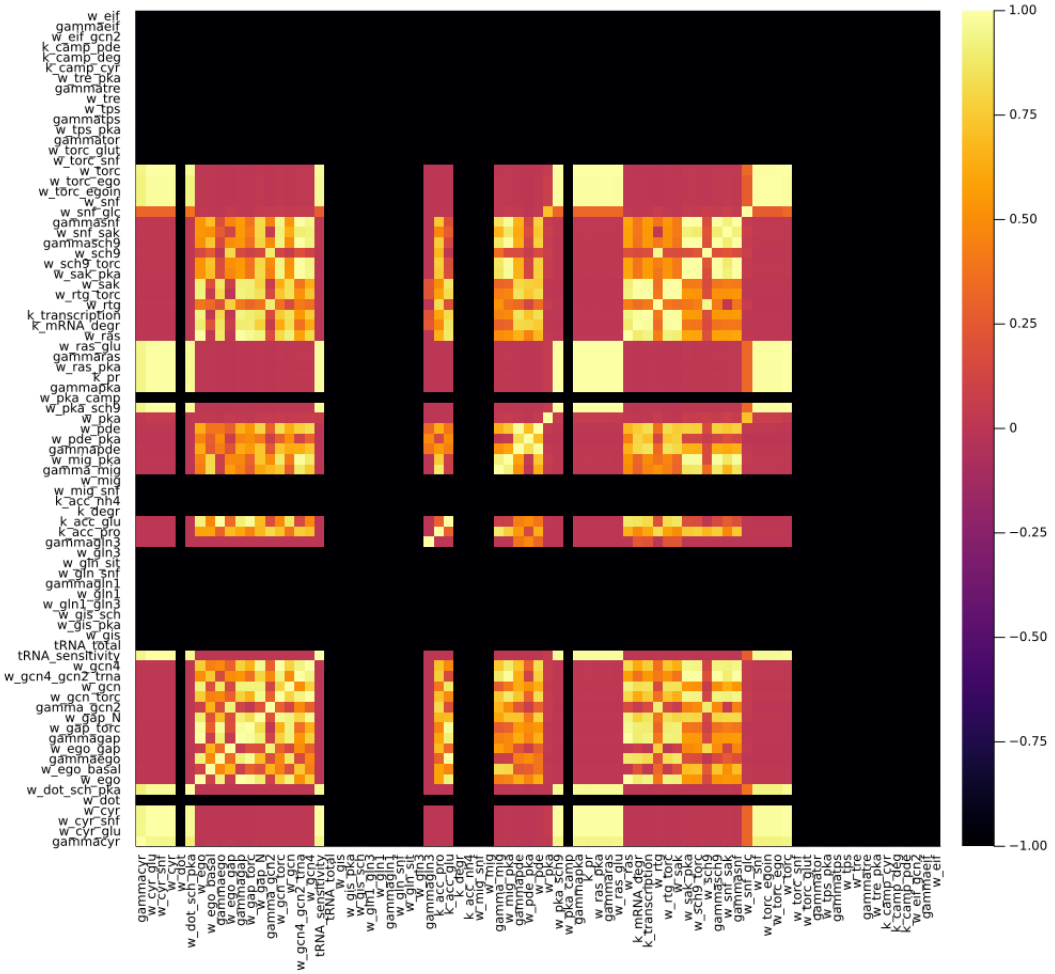
I figurerna C.3, C.4 och C.5 presenteras värmekartor över de parvisa beroendena mellan modellparametrarna. Detta görs för de två parametervektorer som erhölls från parameteruppskattingen och parametervektorn från Jalihal m.fl. i separata figurer C1-C3 nedan. Ju ljusare fält desto högre beroende. De par av känslighetsvektorer för vilka produkten av normen blev noll, det vill säga där ena eller båda parametrarna fick en längd exakt eller numeriskt lika med noll, motsvaras av svarta fält.



Figur C.3: Parvisa beroenden mellan parametrarna i parametervektor från Jalihal m.fl.



Figur C.4: Parvisa beroenden mellan parametrarna i parametervektor 1.



Figur C.5: Parvisa beroenden mellan parametrarna i parametervektor 2.

D Analys av modellens giltighet

För att analysera modellens giltighet användes kvalitativ data insamlad genom en litteraturstudie som simuleringsresultat jämfördes med. Denna data ger information om hur olika delar av systemet bör bete sig under olika förutsättningar. Den kvalitativa data som används i analysen presenteras nedan. Därefter följer en fullständig presentation av resultaten av jämförelserna mellan simuleringsresultat och data i en tabell. Vissa av resultaten presenteras individuellt i avsnitt 4.3 och diskuteras utförligt i avsnitt 5.

D.1 Kvalitativ data för analys av modellens giltighet

Den kvalitativa data som används för att analysera modellens giltighet gäller för ett flertal olika delar av systemet under olika näringsförhållanden. Dessa förhållanden är kol- och kvävesvältning samt kol- och kvävetillsättning. I modellen simuleras dessa näringsförhållanden med variation av nivåer av kvävekällan glutamin och glukos som är en kolkälla. Glukos motsvaras i modellen av ”ATP” och ”Carbon”. Se även 3.4 för en utförlig beskrivning av tillvägagångssättet för att analysera modellens giltighet samt 4.3 för en beskrivning av det sammanställda resultatet av analysen som presenteras i tabell D.4 nedan. Se även figur D.1 för en grafisk sammanställning av alla simuleringsresultat.

Den kvalitativ data som används för att analysera modellens giltighet innehåller både data från muterade celler och celler av vildtyp. Nedan i tabell D.1 och tabell D.2, presenteras kvalitativa data för vildtyp respektive mutanter tillsammans med referenser.

Tabell D.1: Kvalitativ data över vildtyp för analys av modellens giltighet.

Simulering	Signalmolekyl	Kvalitativ data	Källa
Glukossvältning	Snf1	Aktivering	[45]
Gukossvältning	Mig1	Minskad aktivitet	[45, 46]
Glukossvältning	Gis1	Aktiveras	[47, 48, 49]
Glukossvältning	Gln3	Ökad aktivitet	[24]
Glukossvältning	Dot6/Tod6	Ökad aktivitet	[50]
Kvävesvältning	TORC1	Minskad aktivitet	[34]
Kvävesvältning	Sch9	Inaktivering	[34]
Kvävesvältning	Gln3	Ökad aktivitet	[51, 5, 52, 53]
Kvävesvältning	Snf1	Ökad aktivitet	[26, 27]
Kvävesvältning	Rtg1/3	Ökad aktivitet	[54, 55, 56]
Kvävesvältning	Dot6/Tod6	Ökad aktivitet	[57]
Glukostillsättning	Snf1	Inaktivering	[58]
Glukostillsättning	Gis1	Inaktivering	[47, 48, 49]

Tabell D.2: Kvalitativ data över hur mutationerna påverkar signalmolekylens aktivitet. Data här beskriver alltså hur signalmolekylens aktivitet skiljer sig mot vildtyp.

Simulering	Signalmolekyl	Mutation	Förändring	Källa
Kvävesvältning	Gln3 (proxy för NCR-uttryck)	Gtr1Δ (raderar EGO)	Uttryck av NCR ökar (ökad Gln3-aktivitet)	[59]
Glukossvältning	Gln3	Snf1Δ	Ingen ökad aktivitet	[24]
Glukossvältning	Cyr1	Snf1Δ	Markant minskning av aktivitet	[25]

Simuleringarna som görs är de samma som i tabellen B.4 utöver kvävesvältning. Denna återfinns

istället nedan i tabell D.3. Notera att beteckningarna på variablerna är på engelska eftersom det var det var så beteckningarna såg ut vid programmeringen och i artikeln av Jalihal m.fl. [1].

Tabell D.3: Simulering för att genomföra analys av modellens giltighet.

Simulerings namn	Före skifte	Efter skifte
Kvävesvältning	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 1.0$	Carbon = 1.0, ATP = 1.0, $\text{Glutamine}_{ext} = 0.0$

D.2 Fullständiga resultat från analys av modellens giltighet

Genom att använda utförandet beskrivet i 3.4 och kvalitativ data från appendix D.1 genomfördes en analys av modellens giltighet. I analysen användes både parametervektorerna erhållna via paramateruppskattningen som presenterade i avsnitt 4.2 och parametervärdena från Jalihal m.fl. redovisade i tabell B.3. Resultatet från denna analys presenteras nedan i tabell D.4 för de simuleringar som genomförts och vardera parametervektor som använts.

Tabell D.4: Resultat från analys av modellens giltighet.

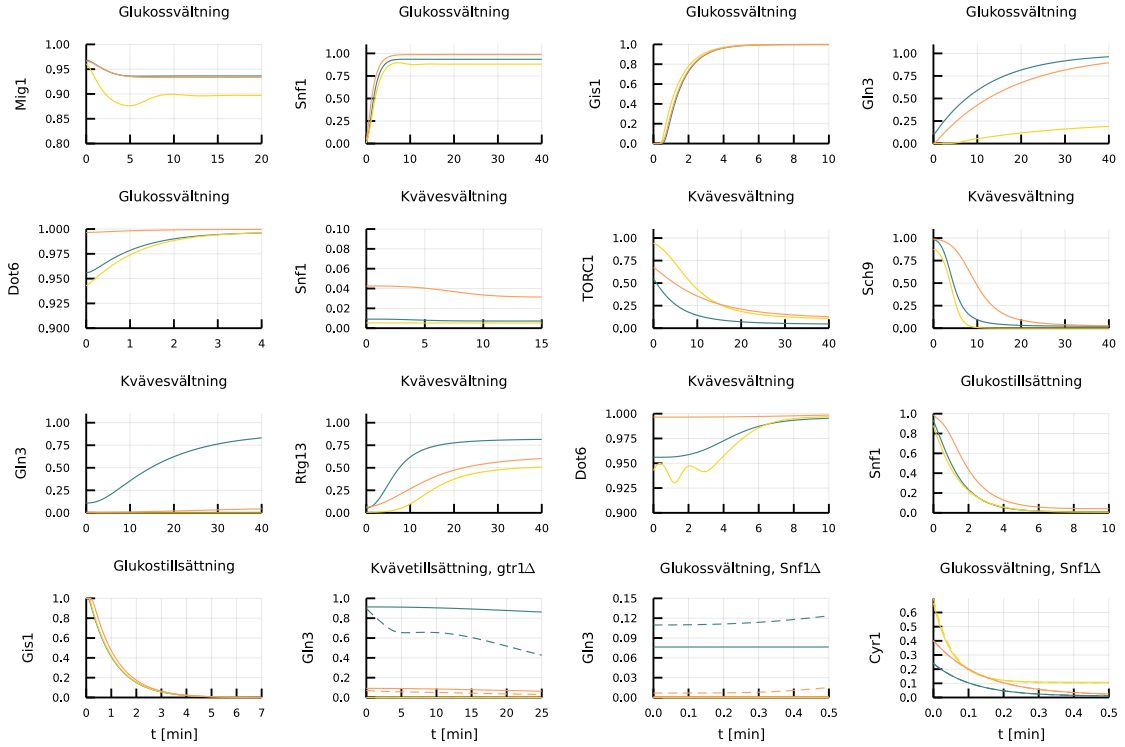
Simuleringstyp	Signalmolekyl	Jalihal m.fl.	P-vektor 1	P-vektor 2
Glukossvältning	Snf1	Sant	Sant	Sant
Glukossvältning	Mig1	Sant	Sant	Sant
Glukossvältning	Gis1	Sant	Sant	Sant
Glukossvältning	Gln3	Sant	Sant	Sant
Glukossvältning	Dot6/Tod6	Sant	Sant	Sant
Kvävesvältning	TORC1	Sant	Sant	Sant
Kvävesvältning	Sch9	Sant	Sant	Sant
Kvävesvältning	Gln3	Sant	Falskt	Falskt
Kvävesvältning	Snf1	Falskt	Falskt	Falskt
Kvävesvältning	Rtg1/3	Sant	Sant	Sant
Kvävesvältning	Dot6/Tod6	Sant	Sant	Sant
Glukostillsättning	Snf1	Sant	Sant	Sant
Glukostillsättning	Gis1	Sant	Sant	Sant
Kvävetillsätting	Gln3 (gtr1Δ)	Sant	Falskt	Sant
Glukossvältning	Gln3 (Snf1Δ)	Sant	Sant	Sant
Glukossvältning	Cyr1 (Snf1Δ)	Falskt	Falskt	Falskt

Generellt sett överensstämmer simuleringarna med kvalitativ data. Värt att notera är dock att det finns flera fall där det är intressanta *kvantitativa* aspekter av resultaten. Ett exempel är där förändringen sker och går i rätt riktning men är liten på den relativa skalan som används i modellen. Eftersom denna skala antas vara proportionell mot den skala som mäts laborativt förutsäger modellresultaten möjligen förändringar som inte kunnat detekteras i experiment. Eftersom vi kan konstatera att dessa resultat finns bör simuleringarna ge relativt stora skillnader på skalan som används i modellen. Dessa små kvantitativa skillnader observeras främst för Mig1 och Dot6/Tod6 under glukossvältning, och kan ses i figur D.1.

En annan kvantitativ aspekt som kan observeras är att det konsekvent är stor skillnad mellan resultaten från de olika parametervektorerna för Gln3. Detta diskuterades närmare i avsnitt 5 för fallet glukossvältning men det kan nämnas här att detta även observeras under kvävesvältning och till viss del under glukossvältning för mutanten Snf1Δ. Fallet Cyr1 under glukossvältning för mutanten Snf1Δ presenteras i avsnitt 4.3.3 och diskuteras i avsnitt 5.

Utöver en tabell som sammanställt resultaten av analysen presenteras även resultaten i form av figurer. Utvalda figurer presenteras i avsnitt 4.3 där fokus främst ligger på att lägga grunden för den diskussion som förs i avsnitt 5 kring hur modellen kan förbättras genom att betrakta några

av de fall som falsifierar modellen. Figurer från alla simuleringsresultat presenteras dock nedan i figur D.1 i redovisningssyfte.



Figur D.1: Resultat från de simuleringar som beskrivs i tabell D.3. Färgerna motsvarar resultat från de olika parametervektorerna, där Jalihal är grön, parametervektor 1 är gul och parametervektor 2 är orange. För simuleringarna som jämför aktivitet mellan mutant och simulering gäller att de streckade linjerna visar resultat för vildtyp och de heldragna linjerna för mutant. Notera att den betraktade delen av y-axeln skiljer sig från 0 till 1 för vissa av figurerna. Detta görs för att skillnader lättare ska kunna observeras.

E Kod

E.1 Länk till GitHub

GitHub-mappen med all kod från projektet nås via följande webbadress: https://github.com/Leoan-chalmers/KandidatProjekt.git eller genom att klicka här.

E.2 Kod relaterad till LHS

E.2.1 LHS

Code/Latin_Hypercube_Sampling/latin_hypercube_sampling.jl

```
using LatinHypercubeSampling
using LinearAlgebra
using DifferentialEquations
using DiffEqSensitivity
using ModelingToolkit
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")
include("../Parameter_Estimation/Optimization.jl")

"""
    Check_if_p_works(p_var, p_const, scaled_plan)

    Checks if the parameter sets from the latin hypercube works and saves
    them into a CSV file.
"""
function Check_if_p_works(p_var, p_const, scaled_plan)

    N = 0
    df = DataFrame()

    for item in 1:length(scaled_plan[:,1])

        print(" Iteration: ",item)

        p_values_process = exp10.(scaled_plan[item,:])
        #p_values_process = scaled_plan[item,:]
        p_values_dual_temp = [Pair(first.(p_var)[1], p_values_process'[1])]
        for i in range(2, length(last.(p_var)))
            B = Pair(first.(p_var)[i], p_values_process'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end
        p_var_new = p_values_dual_temp

        # Check if the cost can be calculated
        MK = 0
        try
            MK = calc_cost(p_var_new,p_const)
        catch
            continue
        end

        N += 1
        # Saves parameters that works in a CSV file
        if N == 1
            df = DataFrame(Parameter = vcat(first.(p_var_new),Num("Cost")),
                        p_0_value_1 = vcat(last.(p_var_new),MK))
            CSV.write("Intermediate/p_0_values.csv",df)
        else
            df[!,Symbol("p_0_value_$N")] = vcat(last.(p_var_new),MK)
            CSV.write("Intermediate/p_0_values.csv",df)
        end

    end

    return N
end

"""
    main()

    Here you define different settings for the LHS, like number of samples,
    and then it generates a LHS and runs the Check_if_p_works(p_var, p_const,
    scaled_plan)
"""
```

```
function main()
    include("../Model/parameter_values.jl")

    plan = randomLHC(10000,81) #samples, dimensions

    #plan_scale = [(0.95*last.(p_var)[1],1.05*last.(p_var)[1])]
    plan_scale = [(-3.0,3.0)]
    for i in 1:80
        #push!(plan_scale,(0.95*last.(p_var)[i+1],1.05*last.(p_var)[i+1]))
        push!(plan_scale,(-3.0,3.0))
    end

    scaled_plan = scaleLHC(plan,plan_scale)

    N = Check_if_p_works(p_var, p_const, scaled_plan)
    println(" Number of working parameter sets: ", N)

end

main()
```

E.2.2 LHS kring den bästa parametervektorn i en lista

Code/Latin_Hypercube_Sampling/Auto_LHS.jl

```
using LatinHypercubeSampling
using LinearAlgebra
using DifferentialEquations
using DiffEqSensitivity
using ModelingToolkit
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")
include("../Parameter_Estimation/Optimization.jl")

"""
    Check_if_p_works(p_var, p_const, scaled_plan)

    Checks if the parameter sets from the latin hypercube works and saves them
    into a CSV file.
"""
function Check_if_p_works(p_var, p_const, scaled_plan)

    N = 0
    df = DataFrame()

    for item in 1:length(scaled_plan[:,1])

        print(" Iteration: ",item)

        #p_values_process = exp10.(scaled_plan[item,:])
        p_values_process = scaled_plan[item,:]
        p_values_dual_temp = [Pair(first.(p_var)[1], p_values_process'[1])]
        for i in range(2, length(last.(p_var)))
            B = Pair(first.(p_var)[i], p_values_process'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end

        p_var_new = p_values_dual_temp
        # Check if the cost can be calculated
        MK = 0
        try
            MK = calc_cost(p_var_new,p_const)
        catch
            continue
        end

        N += 1
        # Saves parameters that works in a CSV file
        if N == 1
            df = DataFrame(Parameter = vcat(first.(p_var_new),Num("Cost")),
                        p_0_value_1 = vcat(last.(p_var_new),MK))
            CSV.write("Intermediate/p_0_values.csv",df)
        else
            df[!,Symbol("p_0_value_$N")] = vcat(last.(p_var_new),MK)
            CSV.write("Intermediate/p_0_values.csv",df)
        end

    end
```

```
    return N
end
"""
    main()

    Finds the best parametervector in a list called p_0_values.csv and then
    generates latin hypercube samples around it (user defines interval),
    takes the best result and repeats until the user stops the iteration.
    If none of the costs were below the previously best one it will try to
    increase the number of samples by a choosen amount and generate a new
    latin hypercube.
"""
function main()
    include("../Model/parameter_values.jl")
    MK_current_best = 1e100
    MK_list = []
    p_0_var = p_var
    N = 0
    Number_of_samples = 200
    while true
        N += 1
        p_var = p_0_var
        if N > 1
            plan = randomLHC(Number_of_samples,81) #samples, dimensions

            plan_scale = [(0.97*last.(p_var)[1],1.03*last.(p_var)[1])]
            for i in 1:80
                push!(plan_scale,(0.97*last.(p_var)[i+1],1.03*last.(p_var)[i+1]))
            end

            scaled_plan = scaleLHC(plan,plan_scale)

            N_check = Check_if_p_works(p_var, p_const, scaled_plan)
            println(" Number of working parameter sets: ", N_check)
        end

        MK_df = 1e100
        Best_i = 0
        df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
        for i in 1:(length(df[1,:])-1)
            df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
            MK_df_new = df[:, Symbol("p_0_value_$i")][end]
            if MK_df_new < MK_df && MK_df_new < MK_current_best
                MK_df = MK_df_new
                Best_i = i
            end
        end
        println("Best_i:",Best_i)
        if Best_i == 0
            Number_of_samples = Number_of_samples + 50 # Adds 50 to the number
            # of samples if none of the parametervectors had lower cost then
            # the previous one
            continue
        end
        try
            df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
            p_names_var = first.(p_var)
            p_0_var = df[:, Symbol("p_0_value_$Best_i")][1:end-1]

            p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
            for item in range(2, length(p_0_var))
                B = Pair(p_names_var[item], p_0_var'[item])
                p_var_temp = vcat(p_var_temp,B)
            end
            p_0_var = p_var_temp
            MK_current_best = MK_df
            append!(MK_list,MK_current_best)
            println("MK:",MK_current_best)
        catch
            println("didn't work")
            Number_of_samples = Number_of_samples + 50
            if Number_of_samples > 1000
                break
            end
            continue
        end
        println("p_0_var:",p_0_var)

        if N == 1
            df = DataFrame(Parameter = vcat(first.(p_0_var),Num("Cost")),
                        p_0_value_1 = vcat(last.(p_0_var),MK_current_best))
```

```
            CSV.write("Intermediate/p_0_values_from_auto.csv",df)
        else
            df = CSV.read("Intermediate/p_0_values_from_auto.csv", DataFrame)
            df[!,Symbol("p_0_value_$N")] = vcat(last.(p_0_var),MK_current_best)
            CSV.write("Intermediate/p_0_values_from_auto.csv",df)
        end

    end
    println("All MK:",MK_list)

end

main()
```

E.2.3 Skriver ut en parametervektor från en lista

Code/Latin_Hypercube_Sampling/Get_one_value_from_p0_value_list.jl

```
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

"""
    Get_one_vector_from_p0_values_list(x)

    Gives the user the parametervector in a form suitable for ModelingToolkit
    for parametervector i in p_0_values.csv that in the Intermediate folder.
"""
function Get_one_vector_from_p0_values_list()
    df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
    i = 1 # The vector that will be displayed

    p_names_var = first.(p_var)

    p_0_var = df[:, Symbol("p_0_value_$i")][1:end-1]

    p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
    for item in range(2, length(p_0_var))
        B = Pair(p_names_var[item], p_0_var'[item])
        p_var_temp = vcat(p_var_temp,B)
    end
    p_0_var = p_var_temp
    println("p 0 var:",p_0_var)
    println("MK: ",df[:, Symbol("p_0_value_$i")][end])
end

Get_one_vector_from_p0_values_list()
```

E.2.4 LHS kring den bästa parametervektorn i en lista (version 2)

Code/Latin_Hypercube_Sampling/Grad_Auto_LHS.jl

```
using LatinHypercubeSampling
using LinearAlgebra
using DifferentialEquations
using DiffEqSensitivity
using ModelingToolkit
using CSV
using DataFrames

include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")
include("../Parameter_Estimation/Optimization.jl")

"""
    Check_if_p_works(p_var, p_const, scaled_plan)

    Checks if the parameter sets from the latin hypercube works and saves them
    into a CSV file.
"""
function Check_if_p_works(p_var, p_const, scaled_plan)

    N = 0
    df = DataFrame()

    for item in 1:length(scaled_plan[:,1])

        print(" Iteration: ",item)
```

```
        #p_values_process = exp10.(scaled_plan[item,:])
        p_values_process = scaled_plan[item,:]
        p_values_dual_temp = [Pair(first.(p_var)[1], p_values_process'[1])]
        for i in range(2, length(last.(p_var)))
            B = Pair(first.(p_var)[i], p_values_process'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end

        p_var_new = p_values_dual_temp
        # Check if the cost can be calculated
        MK = 0
        try
            MK = calc_cost(p_var_new,p_const)
        catch
            continue
        end

        N += 1
        # Saves parameters that works in a CSV file
        if N == 1
            df = DataFrame(Parameter = vcat(first.(p_var_new),Num("Cost")),
                        p_0_value_1 = vcat(last.(p_var_new),MK))
            CSV.write("Intermediate/p_0_values.csv",df)
        else
            df[!,Symbol("p_0_value_$N")] = vcat(last.(p_var_new),MK)
            CSV.write("Intermediate/p_0_values.csv",df)
        end

    end

    return N
end

"""
    main()

    First one must define a list called p_0_values_list.csv with
    prametervectors that the sampling will start around. This list can for
    example be created by first generating a more general latin hypercube with
    latin_hypercube_sampling.jl and one can then filter the results to get a
    list with parametervectors that gave a cost below a choosen value. Then
    the code (main()) will create a latin hypercube based on the
    Number_of_samples and what intervals the user defines. If it finds a
    parametervector with a cost below the previous best one it will start
    generating a new latin hypercube around the new value. If none of the
    costs were below the previously best one it will try to increase the
    number of samples by a choosen amount and generate a new latin hypercube.
    If the cost is less then the previous best but not good enough, specified
    by how much the user wants the cost to decrease, it will move on to the
    next parametervector in p_0_values_list.csv. The code will also save all
    intermediate results in csv files.

    Note that one must do some modifications in
    Check_if_p_works(p_var, p_const, scaled_plan) if the user wants to give
    the intervals on logarithmic-scale.
"""
function main()
    include("../Model/parameter_values.jl")
    N_3 = 0
    while true
        N_3 += 1
        MK_current_best = 1e100
        MK_list = []
        p_0_var = p_var
        N = 0
        Number_of_samples = 300
        N_2 = 1
        MK_prev = 0
        while true
            N += 1
            p_var = p_0_var
            if N > 1
                plan = randomLHC(Number_of_samples,81) #samples, dimensions

                plan_scale = [(0.8*last.(p_var)[1],1.2*last.(p_var)[1])]
                for i in 1:80
                    push!(plan_scale,(0.8*last.(p_var)[i+1],
                    1.2*last.(p_var)[i+1]))
                end

                scaled_plan = scaleLHC(plan,plan_scale)
```

```
    N_check = Check_if_p_works(p_var, p_const, scaled_plan)
    println(" Number of working parameter sets: ", N_check)
end

if N > 1
    MK_df = 1e100
    Best_i = 0
    df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
    for i in 1:(length(df[1,:])-1)
        df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
        MK_df_new = df[:, Symbol("p_0_value_$i")][end]
        if MK_df_new < MK_df && MK_df_new < MK_current_best
            MK_df = MK_df_new
            Best_i = i
        end
    end
    println("Best_i:",Best_i)
    if Best_i == 0
        Number_of_samples = Number_of_samples + 200 # Adds this
        # amount to number of samples if no cost under the
        # previous best was found
        continue
    end
    try
        df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
        p_names_var = first.(p_var)
        p_0_var = df[:, Symbol("p_0_value_$Best_i")][1:end-1]

        p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
        for item in range(2, length(p_0_var))
            B = Pair(p_names_var[item], p_0_var'[item])
            p_var_temp = vcat(p_var_temp,B)
        end
        p_0_var = p_var_temp
        MK_current_best = MK_df
        append!(MK_list,MK_current_best)
        println("MK:",MK_current_best)
    catch
        println("didn't work")
        continue
    end
    println("p_0_var:",p_0_var)
else
    Best_i = N_3

    df = CSV.read("Intermediate/p_0_values_list.csv", DataFrame)
    p_names_var = first.(p_var)
    p_0_var = df[:, Symbol("p_0_value_$Best_i")][1:end-1]
    MK_df = df[:, Symbol("p_0_value_$Best_i")][end]

    p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
    for item in range(2, length(p_0_var))
        B = Pair(p_names_var[item], p_0_var'[item])
        p_var_temp = vcat(p_var_temp,B)
    end
    p_0_var = p_var_temp
    MK_current_best = MK_df
    append!(MK_list,MK_current_best)
    println("MK:",MK_current_best)

    println("p_0_var:",p_0_var)
end

if N == 1
    df = DataFrame(Parameter = vcat(first.(p_0_var),Num("Cost")),
                   p_0_value_1 = vcat(last.(p_0_var),MK_current_best))
    CSV.write("Intermediate/p_0_values_from_auto_$N_3.csv",df)
else
    N_2 += 1
    df = CSV.read("Intermediate/p_0_values_from_auto_$N_3.csv"
    , DataFrame)
    df[!,Symbol("p_0_value_$N_2")] = vcat(last.(p_0_var),
    MK_current_best)
    CSV.write("Intermediate/p_0_values_from_auto_$N_3.csv",df)
end

if N <= 1
    MK_prev = 1e10 # Just to make sure grad_of_cost isn't less
    # then 0.05 for the first iteration
end

grad_of_cost = abs(MK_df - MK_prev)
```

```
            if grad_of_cost < 0.05 # The value of which the cost must
                # decrease from each iteration
                break
            end

            if N > 1
                MK_prev = MK_df # Used in next iteration
            end

        end
    end
    println("All MK:",MK_list)

end

main()
```

E.2.5 Figur som jämför LHS och slumpmässigt urval

Code/Latin_Hypercube_Sampling/Image_for_report_for_LHS.jl

```
using LatinHypercubeSampling
using Plots

# Creates a figure that illustrates the difference between LHS and random
# sampling

plan = randomLHC(20,2)
scaled_plan = scaleLHC(plan,[(0.0,1.0),(0.0,1.0)]) # Defines the interval
# for which values the 2 dimensions can take

gr()
plotd = plot(scaled_plan[:,1], scaled_plan[:,2], seriestype = :scatter,
title = "Latin Hypercube Sampling", legend = false)

plotd2 = plot(rand(20), rand(20), seriestype = :scatter,
title = "Slumpm ssigt urval", legend = false, color = "orange")
l = @layout [a b]

plot_result = plot(plotd, plotd2,layout = l, gridalpha=0.3,xticks=0.0:0.2:1.0,
yticks=0.0:0.2:1.0,tickfontcolor = "black", minorgrid = true, minorticks = 4,
minorgridalpha = 0.3, aspect_ratio=:equal, markersize = 3)
xlabel!("Dimension 1")
ylabel!("Dimension 2")
xlims!((-0.1, 1.1))
ylims!((-0.1, 1.1))

display(plot_result)

Want_to_plot = false
if Want_to_plot == true
    savefig(plot_result, pwd()*"/Results/LHS/LHS_png.png")
    savefig(plot_result, pwd()*"/Results/LHS/LHS_svg.svg")
end
```

E.2.6 Filtrerar lista med parametervektorer baserat på kostnaden

Code/Latin_Hypercube_Sampling/filter_p0_values_list.jl

```
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

"""
    filter_p0_values_list(x)

    Creates a list called p_0_values_MK_under_x.csv which contains all
    parametervectors from p_0_values.csv with a cost below x.
"""
function filter_p0_values_list(x)
    N = 0
    df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
    for i in 1:(length(df[1,:])-1)
        df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
        MK_df = df[:, Symbol("p_0_value_$i")][end]
        MK_that_works = df[:, Symbol("p_0_value_$i")][1:end]
        if MK_df > x
            continue
```

```
        else
            N += 1
            if N == 1
                df = DataFrame(Parameter = vcat(first.(p_var),Num("Cost")),
                               p_0_value_1 = MK_that_works)
                CSV.write("Intermediate/p_0_values_MK_under_x.csv",df)
            else
                df = CSV.read("Intermediate/p_0_values_MK_under_x.csv",
                DataFrame)
                df[!,Symbol("p_0_value_$N")] = MK_that_works
                CSV.write("Intermediate/p_0_values_MK_under_x.csv",df)
            end
        end
    end
    return N
end

x = 1.6 # Define highest allowed cost
filter_p0_values_list(x)
```

E.2.7 Skriver ut en parametervektor från en lista med optimerade vektorer

Code/Latin_Hypercube_Sampling/filter_p0_values_opt_list.jl

```
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

"""
    filter_p0_values_opt_list()

    Gives the user the parametervector in a form suitable for ModelingToolkit
    for parametervector i in optimization_of_p0.csv.
"""
function filter_p0_values_opt_list()
    df = CSV.read("Intermediate/optimization_of_p0.csv", DataFrame)

    i = 11 #Choose manually

    p_names_var = first.(p_var)

    p_0_var = df[:, Symbol("Optimum_from_p_0_value_$i")][1:end-1]

    p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
    for item in range(2, length(p_0_var))
        B = Pair(p_names_var[item], p_0_var'[item])
        p_var_temp = vcat(p_var_temp,B)
    end
    p_0_var = p_var_temp
    println("p 0 var:",p_0_var)
    println("MK: ",df[:, Symbol("Optimum_from_p_0_value_$i")][end])
end

filter_p0_values_opt_list()
```

E.2.8 Skriver ut en parametervektor från en filtrerad lista

Code/Latin_Hypercube_Sampling/filter_p0_values_under_MK_list.jl

```
using CSV
using DataFrames
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

"""
    filter_p0_values_under_MK_list()

    Gives the user the parametervector in a form suitable for ModelingToolkit
    for parametervector i in p_0_values_MK_under_x.csv.
"""
function filter_p0_values_under_MK_list()
    df = CSV.read("Intermediate/p_0_values_MK_under_x.csv", DataFrame)

    i = 2 #Choose manually which vector to display
    df = CSV.read("Intermediate/p_0_values_MK_under_x.csv", DataFrame)

    MK_that_works = df[:, Symbol("p_0_value_$i")][1:end]
```

```
    p_names_var = first.(p_var)

    p_0_var = df[:, Symbol("p_0_value_$i")][1:end-1]

    p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
    for item in range(2, length(p_0_var))
        B = Pair(p_names_var[item], p_0_var'[item])
        p_var_temp = vcat(p_var_temp,B)
    end
    p_0_var = p_var_temp
    println("p 0 var:",p_0_var)
    println("MK: ",df[:, Symbol("p_0_value_$i")][end])
end

filter_p0_values_under_MK_list()
```

E.3 Skapa figur över resultaten från optimeringen

E.3.1 Figur över resultaten från optimeringen

Code/Make_image_of_optimization_results/All_plots_in_a_subplot.jl

```
# Makes a plot for visualisation of results from optimization

include("plot_for_all_ODE_solutions_with_data.jl")
include("Histogram_plots_of_cost.jl")
include("../../Results/Parameter_values/Results_from_optimization.jl")

plot1 = plot_for_all_ODE_solutions_with_data()
plot2 = Histograms_of_cost()

leg_test = true
leg = false

if leg == false && leg_test == false
    plot4 = plot(plot1, plot2, layout = (1, 2), titlefontsize = 14,
    guidefontsize = 10, guide_position = :left, margin= 20Plots.mm,
    markersize = 5.5, markercolor="blue", markerstrokewidth=0.2)
    plot!(plot4,size=(3000,2500))
    display(plot4)
    savefig(plot4, pwd()*"/Results/Plots_for_new_and_old_paramter_values/
    plot_and_histogram_for_p_sets.png")
elseif leg == true
    p3 = plot([0 0 0], showaxis = false, grid = false, legend = false)
    p2 = plot([0 0 0], showaxis = false, grid = false, legend = false)
    p1 = plot([0 0 0], color = [Color_alt_1 Color_alt_2 Color_jalihal],
    showaxis = false, grid = false,
    labels = [label_alt_1 label_alt_2 label_jalihal],labelsize=100,
    legendfontsize=50)
    p4 = plot([0 0 0], showaxis = false, grid = false, legend = false)
    p5 = plot([0 0 0], showaxis = false, grid = false, legend = false)

    plot3 = plot(p1, p2, p3, p4, p5, layout = (5, 1), titlefontsize = 14,
    guidefontsize = 10, guide_position = :left, margin= 20Plots.mm,
    foreground_color_text = "white")
    plot!(plot3,size=(1500,2500))
    plot4 = plot(plot3, plot1, plot2, layout = (1, 3), titlefontsize = 14,
    guidefontsize = 10, guide_position = :left, margin= 20Plots.mm,
    markersize = 5.5, markercolor="blue", markerstrokewidth=0.2)
    plot!(plot4,size=(3750,2500))
    display(plot4)
    savefig(plot4, pwd()*"/Results/Plots_for_new_and_old_paramter_values/
    plot_and_histogram_for_p_sets_with_legend.png")
elseif leg_test == true
    p3 = plot([0 0 0], showaxis = false, grid = false, legend = false)
    p2 = plot([0 0 0], showaxis = false, grid = false, legend = false)
    p4 = plot([0 0 0], color = [Color_alt_1 Color_alt_2 Color_jalihal],
    showaxis = false, grid = false,
    labels = [label_alt_1 label_alt_2 label_jalihal],labelsize=100,
    legendfontsize=50, legend = :bottomright)
    p1 = plot([0 0 0], showaxis = false, grid = false, legend = false)

    plot3 = plot(p1, p2, p3, p4, layout = (1,4), titlefontsize = 14,
    guidefontsize = 10, guide_position = :left, margin= 20Plots.mm,
    foreground_color_text = "white")
    plot!(plot3,size=(3000,500))
    plot4 = plot(plot1, plot2, layout = (1, 2), titlefontsize = 14,
    guidefontsize = 10, guide_position = :left, margin= 20Plots.mm,
    markersize = 5.5, markercolor="blue", markerstrokewidth=0.2)
    plot!(plot4,size=(3000,2700))
```

```
    plot5 = plot(plot3,plot4, layout = grid(2, 1, heights = [0.05, 0.95]),
    titlefontsize = 14, guidefontsize = 10, guide_position = :left,
    margin= 20Plots.mm, markersize = 5.5, markercolor="blue",
    markerstrokewidth=0.2)
    plot!(plot5,size=(3000,3500))
    display(plot5)
    Want_to_plot = false
    if Want_to_plot == true
        savefig(plot5,
        pwd()*"/Results/Plots_for_new_and_old_paramter_values/Result_plot.png")
    end
end
```

E.3.2 Beräknar kostnader för histogrammen

Code/Make_image_of_optimization_results/Cost_for_histograms.jl

```
"""
    Cost_for_histograms()

    Calculates the cost for each set of data for each parametervector from
    optimization and Jalihal m.fl.
"""
function Cost_for_histograms()
    include("../Parameter_Estimation/Optimization.jl")
    include("../Model/ODE_functions.jl")
    include("../Model/parameter_values.jl")
    include("../../Data/exp_data.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    """
        calc_cost_for_p(p_var_FWD, p_const_FWD)

        Caculates the cost based on p_var_FWD and p_const_FWD.
    """
    function calc_cost_for_p(p_var_FWD, p_const_FWD)

        function Calc_cost_Glucose_addition_Mig1(p_var_FWD, p_const_FWD, u0_SS)

            data_for_ode = raw_data_Mig1_glucose_relief
            timelist_for_ode = t_Mig1_glucose_relief
            tspan = (0.0, 20.0) # [min]

            sol = ODE_solver_FWD(u0_SS, (ATP => 1.0, Carbon => 1.0,
            Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
            timelist_for_ode)

            u_approx = getindex.(sol.u, 20) #ODE for MIG1 on row 20 in ODE
            # system

            #Check if sol was a success
            u_approx = Check_if_cost_is_a_success(sol, u_approx)

            MK_Glucose_addition_Mig1 = sum((data_for_ode -
            (log.(10,u_approx./(1e-5 .+ 1.0 .- u_approx)))).^2)

            return MK_Glucose_addition_Mig1
        end

        function Calc_cost_Glucose_addition_Sch9(p_var_FWD, p_const_FWD)

            data_for_ode = Minmaxnorm(raw_data_sch9Delta_cAMP, 0.052, 1.227)
            timelist_for_ode = t_sch9Delta_cAMP
            tspan = (0.0, 3.5) # [min]
            # Mutant Sch9_Delta => Sch9_T = 0
            p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
            Sch9_T => 0.0

            # Pre-shift => ATP, Carbon = 0
            u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
            (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar
            # steady state f r parametrarna p

            # Post-shift => ATP, Carbon = 1
            sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
            Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
            timelist_for_ode)

            u_approx = getindex.(sol.u, 6) #ODE for cAMP on row 6 in ODE
            # system

            #Check if sol was a success
```

```
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_Sch9 = sum((data_for_ode - u_approx).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
    Sch9_T => 1.0

    return MK_Glucose_addition_Sch9
end

function Calc_cost_Glucose_addition_cAMP(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_cAMP, 0.052, 1.227)
    timelist_for_ode = t_cAMP
    tspan = (0.0, 3.5) # [min]

    # Post-shift, Glucose addition => ATP, Carbon = 1
    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 6) #ODE for cAMP on row 6 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_cAMP = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_addition_cAMP
end

function Calc_cost_Glucose_addition_Sch9_p(p_var_FWD, p_const_FWD,
    u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_relief)
    timelist_for_ode = t_Sch9_glucose_relief
    tspan = (0.0, 30) # [min]

    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_Sch9_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_addition_Sch9_p
end

function Calc_cost_Glucose_starvation_Snf1_p(p_var_FWD, p_const_FWD,
    u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Snf1)
    timelist_for_ode = t_Snf1
    tspan = (0.0, 62) # [min]

    # Post-shift, Glucose starvation => Carbon, ATP = 0

    sol = ODE_solver_FWD(u0_SS, (Carbon => 0.0, ATP => 0.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)
    u_approx = getindex.(sol.u, 10) #ODE for Snf1 on row 10 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_starvation_Snf1_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_starvation_Snf1_p
end

function Calc_cost_Glucose_starvation_Sch9_p(p_var_FWD, p_const_FWD,
    u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_starve)
    timelist_for_ode = t_Sch9_glucose_starve
    tspan = (0.0, 30) # [min]
```

```
    sol = ODE_solver_FWD(u0_SS, (Carbon => 0.0, ATP => 0.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_starvation_Sch9_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_starvation_Sch9_p
end

function Calc_cost_Sch9P_glutamine_L(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_L, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_L
    tspan = (0.0, 32) # [min]

    # Low glutamine => Glutamine_ext = 0.3
    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 0.3), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Sch9P_glutamine_L = sum((data_for_ode - u_approx).^2)

    return MK_Sch9P_glutamine_L
end

function Calc_cost_Sch9P_glutamine_H(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_H, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_H
    tspan = (0.0, 32) # [min]

    # High Glutamine => Glutamine_ext = 1.0
    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Sch9P_glutamine_H = sum((data_for_ode - u_approx).^2)

    return MK_Sch9P_glutamine_H
end

function Calc_cost_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD,
    p_const_FWD)

    data_for_ode = Minmaxnorm(raw_data_Sch9_gtr1Delta, 4.82, 52.27)
    timelist_for_ode = t_Sch9_gtr1Delta
    tspan = (0.0, 32) # [min]

    w_torc_ego_true = p_var_FWD[Get_index(p_var_lookup_table,
    "w_torc_ego")]
    w_torc_egoin_true = p_var_FWD[Get_index(p_var_lookup_table,
    "w_torc_egoin")]

    # Mutant gtr1_Delta => EGO_T = 0, w_torc_ego = 0,
    # w_torc_egoin = 0 i
    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] =
    EGO_T => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0

    # Pre-shift => Glutamine_ext = 0, ATP, Carbon = 1
    u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))
```

```
    # High glutamine => Glutamine_ext = 1
    sol = ODE_solver_FWD(u0_SS, (Glutamine_ext => 1.0,
    Carbon => 1.0, ATP => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glutamine_addition_Sch9_gtr1Delta = sum((data_for_ode -
    u_approx).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] =
    EGO_T => 1.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego_true
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin_true

    return MK_Glutamine_addition_Sch9_gtr1Delta
end

function Calc_cost_Rapamycin_treatment(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Rib_rap)
    timelist_for_ode = t_Rib_rap
    tspan = (0.0, 92.0) # [min]

    # Post-shift: Rapamycin treatment => TORC1_T = 0.0
    p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
    TORC1_T => 0.0

    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 25) #ODE for RIB on row 25 in ODE
    # system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Rapamycin_treatment = sum((data_for_ode - (u_approx./(1e-3 +
    u0_SS[25]))).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
    TORC1_T => 1.0

    return MK_Rapamycin_treatment
end

u0_SS_A_C_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
MK_Glucose_starvation_Snf1_p =
Calc_cost_Glucose_starvation_Snf1_p(p_var_FWD, p_const_FWD,
u0_SS_A_C_G)
MK_Glucose_starvation_Sch9_p =
Calc_cost_Glucose_starvation_Sch9_p(p_var_FWD, p_const_FWD,
u0_SS_A_C_G)
MK_Rapamycin_treatment =
Calc_cost_Rapamycin_treatment(p_var_FWD, p_const_FWD, u0_SS_A_C_G)

u0_SS_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD, (ATP => 0.0,
Carbon => 0.0, Glutamine_ext => 1.0))
MK_Glucose_addition_Mig1 = Calc_cost_Glucose_addition_Mig1(p_var_FWD,
p_const_FWD, u0_SS_G)
MK_Glucose_addition_cAMP = Calc_cost_Glucose_addition_cAMP(p_var_FWD,
p_const_FWD, u0_SS_G)
MK_Glucose_addition_Sch9_p =
Calc_cost_Glucose_addition_Sch9_p(p_var_FWD, p_const_FWD, u0_SS_G)

u0_SS_A_C = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0))
MK_Sch9P_glutamine_L = Calc_cost_Sch9P_glutamine_L(p_var_FWD,
p_const_FWD, u0_SS_A_C)
MK_Sch9P_glutamine_H = Calc_cost_Sch9P_glutamine_H(p_var_FWD,
p_const_FWD, u0_SS_A_C)

MK_Glucose_addition_Sch9 = Calc_cost_Glucose_addition_Sch9(p_var_FWD,
p_const_FWD)
```

```
        MK_Glutamine_addition_Sch9_gtr1Delta =
        Calc_cost_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD, p_const_FWD)

        MK = [MK_Glucose_addition_Mig1, MK_Glucose_addition_Sch9,
        MK_Glucose_addition_cAMP, MK_Glucose_addition_Sch9_p,
        MK_Glucose_starvation_Snf1_p, MK_Glucose_starvation_Sch9_p,
        MK_Sch9P_glutamine_L, MK_Sch9P_glutamine_H,
        MK_Glutamine_addition_Sch9_gtr1Delta, MK_Rapamycin_treatment]

        return MK
    end

    """
        Check_if_cost_is_a_success(sol, u_approx)

        Checks if sol.retcode != :Success
    """
    function Check_if_cost_is_a_success(sol, u_approx)
        if sol.retcode != :Success
            println(" Can't Calc Cost ")
            u_approx = "Not working"
        end
        return u_approx
    end

    Cost_alt_1 = calc_cost_for_p(p_var_alt_1, p_const)
    Cost_alt_2 = calc_cost_for_p(p_var_alt_2, p_const)
    Cost_jalihal = calc_cost_for_p(p_var_jalihal, p_const)
    return Cost_alt_1, Cost_alt_2, Cost_jalihal
end
```

E.3.3 Glukostillsättning

Code/Make_image_of_optimization_results/Glucose_addition.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots
function Glucose_addition_cAMP()
    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")
    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift, Glucose addition => ATP, Carbon = 1
    tspan_cAMP = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_cAMP, p_const, p_var_alt_1)

    plot1 = scatter(t_cAMP, data_cAMP, markersize = 5.5, color = color_data,
    markerstrokewidth=0.8, labels= "Data")

    plot!(plot1, sol, vars = cAMP, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)

    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift, Glucose addition => ATP, Carbon = 1
    tspan_cAMP = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_cAMP, p_const, p_var_alt_2)

    plot!(plot1, sol, vars=cAMP, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift, Glucose addition => ATP, Carbon = 1
```

```
    tspan_cAMP = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_cAMP, p_const, p_var_jalihal)

    plot!(plot1, sol, vars=cAMP, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("cAMP")
    ylims!((0.0, 1.02))
    xlims!((-0.02, last(tspan_cAMP)*1.02))
    title!("Glukostillsättning cAMP")
    return plot1
end

function Glucose_addition_Sch9()
    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")

    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_Sch9, p_const, p_var_alt_1)

    plot2 = scatter(t_Sch9_glucose_relief, data_Sch9_glucose_relief,
    markersize = 5.5, color = color_data, markerstrokewidth=0.8,
    label = "Data")

    plot!(plot2, sol, vars = Sch9, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)


    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_Sch9, p_const, p_var_alt_2)

    plot!(plot2, sol, vars=Sch9, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)


    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan_Sch9, p_const, p_var_jalihal)

    plot!(plot2, sol, vars=Sch9, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("Sch9")
    ylims!((0.0, 1.02))
    xlims!((-0.3, last(tspan_Sch9)*1.02))
    title!("Glukostillsättning Sch9 p")
    return plot2
end
```

E.3.4 Glukostillsättning Mig1

Code/Make_image_of_optimization_results/Glucose_addition_Mig1.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Glucose_addition_Mig1()
    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    # Pre-shift => ATP, Carbon = 0
```

```
    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")

    plot1 = scatter(t_Mig1_glucose_relief, data_Mig1_glucose_relief,
    markersize = 5.5, color = color_data, markerstrokewidth=0.8,
    labels= "Data")

    u0_SS = Steady_state_solver(p_const, p_var_alt_1, (ATP => 0.0,
    Carbon => 0.0, Glutamine_ext => 1.0)) # Returnerar steady state  f  r
    # parametrarna p

    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 20.0) # [min]
    sol = ODE_solver(u0_SS, (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_1)

    plot!(plot1, sol, vars=log(10, Mig1/(1e-5 + 1.0-Mig1)),
    color = Color_alt_1, lw=2.5, labels= label_alt_1)

    u0_SS = Steady_state_solver(p_const, p_var_alt_2, (ATP => 0.0,
    Carbon => 0.0, Glutamine_ext => 1.0)) # Returnerar steady state  f  r
    # parametrarna p

    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 20.0) # [min]
    sol = ODE_solver(u0_SS, (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_2)

    plot!(plot1, sol, vars=log(10, Mig1/(1e-5 + 1.0-Mig1)),
    color = Color_alt_2, lw=2.5, labels= label_alt_2)


    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (ATP => 0.0, Carbon => 0.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 20.0) # [min]
    sol = ODE_solver(u0_SS, (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_jalihal)

    plot!(plot1,sol, vars=log(10, Mig1/(1e-5 + 1.0-Mig1)),
    color = Color_jalihal, lw=2.5, labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("log(Mig1/(Mig1_T-Mig1)")
    ylims!((0.9, 1.6))
    title!("Glukostills ttning Mig1")

    return plot1
end
```

E.3.5 Glukostillsättning Sch9

Code/Make_image_of_optimization_results/Glucose_addition_Sch9.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Glucose_addition_Sch9_cAMP()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")
    # Mutant Sch9_Delta => Sch9_T = 0
    p_const[Get_index(p_const_lookup_table, "Sch9_T")] = Sch9_T => 0.0

    # Pre-shift => ATP, Carbon = 0
    u0_SS = Steady_state_solver(p_const, p_var_alt_1, (Carbon => 0.0,
    ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady state  f  r
    # parametrarna p
```

```
    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_1)

    plot1 = scatter(t_sch9Delta_cAMP, data_sch9Delta_cAMP, markersize = 5.5,
    color = color_data, markerstrokewidth=0.8, labels= "Data")

    plot!(plot1, sol, vars=cAMP, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)


    u0_SS = Steady_state_solver(p_const, p_var_alt_2, (Carbon => 0.0,
    ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady state  f  r
    # parametrarna p

    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_2)

    plot!(plot1, sol, vars=cAMP, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    u0_SS = Steady_state_solver(p_const, p_var_jalihal, (Carbon => 0.0,
    ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar steady state  f  r
    # parametrarna p

    # Post-shift => ATP, Carbon = 1
    tspan = (0.0, 3.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_jalihal)

    plot!(plot1, sol, vars=cAMP, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("cAMP")
    ylims!((0, 1.1))
    xlims!((-0.03, last(tspan)*1.01))
    title!("Glukostillsttning Sch9\\Delta")
    return plot1
end
```

E.3.6 Glukossvältning

Code/Make_image_of_optimization_results/Glucose_starvation.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Glucose_starvation_Snf1()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")
    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift, Glucose starvation => Carbon, ATP = 0
    tspan_Snf1 = (0.0, 61.0) # [min]
    sol_Snf1 = ODE_solver(u0_SS, (Carbon => 0.0, ATP => 0.0,
    Glutamine_ext => 1.0), tspan_Snf1, p_const, p_var_alt_1)

    plot1 = scatter(t_Snf1, data_Snf1, markersize = 5.5,
    color = color_data, markerstrokewidth=0.8, label = "Data")

    plot!(plot1, sol_Snf1, vars=Snf1, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)


    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
```

```
    # state  f  r  parametrarna p

    # Post-shift, Glucose starvation => Carbon, ATP = 0
    tspan_Snf1 = (0.0, 61.0) # [min]
    sol_Snf1 = ODE_solver(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Snf1, p_const,
    p_var_alt_2)

    plot!(plot1, sol_Snf1, vars=Snf1, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state  f  r  parametrarna p

    # Post-shift, Glucose starvation => Carbon, ATP = 0
    tspan_Snf1 = (0.0, 61.0) # [min]
    sol_Snf1 = ODE_solver(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Snf1, p_const,
    p_var_jalihal)

    plot!(plot1, sol_Snf1, vars=Snf1, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("Snf1")
    ylims!((0.0, 1.1))
    xlims!((-0.5, last(tspan_Snf1)*1.02))
    title!("Glukossvältning Snf1")
    return plot1
end

function Glucose_starvation_Sch9()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")

    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol_sch9 = ODE_solver(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Sch9, p_const,
    p_var_alt_1)

    plot2 = scatter(t_Sch9_glucose_starve, data_Sch9_glucose_starve,
    markersize = 5.5, color = color_data, markerstrokewidth=0.8,
    label = "Data")

    plot!(plot2, sol_sch9, vars=Sch9, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)


    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol_sch9 = ODE_solver(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Sch9, p_const,
    p_var_alt_2)

    plot!(plot2, sol_sch9, vars=Sch9, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
    tspan_Sch9 = (0.0, 30.0) # [min]
    sol_sch9 = ODE_solver(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Sch9, p_const,
    p_var_jalihal)

    plot!(plot2, sol_sch9, vars=Sch9, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("Sch9")
    ylims!((0.0, 1.1))
    xlims!((-0.5, last(tspan_Sch9)*1.02))
    title!("Glukossvältning Sch9")
    return plot2
```

```
end
```

E.3.7 Kvävetillsättning

Code/Make_image_of_optimization_results/Glutamine_addition.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Glutamine_addition_L()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")
    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady
    # state f r parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_low = ODE_solver(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan, p_const,
    p_var_alt_1)

    plot1 = scatter(t_Sch9P_glutamine_L, data_Sch9P_glutamine_L,
    markersize = 5.5, color = color_data, markerstrokewidth=0.8,
    labels = "Data")

    plot!(plot1, sol_low, vars = Sch9, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)


    u0_SS = Steady_state_solver(p_const, p_var_alt_2, (Carbon => 1.0,
    ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady state f r
    # parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_low = ODE_solver(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan, p_const,
    p_var_alt_2)

    plot!(plot1, sol_low, vars=Sch9, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)


    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady
    # state f r parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_low = ODE_solver(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan, p_const,
    p_var_jalihal)

    plot!(plot1, sol_low, vars=Sch9, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("Sch9")
    ylims!((0, 1.1))
    xlims!((-0.5, last(tspan)*1.02))
    title!("Kv vetills ttning (L g , 0.3)")
    return plot1
end

function Glutamine_addition_H()
    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
```

```
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")

    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady
    # state f r parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_high = ODE_solver(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const,
    p_var_alt_1)

    plot2 = scatter(t_Sch9P_glutamine_H, data_Sch9P_glutamine_H,
    markersize = 5.5, color = color_data, markerstrokewidth=0.8,
    labels = "Data")

    plot!(plot2, sol_high, vars = Sch9, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)

    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady
    # state f r parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_high = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const, p_var_alt_2)

    plot!(plot2, sol_high, vars=Sch9, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)) # Returnerar steady
    # state f r parametrarna p

    # Low glutamine => Glutamine_ext = 0.3
    tspan = (0.0, 30.0) # [min]
    sol_high = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const, p_var_jalihal)

    plot!(plot2, sol_high, vars=Sch9, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)

    xlabel!("t [min]")
    xlims!((-0.5, last(tspan)*1.02))
    ylabel!("Sch9")
    title!("Kv vetills ttning (H g , 1.0)")
    return plot2
end
```

E.3.8 Kvävetillsättning gtr1

Code/Make_image_of_optimization_results/Glutamine_addition_gtr1.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Glutamine_addition_gtr1()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")

    # Mutant gtr1_Delta => EGO_T = 0, w_torc_ego = 0, w_torc_egoin = 0 i
    p_const[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 0.0
    p_var_alt_1[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_alt_1[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0

    # Pre-shift => Glutamine_ext = 0, ATP, Carbon = 1

    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
```

```
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

    # High glutamine => Glutamine_ext = 1
    tspan = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS,
    (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0), tspan, p_const,
    p_var_alt_1)

    plot1 = scatter(t_Sch9_gtr1Delta, data_Sch9_gtr1Delta, markersize = 5.5,
    color = color_data, markerstrokewidth=0.8, label = "Data")

    plot!(plot1, sol, vars=Sch9, color = Color_alt_1, lw=2.5,
    labels= label_alt_1)

    p_var_alt_2[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_alt_2[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0
    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

    # High glutamine => Glutamine_ext = 1
    tspan = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS, (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0),
    tspan, p_const, p_var_alt_2)

    plot!(plot1, sol, vars=Sch9, color = Color_alt_2, lw=2.5,
    labels= label_alt_2)


    p_var_jalihal[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_jalihal[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0
    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

    # High glutamine => Glutamine_ext = 1
    tspan = (0.0, 30.0) # [min]
    sol = ODE_solver(u0_SS, (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0),
    tspan, p_const, p_var_jalihal)

    plot!(plot1, sol, vars=Sch9, color = Color_jalihal, lw=2.5,
    labels= label_jalihal)


    xlabel!("t [min]")
    ylabel!("Sch9")
    ylims!((0.0, 1.1))
    xlims!((-0.5, last(tspan)*1.02))
    title!("Kv vetills ttning gtr1\\Delta")
    return plot1
end
```

E.3.9 Histogram över kostnaderna

Code/Make_image_of_optimization_results/Histogram_plots_of_cost.jl

```
using StatsPlots
"""
  Histograms_of_cost()

  Makes a subplot for all histograms displaying the cost for all ODE solutions
  with data.
"""
function Histograms_of_cost()
  include("../../Results/Parameter_values/Results_from_optimization.jl")
  #include("Cost_for_histograms.jl")
  #Cost_alt_1, Cost_alt_2, Cost_jalihal = Cost_for_histograms()
  #println(Cost_alt_1)
  #println(Cost_alt_2)
  #println(Cost_jalihal)
  Cost_alt_1 = [0.02771829851138364, 0.11530809012155227, 0.14658811799803514,
  0.038428681050842434, 0.02894780772394849, 0.1127683119230797,
  0.06509447763039686, 0.10031530561560253, 0.08076111254137344,
  0.0036672576122928754]
  Cost_alt_2= [0.05048587297214781, 0.10025934231377061, 0.14227750026675529,
  0.027846845642827887, 0.06587255617950803, 0.0377115142788618,
  0.1709282163040912, 0.1258409006202471, 0.029440758752914818,
  0.011341041598241662]
  Cost_jalihal = [0.04352060436354394, 0.16782352005896917,
  0.5803849015840222, 0.0813183538847046, 0.02664445093902895,
```

```
0.015210225897085 37, 0.6459549142857574, 0.6638851878171327,
0.054569323749060414, 0.01492059687602178]

His_Glucose_addition_Mig1 = His1 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[1], Cost_alt_2[1], Cost_jalihal[1]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukostillsttning Mig1",
  legend = false
)

His_Glucose_addition_Sch9 = His2 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[2], Cost_alt_2[2], Cost_jalihal[2]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukostillsttning Sch9\\Delta",
  legend = false
)

His_Glucose_addition_cAMP = His3 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[3], Cost_alt_2[3], Cost_jalihal[3]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukostillsttning cAMP",
  legend = false
)

His_Glucose_addition_Sch9_p = His4 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[4], Cost_alt_2[4], Cost_jalihal[4]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukostillsttning Sch9 p",
  legend = false
)

His_Glucose_starvation_Snf1_p = His5 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[5], Cost_alt_2[5], Cost_jalihal[5]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukossvltning Snf1",
  legend = false
)

His_Glucose_starvation_Sch9_p = His6 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[6], Cost_alt_2[6], Cost_jalihal[6]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Glukossvltning Sch9",
  legend = false
)

His_Sch9P_glutamine_L = His8 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[7], Cost_alt_2[7], Cost_jalihal[7]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Kvvetills ttning (L g , 0.3)",
  legend = false
)

His_Sch9P_glutamine_H = His9 = bar(
  [label_alt_1,label_alt_2,label_jalihal],
  [Cost_alt_1[8], Cost_alt_2[8], Cost_jalihal[8]],
  bar_width = 1,
  color = [Color_alt_1, Color_alt_2, Color_jalihal],
  ylabel = "Cost",
  title = "Kvvetills ttning (H g , 1.0)",
  legend = false
)

His_Glutamine_addition_Sch9_gtr1Delta = His7 = bar(
```

```
    [label_alt_1,label_alt_2,label_jalihal],
    [Cost_alt_1[9], Cost_alt_2[9], Cost_jalihal[9]],
    bar_width = 1,
    color = [Color_alt_1, Color_alt_2, Color_jalihal],
    ylabel = "Cost",
    title = "Kv vetills ttning gtr1\\Delta",
    legend = false
  )

  His_Rapamycin_treatment = His10 = bar(
    [label_alt_1,label_alt_2,label_jalihal],
    [Cost_alt_1[10], Cost_alt_2[10], Cost_jalihal[10]],
    bar_width = 1,
    color = [Color_alt_1, Color_alt_2, Color_jalihal],
    ylabel = "Cost",
    title = "Rapamycin, (TORC1_T = 0)",
    legend = false
  )
  plot1 = plot(His1, His2, His3, His4, His5, His6, His7, His8, His9, His10,
  layout = (5, 2), titlefontsize = 14, guidefontsize = 10,
  guide_position = :left, margin= 20Plots.mm)
  plot!(plot1,size=(1500,2500))
  #display(plot1)
  return plot1
end
```

E.3.10 Rapamycin (TORC1 = 0)

Code/Make_image_of_optimization_results/Rapamycin_treatment.jl

```
#

using DifferentialEquations
using ModelingToolkit
using Plots

function Rapamycin_treatment()

    include("../Model/ODE_functions.jl")

    include("../Model/parameter_values.jl")

    include("../Model/ODE_methods.jl")
    include("../../Data/exp_data_norm.jl")
    include("../../Results/Parameter_values/Results_from_optimization.jl")
    include("../../Data/exp_data_norm.jl")
    u0_SS = Steady_state_solver(p_const, p_var_alt_1,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state f r parametrarna p

    # Post-shift: Rapamycin treatment => TORC1_T = 0.0
    p_const[Get_index(p_const_lookup_table, "TORC1_T")] =
    TORC1_T => 0.0

    # Steady-state vid sv ltning utg r initialv rden f r Glutamine addition
    tspan = (0.0, 90.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_1)

    plot1 = scatter(t_Rib_rap, data_Rib_rap, markersize = 5.5,
    color = color_data, markerstrokewidth=0.8, labels = "Data")

    plot!(plot1, sol, vars=Rib/(1e-3+u0_SS[25]), color = Color_alt_1, lw=2.5,
    labels= label_alt_1)

    p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 1.0
    u0_SS = Steady_state_solver(p_const, p_var_alt_2,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state f r parametrarna p

    p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 0.0
    # Steady-state vid sv ltning utg r initialv rden f r Glutamine addition
    tspan = (0.0, 90.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_alt_2)

    plot!(plot1, sol, vars=Rib/(1e-3+u0_SS[25]), color = Color_alt_2, lw=2.5,
    labels= label_alt_2)

    p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 1.0
    u0_SS = Steady_state_solver(p_const, p_var_jalihal,
```

```
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)) # Returnerar steady
    # state f r parametrarna p

    p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 0.0
    # Steady-state vid sv ltning utg r initialv rden f r Glutamine addition
    tspan = (0.0, 90.0) # [min]
    sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
    tspan, p_const, p_var_jalihal)

    plot!(plot1, sol, vars=Rib/(1e-3+u0_SS[25]), color = Color_jalihal,
    lw=2.5, labels= label_jalihal)

    xlabel!("t [min]")
    ylabel!("Rel. RPL32 mRNA") # Motsvarar konc. Rib relativt steady state
    title!("Rapamycin, (TORC1_T = 0)")
    return plot1
end
```

E.3.11 Figur över alla ODE lösningar med data

Code/Make_image_of_optimization_results/plot_for_all_ODE_solutions_with_data.jl

```
"""
    plot_for_all_ODE_solutions_with_data()

    Makes a subplot for all ODE solutions with data for all 3 parametervectors.
"""
function plot_for_all_ODE_solutions_with_data()
    include("Glucose_addition_Mig1.jl")
    include("Glucose_addition_Sch9.jl")
    include("Glucose_addition.jl")
    include("Glucose_starvation.jl")
    include("Glutamine_addition_gtr1.jl")
    include("Glutamine_addition.jl")
    include("Rapamycin_treatment.jl")

    plot_Glucose_addition_Mig1 = p1 = Glucose_addition_Mig1()
    plot_Glucose_addition_Sch9_cAMP = p2 = Glucose_addition_Sch9_cAMP()
    plot_Glucose_addition_cAMP = p3 = Glucose_addition_cAMP()
    plot_Glucose_addition_Sch9 = p4 = Glucose_addition_Sch9()
    plot_Glucose_starvation_Snf1 = p5 = Glucose_starvation_Snf1()
    plot_Glucose_starvation_Sch9 = p6 = Glucose_starvation_Sch9()
    plot_Glutamine_addition_gtr1 = p7 = Glutamine_addition_gtr1()
    plot_Glutamine_addition_L = p8 = Glutamine_addition_L()
    plot_Glutamine_addition_H = p9 = Glutamine_addition_H()
    plot_Rapamycin_treatment = p10 = Rapamycin_treatment()

    """
    display(plot_Glucose_addition_Mig1)
    display(plot_Glucose_addition_Sch9_cAMP)
    display(plot_Glucose_addition_cAMP)
    display(plot_Glucose_addition_Sch9)
    display(plot_Glucose_starvation_Snf1)
    display(plot_Glucose_starvation_Sch9)
    display(plot_Glutamine_addition_gtr1)
    display(plot_Glutamine_addition_L)
    display(plot_Glutamine_addition_H)
    display(plot_Rapamycin_treatment)
    """

    plot1 = plot(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, layout = (5, 2),
    legend = false, titlefontsize = 14, guidefontsize = 10,
    guide_position = :left, margin= 20Plots.mm)
    plot!(plot1,size=(1500,2500))
    #display(plot1)
    return plot1
end
```

E.4 Modell

E.4.1 Definierar ODE-modell

Code/Model/ODE_functions.jl

```
# Defines the ODE-model using ModelingToolkit

using ModelingToolkit
using DifferentialEquations
```

```
# Defines parameters, variables and derivative operator for constructing the model using ModelingToolkit
@parameters {gammacyr Cyr1_T w_cyr_glu w_cyr_snf sigma_cyr w_cyr w_dot sigma_dot w_dot_sch_pka Dot6_T
w_ego sigma_ego w_ego_basal EGO_T gammaego w_ego_gap gammagap sigma_gap w_gap_torc w_gap_N EGOGAP_T
gamma_gcn2 Gcn2_T w_gcn_torc sigma_gcn2 w_gcn w_gcn4_gcn2_trna w_gcn4 tRNA_sensitivity sigma_gcn4
tRNA_total Gcn4_T sigma_gis1 w_gis Gis1_T w_gis_pka w_gis_sch w_gln1_gln3 Gln1_T w_gln1 gammagln1
sigma_gln1 w_gln_snf sigma_gln w_gln_sit Gln3_T w_gln3 gammagln3 k_acc_pro k_acc_glu k_degr k_acc_nh4
NH4 Proline w_mig_snf w_mig sigma_mig1 Mig1_T gamma_mig w_mig_pka sigma_pde gammapde w_pde_pka PDE_T
w_pde w_pka w_pka_sch9 w_pka_camp sigma_pka gammapka k_pr tRNA_total w_ras_pka sigma_ras Ras_T gammaras
w_ras_glu w_ras k_mRNA_degr k_transcription w_rtg sigma_rtg w_rtg_torc Rtg13_T w_sak Sak_T w_sak_pka
sigma_sak w_sch9_torc w_sch9 gammasch9 sigma_sch9 Sch9_T w_snf_sak gammasnf w_snf_glc sigma_snf w_snf
Snf1_T w_torc_egoin w_torc w_torc_snf w_torc_glut sigma_tor gammator TORC1_T w_torc_ego w_tps_pka sigma_tps
gammatps w_tps PKA_T Tps1_T w_tre gammatre w_tre_pka Trehalase_T sigma_trehalase k_camp_cyr ATP k_camp_deg
k_camp_pde eIF_T w_eif_gcn2 sigma_eif gammaeif w_eif}

@variables{ t Cyr1(t) Dot6(t) EGO(t) EGOGAP(t) Gcn2(t) Gcn4(t) Gis1(t) Gln1(t) Gln3(t) Glutamine(t)
Mig1(t) PDE(t) PKA(t) Protein(t) Ras(t) Rib(t) Rtg13(t) Sak(t)
Sch9(t) Snf1(t) TORC1(t) Tps1(t) Trehalase(t) cAMP(t) eIF(t) Carbon(t) ATP(t) Glutamine_ext(t)}

D = Differential(t)

function soft_Heaviside(sigma, W)
    1/(1+exp(-sigma*W))
end

# Defines the ODE-system for the implemented model
eqs = [
    # Nutrient signal sensing and transduction
    D(Glutamine) ~ (k_acc_glu*Glutamine_ext+k_acc_pro*Proline+k_acc_nh4*NH4*Gln1*Carbon)-k_degr*Glutamine,
    D(Cyr1) ~ gammacyr*(Cyr1_T*soft_Heaviside(sigma_cyr, w_cyr_glu*Carbon*Ras-w_cyr-w_cyr_snf*Snf1)-Cyr1),
    D(Ras) ~ gammaras* (Ras_T*soft_Heaviside(sigma_ras,-w_ras_pka*PKA+w_ras_glu*Carbon+w_ras)-Ras),
    D(EGO) ~ gammaego*(EGO_T*soft_Heaviside(sigma_ego,w_ego_gap*EGOGAP*(Glutamine_ext+0.5*NH4+0.01*Proline)
    -w_ego*(1-Glutamine)-w_ego_basal)-EGO),
    D(EGOGAP) ~ gammagap*(EGOGAP_T*soft_Heaviside(sigma_gap, w_gap_N*(1-Glutamine)-w_gap_torc*TORC1)-EGOGAP),
    D(cAMP) ~ (k_camp_cyr*Cyr1*ATP-k_camp_pde*PDE*cAMP-k_camp_deg*cAMP),
    D(PDE) ~ gammapde*(PDE_T*soft_Heaviside(sigma_pde,w_pde_pka*PKA-w_pde)-PDE),
    D(Sak) ~ (Sak_T*soft_Heaviside(sigma_sak,w_sak-w_sak_pka*PKA)-Sak),
    # Master regulators
    D(TORC1) ~ gammator*(TORC1_T*soft_Heaviside(sigma_tor,w_torc_glut*Glutamine+w_torc_ego*EGO-w_torc_egoin*
    (1-EGO)-w_torc-w_torc_snf*Snf1)-TORC1),
    D(Snf1) ~ gammasnf*(Snf1_T*soft_Heaviside(sigma_snf,-w_snf_glc*Carbon+w_snf_sak*Sak-w_snf)-Snf1),
    D(PKA) ~ gammapka*(PKA_T*soft_Heaviside(sigma_pka, w_pka_camp*cAMP-w_pka-w_pka_sch9*Sch9)-PKA),
    D(Sch9) ~ gammasch9*(Sch9_T*soft_Heaviside(sigma_sch9,w_sch9_torc*TORC1-w_sch9)-Sch9),
    # Downstream responses
    D(Gcn2) ~ gamma_gcn2*(Gcn2_T*soft_Heaviside(sigma_gcn2,w_gcn-w_gcn_torc*Sch9)-Gcn2),
    D(Gcn4) ~ (Gcn4_T*soft_Heaviside(sigma_gcn4,w_gcn4_gcn2_trna*minimum([Gcn2, tRNA_sensitivity*
    (tRNA_total-minimum([tRNA_total, Glutamine]))])-w_gcn4)-Gcn4),
    D(eIF) ~ gammaeif*(eIF_T*soft_Heaviside(sigma_eif,w_eif-w_eif_gcn2*Gcn2)-eIF),
    D(Gln3) ~ gammagln3*(Gln3_T*soft_Heaviside(sigma_gln,-w_gln3+w_gln_snf*Snf1+ w_gln_sit*(1-TORC1))-Gln3),
    D(Gln1) ~ gammagln1*(Gln1_T*soft_Heaviside(sigma_gln1,w_gln1_gln3*Gln3-w_gln1)-Gln1),
    D(Rtg13) ~ (Rtg13_T*soft_Heaviside(sigma_rtg,-w_rtg_torc*TORC1+w_rtg)-Rtg13),
    D(Gis1) ~ (Gis1_T*soft_Heaviside(sigma_gis1,-w_gis_pka*PKA-w_gis_sch*Sch9+w_gis)-Gis1),
    D(Mig1) ~ gamma_mig*(Mig1_T*soft_Heaviside(sigma_mig1,w_mig_pka*PKA-w_mig_snf*Snf1+w_mig)-Mig1),
    D(Dot6) ~ (Dot6_T*soft_Heaviside(sigma_dot,-w_dot_sch_pka*Sch9*PKA+w_dot)-Dot6),
    D(Tps1) ~ gammatps*(Tps1_T*soft_Heaviside(sigma_tps, w_tps_pka*(PKA_T-PKA)-w_tps)-Tps1),
    D(Trehalase) ~ gammatre*(Trehalase_T*soft_Heaviside(sigma_trehalase, w_tre_pka*PKA-w_tre)-Trehalase),
    D(Protein) ~ k_pr*ATP*minimum([minimum([Rib, eIF]) minimum([tRNA_total, Glutamine])])*Protein,
    D(Rib) ~ k_transcription*(1-Dot6)-k_mRNA_degr*Rib, # 25, Rib
    # Model inputs which represent different nutrients. Values are defined as initial conditions
    D(Carbon) ~ 0,
    D(ATP) ~ 0,
    D(Glutamine_ext) ~ 0]

    @named ODE_sys = ODESystem(eqs)

    u_lookup_table = ["Glutamine(t)", "Cyr1(t)", "Ras(t)", "EGO(t)", "EGOGAP(t)", "cAMP(t)",
    "PDE(t)", "Sak(t)", "TORC1(t)", "Snf1(t)", "PKA(t)", "Sch9(t)",
    "Gcn2(t)", "Gcn4(t)", "eIF(t)", "Gln3(t)", "Gln1(t)", "Rtg13(t)",
    "Gis1(t)", "Mig1(t)", "Dot6(t)", "Tps1(t)", "Trehalase(t)",
    "Protein(t)", "Rib(t)", "Carbon(t)", "ATP(t)", "Glutamine_ext(t)"]
```

E.4.2 Parametervärden

Code/Model/parameter_values.jl

```
# Defines the parameter array with values Jalihal et. al.
# A new row represents parameters for the new ODE according to the comments below
# The parameters representing model inputs (ATP, Carbon, Glutamine_ext) are not included, since their
# values are defined as initial conditions when solving the ODEProblem

# p_var corresponds to the parameters that will undergo parameter estimation
p_var = [gammacyr => 8.96, w_cyr_glu => 5.13, w_cyr_snf => 0.12, w_cyr => 1.35, # Cyr1
w_dot => 0.29, w_dot_sch_pka => 0.16, # Dot6
```

```
w_ego => 0.28, w_ego_basal => 0.01, gammaego => 50.66, w_ego_gap => 2.21, # EGO
gammagap => 0.56, w_gap_torc => 88.33, w_gap_N => 7.76, # EGOGAP
gamma_gcn2 => 4.71, w_gcn_torc => 1.29, w_gcn => 0.12, # Gcn2
w_gcn4_gcn2_trna => 1.53, w_gcn4 => 0.74, tRNA_sensitivity => 74.51, tRNA_total => 2.47, # Gcn4
w_gis => 1.3, w_gis_pka => 3.3, w_gis_sch => 0.84, # Gis1
w_gln1_gln3 => 0.52, w_gln1 => 0.22, gammagln1 => 0.06, # Gln1
w_gln_snf => 3.9, w_gln_sit => 0.86, w_gln3 => 0.64, gammagln3 => 0.08, # Gln3
k_acc_pro => 0.0, k_acc_glu => 0.05, k_degr => 0.09, k_acc_nh4 => 0.0, # Glutamine
w_mig_snf => 1.21, w_mig => 10.64, gamma_mig => 0.66, w_mig_pka => 2.31, # Mig1
gammapde => 0.28, w_pde_pka => 2.89, w_pde => 0.38, # PDE
w_pka => 0.06, w_pka_sch9 => 17.5, w_pka_camp => 102.11, gammapka => 2.68, # PKA
k_pr => 0.02, # Protein
w_ras_pka => 1.87, gammaras => 1.82, w_ras_glu => 0.21, w_ras => 0.02, # Ras
k_mRNA_degr => 0.07, k_transcription => 0.24, # Rib
w_rtg => 0.19, w_rtg_torc => 0.88, # Rtg13
w_sak => 0.21, w_sak_pka => 0.38, # Sak
w_sch9_torc => 1.96, w_sch9 => 0.57, gammasch9 => 4.63, # Sch9
w_snf_sak => 1.52, gammasnf => 0.82, w_snf_glc => 1.15, w_snf => 0.54, # Snf1
w_torc_egoin => 0.3, w_torc_ego => 0.88, w_torc => 0.54, w_torc_snf => 0.44, w_torc_glut => 0.86,
gammator => 7.55, # TORC1
w_tps_pka => 0.57, gammatps => 0.47, w_tps => 0.05, # Tps1
w_tre => 1.07, gammatre => 0.34, w_tre_pka => 3.07, # Trehalase
k_camp_cyr => 10.87, k_camp_deg => 0.08, k_camp_pde => 14.12, # cAMP
w_eif_gcn2 => 0.28, gammaeif => 0.47, w_eif => 3.73] # eIF

p_var_names = first.(p_var)

# Create lookup table
p_var_lookup_table = []
for i in range(1, length(p_var_names))
    push!(p_var_lookup_table, string(p_var_names[i]))
end

# p_const contains the parameters that have fixed values
p_const = [Cyr1_T => 1.0, sigma_cyr => 3.5, # Cyr1
sigma_dot => 20.0, Dot6_T => 1.0, # Dot6
sigma_ego => 5.0, EGO_T => 1.0, # EGO
sigma_gap => 1.0, EGOGAP_T => 1.0, # EGOGAP
Gcn2_T => 1.0, sigma_gcn2 => 20.0, # Gcn2
sigma_gcn4 => 5.0, Gcn4_T => 1.0, # Gcn4
sigma_gis1 => 10.0, Gis1_T => 1.0, #Gis1
Gln1_T => 1.0, sigma_gln1 => 1.0, # Gln1
sigma_gln => 10.0, Gln3_T => 1.0, # Gln3
NH4 => 0.0, Proline => 0.0, # Glutamine
sigma_mig1 => 0.27, Mig1_T => 1.0, # Mig1
sigma_pde => 1.9, PDE_T => 1.0, # PDE
PKA_T => 1.0, sigma_pka => 1.0, # PKA
sigma_ras => 1.0, Ras_T => 1.0, # Ras
sigma_rtg => 10.0, Rtg13_T => 1.0, # Rtg13
Sak_T => 1.0, sigma_sak => 20.0, # Sak
sigma_sch9 => 8.0, Sch9_T => 1.0, # Sch9
sigma_snf => 3.0, Snf1_T => 1.0, # Snf1
TORC1_T => 1.0, sigma_tor => 5.0, # TORC1
sigma_tps => 5.0, Tps1_T => 1.0, # Tps1
Trehalase_T => 1.0, sigma_trehalase => 10.0, # Trehalase
eIF_T => 1.0, sigma_eif => 1.0] # eIF

p_const_names = first.(p_const)

p_const_lookup_table = []
for i in range(1, length(p_const_names))
    push!(p_const_lookup_table, string(p_const_names[i]))
end

p_conc = vcat(p_const, p_var)
p_conc_lookup_table = vcat(p_const_lookup_table, p_var_lookup_table)
```

E.4.3 Funktioner att lösa ODE-system m.m.

Code/Model/ODE_methods.jl

```
# Contains the functions and tools required to solve the ODE-model over time and to steady-state .
# Also contains various tools used throughout this process, such as a normalization function and and
# a tool to get the index of an element in an array using a lookup table

using DifferentialEquations
using DiffEqSensitivity
using Documenter
using Plots

color_jalihal = RGB(59/255,129/255,131/255)
```

```
include("ODE_functions.jl")

"""
    Steady_state_solver(p_const, p_var, model_inputs)
    Steady_state_solver(p_conc, model_inputs)

Compute the steady state for the parameters and nutrient conditions in ‘model_input‘

# Examples
‘‘‘julia-repl
julia> Steady_state_solver(p_conc, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
‘‘‘
"""
function Steady_state_solver(p_const, p_var, model_inputs)
    # Concatenates the component vectors in the form required as input by ModelingToolkit
    p_cat = vcat(p_const, p_var)

    u0 = zeros(length(eqs))  # Initial concentrations of are zero

    # Implements the nutrient condition values as initial guesses, values will be kept during calculations
    for i in range(1, length(model_inputs))
        u0[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end

    SS_prob = SteadyStateProblem(ODE_sys, u0, p_cat)

    SS_sol = solve(SS_prob, DynamicSS(Rodas4P(), abstol=1e-9, reltol=1e-9))
    return SS_sol.u
end

function Steady_state_solver(p_conc, model_inputs)

    u0 = zeros(length(eqs))

    for i in range(1, length(model_inputs))
        u0[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end
    SS_prob = SteadyStateProblem(ODE_sys, u0, p_conc)

    SS_sol = solve(SS_prob, DynamicSS(Rodas4P(), abstol=1e-9, reltol=1e-9))
    return SS_sol.u
end

"""
    ODE_solver(u0_SS, model_inputs, tspan, p_const, p_var)
    ODE_solver(u0_SS, model_inputs, tspan, p_conc)

Compute the solution of the ODE system with nutrient conditions of ‘model_inputs‘,
and the initial conditions in ‘u0_SS‘. Unit of time in ‘tspan‘ is minutes.

# Examples
‘‘‘julia-repl
julia> ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_conc)
‘‘‘
"""
function ODE_solver(u0_SS, model_inputs, tspan, p_const, p_var)

    p = vcat(p_const, p_var)

    # Implements the nutrient conditions as initial conditions, values will be kept during the calculations
    for i in range(1, length(model_inputs))
        u0_SS[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p)
    return solve(prob, Rodas4P(), abstol=1e-9, reltol=1e-9)
end

function ODE_solver(u0_SS, model_inputs, tspan, p_conc)

    for i in range(1, length(model_inputs))
        u0_SS[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_conc)
    return solve(prob, Rodas4P(), abstol=1e-9, reltol=1e-9)
end

"""

"""
```

```julia
function sensitivity_solver(u0_SS, model_inputs, tvals, p_const, p_var)

    p = vcat(p_const, p_var)

    for i in range(1, length(model_inputs))
        u0_SS[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end

    prob = ODEForwardSensitivityProblem(
        ODE_sys, u0_SS, [first(tvals),last(tvals)], p, sensealg=ForwardSensitivity(autodiff=false)
        )

    return solve(prob,Rodas4P(autodiff = false),saveat=tvals, sensealg=ForwardSensitivity(autodiff=false))
end

function sensitivity_solver(u0_SS, model_inputs, tspan, p_conc)

    for i in range(1, length(model_inputs))
        u0_SS[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
    end

    prob = ODEForwardSensitivityProblem(ODE_sys, u0_SS, tspan, p_conc)

    return solve(prob, Rodas4P())
end

"""
    Minmaxnorm(input_list)
    Minmaxnorm(input_list, min, max)

Normalize values in ‘input_list‘ according to a minimum and maximum value.
For an element in the list, normalization occurs according to (element-min)/(max-min).
The min/max values is either retrieved automatically from ‘input_list‘ or as function inputs.

# Examples
‘‘‘julia-repl
julia> Minmaxnorm([4, 2, 4, 1])
4-element Vector{Any}:
 1.0
 0.3333333333333333
 1.0
 0.0
‘‘‘
‘‘‘julia-repl
julia> Minmaxnorm([4, 2, 4, 1], 0.5, 6)
4-element Vector{Any}:
0.6363636363636364
0.2727272727272727
0.6363636363636364
0.09090909090909091
‘‘‘
"""
function Minmaxnorm(input_list)
    min = minimum(input_list)
    max = maximum(input_list)

    output_list = []
    for element in input_list
        append!(output_list, (element - min)/(max-min))     # Fills the output_list with normalized values
    end

    return output_list
end

function Minmaxnorm(input_list, min, max)

    output_list = []
    for element in input_list
        append!(output_list, (element - min)/(max-min))
    end

    return output_list
end

"""
    Get_index(list_lookup_table, key)

Retrieve the index of an element in a list using the corresponding ‘list_lookup_table‘
using a ‘key‘ in the form of a string.
"""
function Get_index(list_lookup_table, key)
    return findfirst(x->x==key, list_lookup_table) # Finds the first instance of the key and returns the index
end
```

```
nutrient_shifts = [
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0) => (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0) => (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0) => (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0) => (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0)
]
nutrient_shifts_lookup_table = [
    "Glucose starvation", "Glucose addition", "High glutamine", "Nitrogen starvation"
    ]

index_glucose_starvation = Get_index(nutrient_shifts_lookup_table, "Glucose starvation")
index_glucose_addition = Get_index(nutrient_shifts_lookup_table, "Glucose addition")
index_high_glutamine = Get_index(nutrient_shifts_lookup_table, "High glutamine")
index_nitrogen_starvation = Get_index(nutrient_shifts_lookup_table, "Nitrogen starvation")

"""

"""
function Steady_state_solver_FWD(p_const_FWD, p_var_FWD, model_inputs)

    p = vcat(p_var_FWD, p_const_FWD)

    u0 = zeros(length(eqs))

    for i in range(1, length(model_inputs))
        u0[findfirst(x->x==string(first.(model_inputs)[i]), u_lookup_table)] = last.(model_inputs)[i]
    end

    # Defines the SteadyStateProblem, which corresponds to a simple ODEProblem
    SS_prob = SteadyStateProblem(ODE_sys, u0, p)

    SS_sol = solve(SS_prob, DynamicSS(Rodas4P(), abstol=1e-3, reltol=1e-3), abstol=1e-8, reltol=1e-8)

    steady_state_solution = SS_sol.u

    # Checks derivatives
    if SS_sol.retcode != :Success
        du = zeros(length(SS_prob.u0))
        SS_prob.f(du, steady_state_solution, SS_prob.p, 1.0)
        #println(" all: ",du)
        for i in 1:length(du)
            if i     [2,3,6,8,11]
                if abs(du[i]) > 0.03
                    println("i that gave to big gradient after steady state")
                    println("i = $i")
                    steady_state_solution = "Not working" # Will be flagged as an error in the next step of optimization
                    break
                end
            end
        end
    end

    return steady_state_solution
end

"""

"""
function ODE_solver_FWDgrad(u0_SS, model_inputs, tspan, p_values, timelist_for_ode, prob)

    prev_model_inputs = (Carbon => u0_SS[end-2], ATP => u0_SS[end-1], Glutamine_ext => u0_SS[end])

    for i in range(1, length(model_inputs))
        u0_SS[findfirst(x->x==string(first.(model_inputs)[i]), u_lookup_table)] = last.(model_inputs)[i]
    end

    nprob = remake(prob; p = p_values)
    nprob = remake(nprob; u0_SS = u0_SS)
    nprob = remake(nprob; tspan = tspan)
    sol = solve(nprob,Rodas4P(),abstol=1e-8,reltol=1e-8, saveat = timelist_for_ode)

    for i in range(1, length(prev_model_inputs))
        u0_SS[findfirst(x->x==string(first.(prev_model_inputs)[i]), u_lookup_table)] = last.(prev_model_inputs)[i]
    end

    return sol#, maxiters = 1e5,dtmin = 1e-5)
end

"""

"""
```

```
function ODE_solver_FWD(u0_SS, model_inputs, tspan, p_const_FWD, p_var_FWD, timelist_for_ode)

    p = vcat(p_var_FWD, p_const_FWD)

    prev_model_inputs = (Carbon => u0_SS[end-2], ATP => u0_SS[end-1], Glutamine_ext => u0_SS[end])

    for i in range(1, length(model_inputs))
        u0_SS[findfirst(x->x==string(first.(model_inputs)[i]), u_lookup_table)] = last.(model_inputs)[i]
    end

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p)

    for i in range(1, length(prev_model_inputs))
        u0_SS[findfirst(x->x==string(first.(prev_model_inputs)[i]), u_lookup_table)] = last.(prev_model_inputs)[i]
    end

    return solve(prob,Rodas4P(),abstol=1e-8,reltol=1e-8, saveat = timelist_for_ode)#, maxiters = 1e5,dtmin = 1e-5)
end
```

E.5 Modellanalys

E.5.1 Test för att se om modellen är styv eller ej

Code/Model_analysis/stiff_or_nonstiff_test.jl

```
using DifferentialEquations
include("../Model/ODE_functions.jl")
include("../Model/ODE_methods.jl")
include("../Model/parameter_values.jl")

u0_SS = Steady_state_solver(p_conc, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))

model_inputs = (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)

for i in range(1, length(model_inputs))
    u0_SS[Get_index(u_lookup_table, string(first.(model_inputs)[i]))] = last.(model_inputs)[i]
end

prob = ODEProblem(ODE_sys, u0, (0.0, 30.0), p_conc)

# Compares solve times of stiff and non-stiff solver
println("Stiff solver:")
@time sol_stiff = solve(prob, Kvaerno4(), abstol=1e-8, reltol=1e-8)
println("Non-stiff solver:")
@time sol_nonstiff = solve(prob, Tsit5(), abstol=1e-8, reltol=1e-8)
println("RodasS (stiff-aware):")
@time sol_Rodas5 = solve(prob, Rodas5(), abstol=1e-8, reltol=1e-8)
println("Rodas4P (stiff-aware):")
@time sol_Rodas4P = solve(prob, Rodas4P(), abstol=1e-8, reltol=1e-8)

# The non-stiff solver ends up being faster than the stiff solver => The ODE problem is stiff
# In the end, Rodas4P() ends up being the fastest solver
```

E.5.2 Simuleringsmetoder

Code/Model_analysis/simulation_methods.jl

```
include("../Model/ODE_methods.jl")
include("../Model/parameter_values.jl")
include("../../Results/Parameter_values/Results_from_optimization.jl")

color_scheme = [Color_jalihal Color_alt_1 Color_alt_2]
labels = [label_jalihal label_alt_1 label_alt_2]

using Plots
default(dpi = 300)
default(titlefontsize=13)

"""
    Generate_sol_array(shift_index, tspan)
    Generate_sol_array(shift_index, tspan, p_changes)

Compute the solution to the ODE system for all three parameter sets during the shift indicated by its ‘shift_index‘.
Solutions are returned as an array in the order [Jalihal, alt1, alt2]

# Examples
‘‘‘julia-repl
julia> Generate_sol_array(index_glucose_addition, (0.0, 30.0), [EGO_T => 0.0])
‘‘‘
```

```
"""
function Generate_sol_array(shift_index, tspan)

    u0_SS_Jalihal = Steady_state_solver(p_conc_jalihal, first.(nutrient_shifts[shift_index]))
    sol_jalihal = ODE_solver(u0_SS_Jalihal, last.(nutrient_shifts[shift_index]), tspan, p_conc_jalihal)

    u0_SS_alt_1 = Steady_state_solver(p_conc_alt1, first.(nutrient_shifts[shift_index]))
    sol_alt1 = ODE_solver(u0_SS_alt_1, last.(nutrient_shifts[shift_index]), tspan, p_conc_alt1)

    u0_SS_alt_2 = Steady_state_solver(p_conc_alt2, first.(nutrient_shifts[shift_index]))
    sol_alt2 = ODE_solver(u0_SS_alt_2, last.(nutrient_shifts[shift_index]), tspan, p_conc_alt2)

    return [sol_jalihal, sol_alt1, sol_alt2]
end

function Generate_sol_array(shift_index, tspan, p_changes)

    # creates a copy of the parameter vectors, to prevent changes being passed to the original
    p_conc_jalihal_copy = copy(p_conc_jalihal)
    p_conc_alt1_copy = copy(p_conc_alt1)
    p_conc_alt2_copy = copy(p_conc_alt2)

    # implement parameter changes
    for i in range(1, length(p_changes))
        p_conc_jalihal_copy[Get_index(p_conc_lookup_table, string(first.(p_changes)[i]))] =
        first.(p_changes)[i] => last.(p_changes)[i]
    end

    for i in range(1, length(p_changes))
        p_conc_alt1_copy[Get_index(p_conc_lookup_table, string(first.(p_changes)[i]))] =
        first.(p_changes)[i] => last.(p_changes)[i]
    end

    for i in range(1, length(p_changes))
        p_conc_alt2_copy[Get_index(p_conc_lookup_table, string(first.(p_changes)[i]))] =
        first.(p_changes)[i] => last.(p_changes)[i]
    end

    u0_SS_Jalihal = Steady_state_solver(p_conc_jalihal_copy, first.(nutrient_shifts[shift_index]))
    sol_jalihal = ODE_solver(u0_SS_Jalihal, last.(nutrient_shifts[shift_index]), tspan, p_conc_jalihal_copy)

    u0_SS_alt_1 = Steady_state_solver(p_conc_alt1_copy, first.(nutrient_shifts[shift_index]))
    sol_alt1 = ODE_solver(u0_SS_alt_1, last.(nutrient_shifts[shift_index]), tspan, p_conc_alt1_copy)

    u0_SS_alt_2 = Steady_state_solver(p_conc_alt2_copy, first.(nutrient_shifts[shift_index]))
    sol_alt2 = ODE_solver(u0_SS_alt_2, last.(nutrient_shifts[shift_index]), tspan, p_conc_alt2_copy)

    return [sol_jalihal, sol_alt1, sol_alt2]
end

"""
    plot_simulation_result(show_plot, sol_array, output_variable, ylims, plot_title, file_name, legend_placement)
    plot_simulation_result(show_plot, sol_array, output_variable, xlims, ylims, plot_title, file_name, legend_placement)

Plot the results of simulation for a specific ‘output_variable‘, using the corresponding ‘sol_array‘.
Also save the plot both as PNG and PDF using ‘file_name‘
"""
function plot_simulation_result(show_plot::Bool, sol_array, output_variable::Num, y_lims, plot_title::String,
                                file_name::String, legend_placement)

    # iterates over the sol_array, using both solution and its index
    for (index, sol) in enumerate(sol_array)
        if index == 1 # create plot object
        global plot1 = plot(
            sol, vars = output_variable, title=plot_title, legend=legend_placement, label=labels[index],
            color=color_scheme[index], lw=1.5, show = false
            )
        else    # draw the rest of the graphs
            plot!(sol, vars = output_variable, label=labels[index], color=color_scheme[index], lw=1.5, show = false)
        end
    end
    ylims!(y_lims)
    xlabel!("t [min]")
    ylabel!(replace("$output_variable", "(t)" => ""))

    savefig(plot1, pwd()*"/Results/Model_analysis/pixel_images/"*file_name*".png")
    savefig(plot1, pwd()*"/Results/Model_analysis/vector_images/"*file_name*".pdf")

    if show_plot == true
        display(plot1)
    end
end
```

```
function plot_simulation_result(
    show_plot::Bool, sol_array, output_variable::Num, x_lims, y_lims,
    plot_title::String, file_name::String, legend_placement
    )

    for (index, sol) in enumerate(sol_array)
        if index == 1
        global plot1 = plot(
            sol, vars = output_variable, title=plot_title, legend=legend_placement, label=labels[index],
            color=color_scheme[index], lw=1.5, show = false
            )
        else
            plot!(sol, vars = output_variable, label=labels[index], color=color_scheme[index], lw=1.5, show = false)
        end
    end

    ylims!(y_lims)
    xlims!(x_lims)
    xlabel!("t [min]")
    ylabel!(replace("$output_variable", "(t)" => ""))

    savefig(plot1, pwd()*"/Results/Model_analysis/pixel_images/"*file_name*".png")
    savefig(plot1, pwd()*"/Results/Model_analysis/vector_images/"*file_name*".pdf")

    if show_plot == true
        display(plot1)
    end
end

"""
    plot_wt_mut_results(show_plot, sol_array_wt, sol_array_mut, output_variable, x_lims, y_lims, plot_title, file_name)

Plot the results of a simulation for both the wild type and mutant, using the corresponding sol_arrays.
The lines will be drawn in the same plot using different line types for wildtype and mutant.
"""
function plot_wt_mut_results(
    show_plot::Bool, sol_array_wt, sol_array_mut, output_variable::Num, x_lims, y_lims, plot_title::String,
    file_name::String, legend_placement
    )

    sol_arrays = [sol_array_wt, sol_array_mut]

    for (index_array, sol_array) in enumerate(sol_arrays)
        for (index, sol) in enumerate(sol_array)
            if index_array == 1     # plot the wild type
                if index == 1   # create the plot object
                global plot1 = plot(
                    sol, vars = output_variable, title=plot_title, legend=legend_placement, label="",
                    color=color_scheme[index], linestyle=:dash, lw=1.5, show = false
                    )
                else
                    plot!(
                        sol, vars = output_variable, label="", linestyle=:dash,
                         color=color_scheme[index], lw=1.5, show = false
                         )
                end
            else    # plot the mutants
                plot!(sol, vars = output_variable, label="", color=color_scheme[index], lw=1.5, show = false)
            end
        end
    end

    # Adds legend entries manually
    plot!([1], [0], label = "Jalihal", color=color_jalihal)
    plot!([1], [0], label="Parametervektor 1", color=Color_alt_1)
    plot!([1], [0], label = "Parametervektor 2", color=Color_alt_2)
    plot!([1], [0], linestyle=:dash, label = "vildtyp", color="black")
    plot!([1], [0], label="mutant", color="black")

    ylims!(y_lims)
    xlims!(x_lims)
    xlabel!("t [min]")
    ylabel!(replace("$output_variable", "(t)" => ""))

    savefig(plot1, pwd()*"/Results/Model_analysis/pixel_images/"*file_name*".png")
    savefig(plot1, pwd()*"/Results/Model_analysis/vector_images/"*file_name*".pdf")

    if show_plot == true
        display(plot1)
    end
end
```

```
###### For plotting all simulation results in subplot #######

function return_simulation_result(sol_array, output_variable::Num, x_lims, y_lims, plot_title::String, legend_placement)

    for (index, sol) in enumerate(sol_array)
        if index == 1
        global plot1 = plot(
            sol, vars = output_variable, title=plot_title, legend=legend_placement, label=labels[index],
            color=color_scheme[index], show = false
            )
        else
            plot!(sol, vars = output_variable, label=labels[index], color=color_scheme[index], show = false)
        end
    end

    ylims!(y_lims)
    xlims!(x_lims)
    ylabel!(replace("$output_variable", "(t)" => ""))
    if string(output_variable)=="Gis1(t)" && plot_title=="Glukostillsättning"
        xlabel!("t [min]")
    else
        xlabel!("")
    end

    return plot1
end

function return_wt_mut_results(sol_array_wt, sol_array_mut, output_variable::Num, x_lims, y_lims,
                               plot_title::String, legend_placement)

    sol_arrays = [sol_array_wt, sol_array_mut]

    for (index_array, sol_array) in enumerate(sol_arrays)
        for (index, sol) in enumerate(sol_array)
            if index_array == 1     # plot the wild type
                if index == 1   # create the plot object
                global plot1 = plot(sol, vars = output_variable, title=plot_title, legend=legend_placement,
                                    label="", color=color_scheme[index], linestyle=:dash, show = false)
                else
                    plot!(
                        sol, vars = output_variable, label="", linestyle=:dash,
                        color=color_scheme[index], show = false
                        )
                end
            else    # plot the mutants
                plot!(sol, vars = output_variable, label="", color=color_scheme[index], show = false)
            end
        end
    end

    # Adds invisible lines for extra legend entries
    plot!([1], [0], label = "Jalihal", color=color_jalihal)
    plot!([1], [0], label="Parametervektor 1", color=Color_alt_1)
    plot!([1], [0], label = "Parametervektor 2", color=Color_alt_2)
    plot!([1], [0], linestyle=:dash, label = "vildtyp", color="black")
    plot!([1], [0], label="mutant", color="black")

    ylims!(y_lims)
    xlims!(x_lims)
    xlabel!("t [min]")
    ylabel!(replace("$output_variable", "(t)" => ""))

    return plot1
end
```

E.5.3 Simuleringar

Code/Model_analysis/simulations.jl

```
include("simulation_methods.jl")

###### Glucose starvation ######
sol_array_gluc_starve = Generate_sol_array(index_glucose_starvation, (0.0, 40.0)) # Generates array of simulation

plot_simulation_result(
    true, sol_array_gluc_starve, Mig1, (0.0, 20.0), (0.8, 1.0), "Glukossvältning", "Mig1_gluc_starve", :topright
)
plot_simulation_result(
    true, sol_array_gluc_starve, Snf1, (0.0, 1.0), "Glukossvältning", "Snf1_gluc_starve", :right
    )
```

```
plot_simulation_result(
    true, sol_array_gluc_starve, Gis1, (0.0, 10.0), (0.0, 1.0), "Glukossvältning", "Gis1_gluc_starve", :topright
    )
plot_simulation_result(
    true, sol_array_gluc_starve, Gln3, (0.0, 1.0), "Glukossvältning", "Gln3_gluc_starve", :right
    )
plot_simulation_result(
    true, sol_array_gluc_starve, Dot6, (0.0, 4.0), (0.8, 1.0), "Glukossvältning", "Dot6_gluc_starve", :topright
    )


###### Nitrogen starvation ######
sol_array_nit_starve = Generate_sol_array(index_nitrogen_starvation, (0.0, 40.0))

plot_simulation_result(
    true, sol_array_nit_starve, Snf1, (0.0, 15.0), (0.0, 0.2), "Kvävesvältning", "Snf1_nit_starve", :right
    )
plot_simulation_result(
    true, sol_array_nit_starve, TORC1, (0.0, 1.0), "Kvävesvältning", "TORC1_nit_starve", :right
    )
plot_simulation_result(
    true, sol_array_nit_starve, Sch9, (0.0, 1.0), "Kvävesvältning", "Sch9_nit_starve", :right
    )
plot_simulation_result(
    true, sol_array_nit_starve, Gln3, (0.0, 1.0), "Kvävesvältning", "Gln3_nit_starve" ,:right
    )
plot_simulation_result(
    true, sol_array_nit_starve, Rtg13, (0.0, 1.0), "Kvävesvältning", "Rtg13_nit_starve" ,:bottomright
    )
plot_simulation_result(
    true, sol_array_nit_starve, Dot6, (0.0, 10.0), (0.8, 1.0), "Kvävesvältning", "Dot6_nit_starve" ,:right
    )


###### Glucose addition ######
sol_array_gluc_add = Generate_sol_array(index_glucose_addition, (0.0, 15.0))

plot_simulation_result(
    true, sol_array_gluc_add, cAMP, (0.0, 15.0), (0.0, 1.05), "Glukostillsättning", "cAMP_gluc_add", :right
    )
plot_simulation_result(
    true, sol_array_gluc_add, PKA, (0.0, 15.0), (0.0, 1.05), "Glukostillsättning", "PKA_gluc_add", :bottomright
    )
plot_simulation_result(
    true, sol_array_gluc_add, Snf1, (0.0, 10.0), (0.0, 1.0), "Glukostillsättning", "Snf1_gluc_add", :right
    )
plot_simulation_result(
    true, sol_array_gluc_add, Gis1, (0.0, 7.0), (0.0, 1.0), "Glukostillsättning", "Gis1_gluc_add", :right
    )

###### Gtr1Delta | EGO deletion, Nitrogen addition ######
sol_array_nit_add = Generate_sol_array(index_high_glutamine, (0.0, 40.0))
sol_array_gtr1Delta = Generate_sol_array(index_high_glutamine, (0.0, 40.0), [EGO_T => 0.0])
plot_wt_mut_results(
    true, sol_array_nit_add, sol_array_gtr1Delta, Gln3, (0.0, 25.0), (0.0, 1.0),
     "Kvävetillsättning, gtr1\$\\Delta\$", "Gln3_gtr1Delta_nit_add", :left
    )

###### Snf1 deletion, Glucose starvation ######
sol_array_Snf1Delta = Generate_sol_array(index_glucose_starvation, (0.0, 1.0), [Snf1_T => 0.0])

plot_wt_mut_results(
    true, sol_array_gluc_starve, sol_array_Snf1Delta, Gln3, (0.0, 0.5), (0.0, 0.8),
    "Glukossvältning, Snf1\$\\Delta\$", "Cyr1_gluc_add", :right
    )

plot_wt_mut_results(
    true, sol_array_gluc_starve, sol_array_Snf1Delta, Cyr1, (0.0, 0.5), (0.0, 0.8),
    "Glukossvältning, Snf1\$\\Delta\$", "Cyr1_gluc_add", :right
    )
```

E.5.4 Rita resultat från alla simuleringar i en figur

Code/Model_analysis/plot_all_sims_in_subplot.jl

```
include("simulation_methods.jl")

default(titlefontsize=5)
default(xtickfont=4)
default(ytickfont=4)
default(guidefont=6)
default(ylabelfontsize=5)
```

```
default(xlabelfontsize=5)
default(lw=0.6)

###### Glucose starvation ######
sol_array_gluc_starve = Generate_sol_array(index_glucose_starvation, (0.0, 40.0)) # Generates array of simulation

p1=return_simulation_result(sol_array_gluc_starve, Mig1, (0.0, 20.0), (0.8, 1.0), "Glukossvältning", :topright)
p2=return_simulation_result(sol_array_gluc_starve, Snf1, (0.0, 40.0), (0.0, 1.1), "Glukossvältning", :right)
p3=return_simulation_result(sol_array_gluc_starve, Gis1, (0.0, 10.0), (0.0, 1.0), "Glukossvältning", :topright)
p4=return_simulation_result(sol_array_gluc_starve, Gln3, (0.0, 40.0), (0.0, 1.1), "Glukossvältning", :right)
p5=return_simulation_result(sol_array_gluc_starve, Dot6, (0.0, 4.0), (0.9, 1.0), "Glukossvältning",
:topright)


###### Nitrogen starvation ######
sol_array_nit_starve = Generate_sol_array(index_nitrogen_starvation, (0.0, 40.0))

p6=return_simulation_result(sol_array_nit_starve, Snf1, (0.0, 15.0), (0.0, 0.10), "Kvävesvältning", :right)
p7=return_simulation_result(sol_array_nit_starve, TORC1, (0.0, 40.0), (0.0, 1.1), "Kvävesvältning", :right)
p8=return_simulation_result(sol_array_nit_starve, Sch9, (0.0, 40.0), (0.0, 1.1), "Kvävesvältning", :right)
p9=return_simulation_result(sol_array_nit_starve, Gln3, (0.0, 40.0), (0.0, 1.1), "Kvävesvältning", :right)
p10=return_simulation_result(sol_array_nit_starve, Rtg13, (0.0, 40.0), (0.0, 1.1), "Kvävesvältning", :bottomright)
p11=return_simulation_result(sol_array_nit_starve, Dot6, (0.0, 10.0), (0.9, 1.0), "Kvävesvältning",:bottomright)


###### Glucose addition ######
sol_array_gluc_add = Generate_sol_array(index_glucose_addition, (0.0, 15.0))

p12=return_simulation_result(sol_array_gluc_add, Snf1, (0.0, 10.0), (0.0, 1.0), "Glukostillsättning", :right)
p13=return_simulation_result(sol_array_gluc_add, Gis1, (0.0, 7.0), (0.0, 1.0), "Glukostillsättning", :right)

###### Gtr1Delta | EGO deletion, Nitrogen addition ######
sol_array_nit_add = Generate_sol_array(index_high_glutamine, (0.0, 40.0))
sol_array_gtr1Delta = Generate_sol_array(index_high_glutamine, (0.0, 40.0), [EGO_T => 0.0])

p14=return_wt_mut_results(
    sol_array_nit_add, sol_array_gtr1Delta, Gln3, (0.0, 25.0), (0.0, 1.0),
    "Kvävetillsättning, gtr1\$\\Delta\$", :right
    )

###### Snf1 deletion, Glucose starvation ######
sol_array_Snf1Delta = Generate_sol_array(index_glucose_starvation, (0.0, 1.0), [Snf1_T => 0.0])

p15=return_wt_mut_results(
    sol_array_gluc_starve, sol_array_Snf1Delta, Gln3, (0.0, 0.5), (0.0, 0.15),
    "Glukossvältning, Snf1\$\\Delta\$", :right
    )
p16=return_wt_mut_results(
    sol_array_gluc_starve, sol_array_Snf1Delta, Cyr1, (0.0, 0.5), (0.0, 0.7),
    "Glukossvältning, Snf1\$\\Delta\$", :right
    )

plot_list = [p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, p16]
l = @layout [
    grid(4,4)
]

plot_all = plot(plot_list..., layout=l, legend = false)
display(plot_all)

file_name = "model_analysis_subplot"
savefig(plot_all, pwd()*"/Results/Model_analysis/"*file_name*".pdf")
savefig(plot_all, pwd()*"/Results/Model_analysis/"*file_name*".png")
```

E.6 Kod för optimeringen

E.6.1 Både LHS och optimering

Code/Parameter_Estimation/Run_all.jl

```
# Starts by making a latin hypercube sample with 10000 samples on
# logarithmic scale between -3 and 3, with x working ones. Then these are
# optimized.
include("../Latin_Hypercube_Sampling/latin_hypercube_sampling.jl")
main()
include("optimize_all_p_0_values.jl")
```

E.6.2 Optimering

Code/Parameter_Estimation/Optimization.jl

```
using LinearAlgebra
using DifferentialEquations
using Plots
using DiffEqSensitivity
using SteadyStateDiffEq
using ForwardDiff
using ProgressMeter
using ModelingToolkit
include("../Model/ODE_functions.jl")
include("../Model/ODE_methods.jl")
include("../../Data/exp_data.jl")

"""
    main_while_loop(p_var_FWD, p_const_FWD, my, my2, abs_min_alpha,
    condition_num_to_use_SD)

    All calculations to be done per iteration.
"""
function main_while_loop(p_var_FWD, p_const_FWD, my, my2, abs_min_alpha,
    condition_num_to_use_SD)
    N = 0

    #prog = ProgressUnknown(;dt=0.1,desc="Progress:", color=:green)
    step_dir = []
    step_length = []
    Hessian_inv = []
    grad = []
    prev_grad = []
    MK = []

    while true
        N += 1
        println("N:",N)
        #ProgressMeter.next!(prog)

        if N == 1
            try
                New_MK = calc_cost(p_var_FWD, p_const_FWD) # Calculates the
                # cost of the initial parameter values
                MK = append!(MK, New_MK)
            catch
                return nothing
                break
            end
        end
        if N == 1
            grad = []
            try
                grad = calc_grad(p_var_FWD, p_const_FWD)
            catch
                break
            end
        end

        Hessian_inv = calc_hessian(prev_grad, grad, step_length,
        Hessian_inv, p_var_FWD)

        if cond(inv(Hessian_inv)) > condition_num_to_use_SD || N == 1
            # Atm I'm not really using cond(inv(Hessian_inv)) >
            #condition_num_to_use_SD
            # I'm doing steepest descent for the first iteration, N == 1
            println(" USING STEEPEST ")
            step_dir = -grad
            step_length = steepest_descent_calc_length(step_dir, p_var_FWD,
            p_const_FWD, abs_min_alpha, MK)
        else
            step_dir = calc_direction(grad, Hessian_inv)
            step_length, alpha_control = calc_length(step_dir, p_var_FWD,
            p_const_FWD, MK, grad, my, my2, abs_min_alpha)
            if alpha_control == 0
                println(" USING STEEPEST ")
                step_dir = -grad
                step_length = steepest_descent_calc_length(step_dir,
                p_var_FWD, p_const_FWD, abs_min_alpha, MK)
            end
        end

        p_values_process = exp.(log.(last.(p_var_FWD))+step_length)
        # new p values based on step length calculated previously

        p_values_dual_temp = [Pair(first.(p_var_FWD)[1], p_values_process'[1])]
        for i in range(2, length(last.(p_var_FWD)))
```

```
            B = Pair(first.(p_var_FWD)[i], p_values_process'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end
        p_var_FWD = p_values_dual_temp # Makes it into a Pair vector

        New_MK = calc_cost(p_var_FWD, p_const_FWD)

        MK = append!(MK, New_MK)

        prev_grad = grad

        grad = []
        try
            grad = calc_grad(p_var_FWD, p_const_FWD)
        catch
            break
        end

        Number_of_fulfilled_criteria = termination_criteria(MK, grad,
        step_length, p_var_FWD)
        if Number_of_fulfilled_criteria >= 2
            break
        end
    end
    #ProgressMeter.finish!(prog)

    return MK, p_var_FWD, N
end

"""
    termination_criteria(MK, grad, step_length, p_var_FWD)

    Checks how many termination criterias are fulfilled.
"""
function termination_criteria(MK, grad, step_length, p_var_FWD)

    epsilon_1 = 1e-6
    epsilon_2 = 1e-6
    epsilon_3 = 1e-6

    Number_of_fulfilled_criteria = 0
    #criteria_1
    if norm(grad) <= epsilon_1*(1+abs(MK[end]))
        Number_of_fulfilled_criteria += 1
    end
    #criteria_2
    if MK[end-1] - MK[end] <= epsilon_2*(1+abs(MK[end]))
        Number_of_fulfilled_criteria += 1
    end
    #criteria_3
    if norm(step_length) <= epsilon_3*(1+norm(last.(p_var_FWD)))
        Number_of_fulfilled_criteria += 1
    end

    return Number_of_fulfilled_criteria
end

"""
    calc_hessian(prev_grad, grad, step_length, Hessian_inv, p_var_FWD)

    Calculate the inverse of the hessian.
"""
function calc_hessian(prev_grad, grad, step_length, Hessian_inv, p_var_FWD)

    if prev_grad == []
        dimentions = size(last.(p_var_FWD))[1]
        Hessian = Matrix(I,dimentions,dimentions)
        Hessian_inv  = inv(Hessian)
    else
        y = grad .- prev_grad
        #Hessian invers with " B r o y d e n FletcherGoldfarbShanno  algorithm"
        Hessian_inv = Hessian_inv .+ (step_length'*y +
        y'*(Hessian_inv*y))*step_length*step_length'/((y'*step_length)^2) .-
        (Hessian_inv*y*step_length' .+
        step_length*y'*Hessian_inv)/(step_length'*y)
    end

    return Hessian_inv
end

"""
    calc_direction(grad, Hessian_inv)
```

```
    Calculate the step direction.
"""
function calc_direction(grad, Hessian_inv)

    step_dir = -Hessian_inv*grad

    return step_dir
end

"""
    calc_length(step_dir, p_var_FWD, p_const_FWD, MK, grad, my, my2,
    abs_min_alpha)

    Finds a step length that fulfills the Armijo and Wolfe condition
"""
function calc_length(step_dir, p_var_FWD, p_const_FWD, MK, grad, my, my2,
    abs_min_alpha)
    #g   i stegriktningen
    alpha = 1
    p_var_new = []

    while true
        #Minimum alpha allowed so it doesn't go to inf.
        if alpha < abs_min_alpha
            println(" BAD ALPHA ")
            alpha = 0
            # used to switch to steepest descent if alpha < abs_min_alpha
            # for quasi-newton
            break
        end

        p_values_prev = exp.(log.(last.(p_var_FWD))+alpha*step_dir)

        p_values_dual_temp = [Pair(first.(p_var_FWD)[1], p_values_prev'[1])]
        for i in range(2, length(last.(p_var_FWD)))
            B = Pair(first.(p_var_FWD)[i], p_values_prev'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end
        p_var_new = p_values_dual_temp

        MK_comp_val = 0
        try
            MK_comp_val = calc_cost(p_var_new, p_const_FWD) # Checks if the
            # cost can be calculated for the parameters
        catch
            println(" Can't calculate MK in Quasi, alpha = ",alpha)
            alpha  = alpha/2
            continue
        end
        println(" QUASI_MK: ", MK_comp_val)

        MK_diff = MK_comp_val - MK[end]

        Armijo_cond = my * alpha * step_dir[:,end]' * grad[:,end]


        if MK_diff > Armijo_cond
            alpha  = alpha/2
        else
            println("Fullfilled first condition!")
            D = 0
            try
                D = calc_grad(p_var_new, p_const_FWD)
            catch
                println(" Can't calculate grad in Quasi, alpha = ",alpha)
                alpha  = alpha/2
                continue
            end

            step_direct_grad = -step_dir[:,end]' * D

            Wolfe_cond = -my2 * step_dir[:,end]' * grad[:,end]

            if abs(step_direct_grad) < abs(Wolfe_cond)
                #strong Wolfe conditions
                break
            else
                alpha = alpha/2
            end
        end

    end
```

```
    step_length = alpha*step_dir[:,end]

    return step_length, alpha
end

"""
    calc_grad(p_var_FWD, p_const_FWD)

    Calculate the gradient of the cost function.
"""
function calc_grad(p_var_FWD, p_const_FWD)

    function dMK_dp_Glucose_addition_Mig1(p_var_FWD,p_const_FWD, u0_SS)
        function calc_FWD_grad(p_values)
            p_values = exp.(p_values)
            sol = ODE_solver_FWDgrad(u0_SS, (ATP => 1.0, Carbon => 1.0,
            Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)

            u_approx = getindex.(sol.u, 20) #ODE for MIG1 on row 20 in
            # ODE system

            MK_for_diff = sum((data_for_ode -
            (log.(10,u_approx./(1e-5 .+ 1.0 .- u_approx)))).^2)
            return MK_for_diff
        end

        data_for_ode = raw_data_Mig1_glucose_relief
        timelist_for_ode = t_Mig1_glucose_relief
        tspan = (0.0, 20.0) # [min]
        p_values = vcat(p_var_FWD,p_const_FWD)

        prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
        dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

        return dMK_dp
    end

    function dMK_dp_Glucose_addition_Sch9(p_var_FWD,p_const_FWD)
        function calc_FWD_grad(p_values)
            p_values = exp.(p_values)
            sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
            Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
            u_approx = getindex.(sol.u, 6)

            MK_for_diff = sum((data_for_ode - u_approx).^2)
            return MK_for_diff
        end

        data_for_ode = Minmaxnorm(raw_data_sch9Delta_cAMP, 0.052, 1.227)
        timelist_for_ode = t_sch9Delta_cAMP
        tspan = (0.0, 3.5) # [min]
        p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
        Sch9_T => 0.0
        p_values = vcat(p_var_FWD,p_const_FWD)
        u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
        (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))
        prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
        dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))
        p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
        Sch9_T => 1.0

        return dMK_dp
    end

    function dMK_dp_Glucose_addition_cAMP(p_var_FWD,p_const_FWD, u0_SS)
        function calc_FWD_grad(p_values)
            p_values = exp.(p_values)
            sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
            Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
            u_approx = getindex.(sol.u, 6)

            MK_for_diff = sum((data_for_ode - u_approx).^2)
            return MK_for_diff
        end

        data_for_ode = Minmaxnorm(raw_data_cAMP, 0.052, 1.227)
        timelist_for_ode = t_cAMP
        tspan = (0.0, 3.5) # [min]
        p_values = vcat(p_var_FWD,p_const_FWD)

        prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
        dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))
```

```
    return dMK_dp
end

function dMK_dp_Glucose_addition_Sch9_p(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
        Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 12)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_relief)
    timelist_for_ode = t_Sch9_glucose_relief
    tspan = (0.0, 32) # [min]
    p_values = vcat(p_var_FWD,p_const_FWD)

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

    return dMK_dp
end

function dMK_dp_Glucose_starvation_Snf1_p(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 0.0, ATP => 0.0,
        Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 10)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Snf1)
    timelist_for_ode = t_Snf1
    tspan = (0.0, 62) # [min]
    p_values = vcat(p_var_FWD,p_const_FWD)

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

    return dMK_dp
end

function dMK_dp_Glucose_starvation_Sch9_p(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 0.0, ATP => 0.0,
        Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 12)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_starve)
    timelist_for_ode = t_Sch9_glucose_starve
    tspan = (0.0, 32) # [min]
    p_values = vcat(p_var_FWD,p_const_FWD)

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

    return dMK_dp
end

function dMK_dp_Sch9P_glutamine_L(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
        Glutamine_ext => 0.3), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 12)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_L, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_L
```

```
    tspan = (0.0, 32) # [min]
    p_values = vcat(p_var_FWD,p_const_FWD)

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

    return dMK_dp
end

function dMK_dp_Sch9P_glutamine_H(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
        Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 12)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_H, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_H
    tspan = (0.0, 32) # [min]
    p_values = vcat(p_var_FWD,p_const_FWD)

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))

    return dMK_dp
end

function dMK_dp_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD,p_const_FWD)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS,
        (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0), tspan,
        p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 12)

        MK_for_diff = sum((data_for_ode - u_approx).^2)
        return MK_for_diff
    end

    data_for_ode = Minmaxnorm(raw_data_Sch9_gtr1Delta, 4.82, 52.27)
    timelist_for_ode = t_Sch9_gtr1Delta
    tspan = (0.0, 32) # [min]
    w_torc_ego_true =
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")]
    w_torc_egoin_true =
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")]

    # Mutant gtr1_Delta => EGO_T = 0, w_torc_ego = 0,
    # w_torc_egoin = 0 i
    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] =
    EGO_T => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0
    p_values = vcat(p_var_FWD,p_const_FWD)
    u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
    dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))
    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 1.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego_true
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin_true

    return dMK_dp
end

function dMK_dp_Rapamycin_treatment(p_var_FWD,p_const_FWD, u0_SS)
    function calc_FWD_grad(p_values)
        p_values = exp.(p_values)
        sol = ODE_solver_FWDgrad(u0_SS, (Carbon => 1.0, ATP => 1.0,
        Glutamine_ext => 1.0), tspan, p_values, timelist_for_ode, prob)
        u_approx = getindex.(sol.u, 25)

        MK_for_diff =
        sum((data_for_ode - (u_approx./(1e-3 .+ u0_SS[25]))).^2)
```

```
            return MK_for_diff
        end

        data_for_ode = Minmaxnorm(raw_data_Rib_rap)
        timelist_for_ode = t_Rib_rap
        tspan = (0.0, 92.0) # [min]

        p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
        TORC1_T => 0.0
        p_values = vcat(p_var_FWD,p_const_FWD)
        prob = ODEProblem(ODE_sys, u0_SS, tspan, p_values)
        dMK_dp = ForwardDiff.gradient(calc_FWD_grad, log.(prob.p))
        p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
        TORC1_T => 1.0

        return dMK_dp
    end

    u0_SS_A_C_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
    #println("u0_SS_A_C_G:",u0_SS_A_C_G)
    dMK_dp_Glucose_starvation_Snf1_p_solution =
    dMK_dp_Glucose_starvation_Snf1_p(p_var_FWD,p_const_FWD, u0_SS_A_C_G)
    dMK_dp_Glucose_starvation_Sch9_p_solution =
    dMK_dp_Glucose_starvation_Sch9_p(p_var_FWD,p_const_FWD, u0_SS_A_C_G)
    dMK_dp_Rapamycin_treatment_solution =
    dMK_dp_Rapamycin_treatment(p_var_FWD,p_const_FWD, u0_SS_A_C_G)

    u0_SS_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (ATP => 0.0, Carbon => 0.0, Glutamine_ext => 1.0))
    #println("u0_SS_G:",u0_SS_G)
    dMK_dp_Glucose_addition_Mig1_solution =
    dMK_dp_Glucose_addition_Mig1(p_var_FWD,p_const_FWD, u0_SS_G)
    dMK_dp_Glucose_addition_cAMP_solution =
    dMK_dp_Glucose_addition_cAMP(p_var_FWD,p_const_FWD, u0_SS_G)
    dMK_dp_Glucose_addition_Sch9_p_solution =
    dMK_dp_Glucose_addition_Sch9_p(p_var_FWD,p_const_FWD, u0_SS_G)

    u0_SS_A_C = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0))
    #println("u0_SS_A_C:",u0_SS_A_C)
    dMK_dp_Sch9P_glutamine_L_solution =
    dMK_dp_Sch9P_glutamine_L(p_var_FWD,p_const_FWD, u0_SS_A_C)
    dMK_dp_Sch9P_glutamine_H_solution =
    dMK_dp_Sch9P_glutamine_H(p_var_FWD,p_const_FWD, u0_SS_A_C)

    dMK_dp_Glucose_addition_Sch9_solution =
    dMK_dp_Glucose_addition_Sch9(p_var_FWD,p_const_FWD)
    dMK_dp_Glutamine_addition_Sch9_gtr1Delta_solution =
    dMK_dp_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD,p_const_FWD)

    True_grad = dMK_dp_Glucose_addition_Mig1_solution +
    dMK_dp_Glucose_addition_Sch9_solution +
    dMK_dp_Glucose_addition_cAMP_solution +
    dMK_dp_Glucose_addition_Sch9_p_solution +
    dMK_dp_Glucose_starvation_Snf1_p_solution +
    dMK_dp_Glucose_starvation_Sch9_p_solution +
    dMK_dp_Sch9P_glutamine_L_solution +
    dMK_dp_Sch9P_glutamine_H_solution +
    dMK_dp_Glutamine_addition_Sch9_gtr1Delta_solution +
    dMK_dp_Rapamycin_treatment_solution

    True_grad = Rearrange_the_gradient(p_var_FWD, p_const_FWD, True_grad)

    return True_grad
end

"""
    Rearrange_the_gradient(p_var_FWD, p_const_FWD, dMK_dp)

    Rearrange the gradient so that they are in the correct order.
    This is needed since the combination of FAD and ModelingToolkit results
    in some strange rearranging of the gradient, this also happens to the
    parameters if you use prob.p to get them from the ODEproblem.
    (This effect is easier to see in a simpler example.)
"""
function Rearrange_the_gradient(p_var_FWD, p_const_FWD, dMK_dp)

    tspan = (0.0, 30.0) # Simply needs to be defined, but that values
    # doesn't matter
    u0_SS = ones(1,28) # Simply needs to be defined, but that values
    # doesn't matter
```

```
    p_var_FWD_new = []
    for i in 1:length(last.(p_var_FWD))
        append!(p_var_FWD_new,1 + 0.01*i)
    end

    p_const_FWD_new = []
    for i in 1:length(last.(p_const_FWD))
        append!(p_const_FWD_new,2 + 0.01*i)
    end

    p_values_process = vcat(p_var_FWD_new, p_const_FWD_new)

    p_original = vcat(p_var_FWD,p_const_FWD)

    p_values_dual_temp = [Pair(first.(p_original)[1], p_values_process'[1])]
    for i in range(2, length(last.(p_original)))
        B = Pair(first.(p_original)[i], p_values_process'[i])
        p_values_dual_temp = vcat(p_values_dual_temp,B)
    end
    p_new = p_values_dual_temp

    prob = ODEProblem(ODE_sys, u0_SS, tspan, p_new)

    all_p_values_after_prob = prob.p

    p_var_FWD_values_after_prob = []
    for i in all_p_values_after_prob
        if i < 2.0
            append!(p_var_FWD_values_after_prob,round((i-1)*100; digits = 3))
        end
    end

    Pre_process_grad = []

    for i in 1:length(dMK_dp)
        if all_p_values_after_prob[i] < 2
            append!(Pre_process_grad,dMK_dp[i])
        end
    end

    finished_process_grad = []
    for i in 1:length(p_var_FWD_values_after_prob)
        N = 0
        for item in p_var_FWD_values_after_prob
            N += 1
            if item == i
                append!(finished_process_grad,Pre_process_grad[N])
                break
            end
        end
    end

    return finished_process_grad
end

"""
    calc_cost(p_var_FWD, p_const_FWD)

    Calculate the sum of least square / cost for all chosen ODE:s
"""
function calc_cost(p_var_FWD, p_const_FWD)

    function Calc_cost_Glucose_addition_Mig1(p_var_FWD, p_const_FWD, u0_SS)

        data_for_ode = raw_data_Mig1_glucose_relief
        timelist_for_ode = t_Mig1_glucose_relief
        tspan = (0.0, 20.0) # [min]

        sol = ODE_solver_FWD(u0_SS,
        (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0), tspan,
        p_const_FWD, p_var_FWD, timelist_for_ode)

        u_approx = getindex.(sol.u, 20) #ODE for MIG1 on row 20 in ODE system

        #Check if sol was a success
        u_approx = Check_if_cost_is_a_success(sol, u_approx)

        MK_Glucose_addition_Mig1 = sum((data_for_ode -
        (log.(10,u_approx./(1e-5 .+ 1.0 .- u_approx)))).^2)

        return MK_Glucose_addition_Mig1
    end
```

```
function Calc_cost_Glucose_addition_Sch9(p_var_FWD, p_const_FWD)

    data_for_ode = Minmaxnorm(raw_data_sch9Delta_cAMP, 0.052, 1.227)
    timelist_for_ode = t_sch9Delta_cAMP
    tspan = (0.0, 3.5) # [min]
    # Mutant Sch9_Delta => Sch9_T = 0
    p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
    Sch9_T => 0.0

    # Pre-shift => ATP, Carbon = 0
    u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0)) # Returnerar
    # steady state  f  r  parametrarna p

    # Post-shift => ATP, Carbon = 1
    sol = ODE_solver_FWD(u0_SS, (Carbon => 1.0, ATP => 1.0,
    Glutamine_ext => 1.0), tspan, p_const_FWD, p_var_FWD,
    timelist_for_ode)

    u_approx = getindex.(sol.u, 6) #ODE for cAMP on row 6 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_Sch9 = sum((data_for_ode - u_approx).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "Sch9_T")] =
    Sch9_T => 1.0

    return MK_Glucose_addition_Sch9
end

function Calc_cost_Glucose_addition_cAMP(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_cAMP, 0.052, 1.227)
    timelist_for_ode = t_cAMP
    tspan = (0.0, 3.5) # [min]

    # Post-shift, Glucose addition => ATP, Carbon = 1
    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 6) #ODE for cAMP on row 6 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_cAMP = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_addition_cAMP
end

function Calc_cost_Glucose_addition_Sch9_p(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_relief)
    timelist_for_ode = t_Sch9_glucose_relief
    tspan = (0.0, 30) # [min]

    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_addition_Sch9_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_addition_Sch9_p
end

function Calc_cost_Glucose_starvation_Snf1_p(p_var_FWD, p_const_FWD,u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Snf1)
    timelist_for_ode = t_Snf1
    tspan = (0.0, 62) # [min]

    # Post-shift, Glucose starvation => Carbon, ATP = 0

    sol = ODE_solver_FWD(u0_SS,
```

```
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)
    u_approx = getindex.(sol.u, 10) #ODE for Snf1 on row 10 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_starvation_Snf1_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_starvation_Snf1_p
end

function Calc_cost_Glucose_starvation_Sch9_p(p_var_FWD, p_const_FWD,u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9_glucose_starve)
    timelist_for_ode = t_Sch9_glucose_starve
    tspan = (0.0, 30) # [min]

    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glucose_starvation_Sch9_p = sum((data_for_ode - u_approx).^2)

    return MK_Glucose_starvation_Sch9_p
end

function Calc_cost_Sch9P_glutamine_L(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_L, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_L
    tspan = (0.0, 32) # [min]

    # Low glutamine => Glutamine_ext = 0.3
    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Sch9P_glutamine_L = sum((data_for_ode - u_approx).^2)

    return MK_Sch9P_glutamine_L
end

function Calc_cost_Sch9P_glutamine_H(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Sch9P_glutamine_H, 4.82, 52.57)
    timelist_for_ode = t_Sch9P_glutamine_H
    tspan = (0.0, 32) # [min]

    # High Glutamine => Glutamine_ext = 1.0
    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Sch9P_glutamine_H = sum((data_for_ode - u_approx).^2)

    return MK_Sch9P_glutamine_H
end

function Calc_cost_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD, p_const_FWD)

    data_for_ode = Minmaxnorm(raw_data_Sch9_gtr1Delta, 4.82, 52.27)
    timelist_for_ode = t_Sch9_gtr1Delta
    tspan = (0.0, 32) # [min]

    w_torc_ego_true =
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")]
    w_torc_egoin_true =
```

```
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")]

    # Mutant gtr1_Delta => EGO_T = 0, w_torc_ego = 0, w_torc_egoin = 0 i
    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego => 0.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin => 0.0

    # Pre-shift => Glutamine_ext = 0, ATP, Carbon = 1
    u0_SS = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
    (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

    # High glutamine => Glutamine_ext = 1
    sol = ODE_solver_FWD(u0_SS,
    (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 12) #ODE for Sch9 on row 12 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Glutamine_addition_Sch9_gtr1Delta =
    sum((data_for_ode - u_approx).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 1.0
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_ego")] =
    w_torc_ego_true
    p_var_FWD[Get_index(p_var_lookup_table, "w_torc_egoin")] =
    w_torc_egoin_true

    return MK_Glutamine_addition_Sch9_gtr1Delta
end

function Calc_cost_Rapamycin_treatment(p_var_FWD, p_const_FWD, u0_SS)

    data_for_ode = Minmaxnorm(raw_data_Rib_rap)
    timelist_for_ode = t_Rib_rap
    tspan = (0.0, 92.0) # [min]

    # Post-shift: Rapamycin treatment => TORC1_T = 0.0
    p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
    TORC1_T => 0.0

    sol = ODE_solver_FWD(u0_SS,
    (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan,
    p_const_FWD, p_var_FWD, timelist_for_ode)

    u_approx = getindex.(sol.u, 25) #ODE for RIB on row 25 in ODE system

    #Check if sol was a success
    u_approx = Check_if_cost_is_a_success(sol, u_approx)

    MK_Rapamycin_treatment =
    sum((data_for_ode - (u_approx./(1e-3 + u0_SS[25]))).^2)

    p_const_FWD[Get_index(p_const_lookup_table, "TORC1_T")] =
    TORC1_T => 1.0

    return MK_Rapamycin_treatment
end

u0_SS_A_C_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))
MK_Glucose_starvation_Snf1_p =
Calc_cost_Glucose_starvation_Snf1_p(p_var_FWD, p_const_FWD, u0_SS_A_C_G)
MK_Glucose_starvation_Sch9_p =
Calc_cost_Glucose_starvation_Sch9_p(p_var_FWD, p_const_FWD, u0_SS_A_C_G)
MK_Rapamycin_treatment =
Calc_cost_Rapamycin_treatment(p_var_FWD, p_const_FWD, u0_SS_A_C_G)

u0_SS_G = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
(ATP => 0.0, Carbon => 0.0, Glutamine_ext => 1.0))
MK_Glucose_addition_Mig1 =
Calc_cost_Glucose_addition_Mig1(p_var_FWD, p_const_FWD, u0_SS_G)
MK_Glucose_addition_cAMP =
Calc_cost_Glucose_addition_cAMP(p_var_FWD, p_const_FWD, u0_SS_G)
MK_Glucose_addition_Sch9_p =
Calc_cost_Glucose_addition_Sch9_p(p_var_FWD, p_const_FWD, u0_SS_G)

u0_SS_A_C = Steady_state_solver_FWD(p_const_FWD, p_var_FWD,
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0))
```

```
    MK_Sch9P_glutamine_L =
    Calc_cost_Sch9P_glutamine_L(p_var_FWD, p_const_FWD, u0_SS_A_C)
    MK_Sch9P_glutamine_H =
    Calc_cost_Sch9P_glutamine_H(p_var_FWD, p_const_FWD, u0_SS_A_C)

    MK_Glucose_addition_Sch9 =
    Calc_cost_Glucose_addition_Sch9(p_var_FWD, p_const_FWD)
    MK_Glutamine_addition_Sch9_gtr1Delta =
    Calc_cost_Glutamine_addition_Sch9_gtr1Delta(p_var_FWD, p_const_FWD)

    MK = MK_Glucose_addition_Mig1 + MK_Glucose_addition_Sch9 +
    MK_Glucose_addition_cAMP + MK_Glucose_addition_Sch9_p +
    MK_Glucose_starvation_Snf1_p + MK_Glucose_starvation_Sch9_p +
    MK_Sch9P_glutamine_L + MK_Sch9P_glutamine_H +
    MK_Glutamine_addition_Sch9_gtr1Delta + MK_Rapamycin_treatment

    return MK
end

"""
    Check_if_cost_is_a_success(sol, u_approx)

    Checks if sol.retcode != :Success
"""
function Check_if_cost_is_a_success(sol, u_approx)
    if sol.retcode != :Success
        println(" Can't Calc Cost ")
        u_approx = "Not working"
    end
    return u_approx
end

"""
    plot_results(Bild, Bild_format, MK, Number_of_iterations)

    Plot the chosen ODE with or without data points.
"""
function plot_results(Bild, Bild_format, MK, Number_of_iterations)
    if Bild == true
        gr()
        plotd = plot((0:Number_of_iterations),MK,legend=:topright)
        plot!(plotd,(0:Number_of_iterations),MK, seriestype = :scatter)
        xlabel!("Iteration")
        ylabel!("Kostnad")
        title!("Kostnad Vs Iteration")
        if Bild_format == "PNG"
            savefig(plotd, pwd()*"/Results/MK_plot.png")
        elseif Bild_format == "SVG"
            savefig(plotd, pwd()*"/Results/MK_plot.svg")
        end
        display(plotd)
    end
end

"""
    steepest_descent_calc_length(step_dir, p_var_FWD, p_const_FWD,
    abs_min_alpha, MK)

    Finds a step length for steepest descent.
"""
function steepest_descent_calc_length(step_dir, p_var_FWD, p_const_FWD,
    abs_min_alpha, MK)
    alpha0 = 1e-9
    alpha = 1e-9
    p_var_temp = []
    abs_min_alpha = abs_min_alpha*1e-10

    while true

        p_values_prev = exp.(log.(last.(p_var_FWD)) + (alpha-alpha0)*step_dir)

        println(" Alpha: ", alpha)

        p_values_dual_temp = [Pair(first.(p_var_FWD)[1], p_values_prev'[1])]
        for i in range(2, length(last.(p_var_FWD)))
            B = Pair(first.(p_var_FWD)[i], p_values_prev'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end
        p_var_temp = p_values_dual_temp

        prev_MK = 0
        try
```

```
            prev_MK = calc_cost(p_var_temp, p_const_FWD)
        catch
            if alpha <= abs_min_alpha
                alpha = 0.0
                break
            end
            if alpha > alpha0
                alpha = alpha-alpha0
                break
            else
                alpha0 = alpha0/10
                alpha = alpha0
                continue
            end
        end

        p_values_new = exp.(log.(last.(p_var_FWD))+alpha*step_dir)

        p_values_dual_temp = [Pair(first.(p_var_FWD)[1], p_values_new'[1])]
        for i in range(2, length(last.(p_var_FWD)))
            B = Pair(first.(p_var_FWD)[i], p_values_new'[i])
            p_values_dual_temp = vcat(p_values_dual_temp,B)
        end

        p_var_new = p_values_dual_temp

        new_MK = 0
        try
            new_MK = calc_cost(p_var_new, p_const_FWD)
        catch
            if alpha <= abs_min_alpha
                alpha = 0.0
                break
            end
            if alpha > alpha0
                alpha = alpha-alpha0
                break
            else
                alpha0 = alpha0/10
                alpha = alpha0
                continue
            end
        end
        println("New_MK: ",new_MK)


        # If alpha = 0 it will try reducing alpha0 until alpha is not 0 or
        # alpha0 > abs_min_alpha
        if new_MK > prev_MK
            alpha = alpha-alpha0
            if alpha == 0
                if alpha0 <= abs_min_alpha
                    break
                end
                alpha0 = alpha0/10
                alpha = alpha0
                continue
            end
            break
        else
            if alpha <= abs_min_alpha
                alpha = 0.0
                break
            end
            if alpha >= 10*alpha0
                alpha0 = 10*alpha0
            end

            if round(alpha) >= 1e6
                break
            end

            alpha = alpha + alpha0
        end
    end

    step_length =alpha*step_dir[:,end]

    println(" Final_ALPHA: ",alpha)

    return step_length
end
```

```
"""
    main_FWD_optimization(p_0_var)

    Here you choose different options and values for the entire optimisation
    process.
"""
function main_FWD_optimization(p_0_var)

    include("../Model/parameter_values.jl")

    p_var = p_0_var
    condition_num_to_use_SD = 1e100 #Not really using,
    # hence a big number 1e100

    #calc_length options:
    my = 1e-4
    my2 = 0.95
    #my2 = 1e3
    abs_min_alpha = 1e-8

    ####################Quasi-Newton#####################
    MK, p_var_reslut, Number_of_iterations = @time main_while_loop(p_var,
    p_const, my, my2, abs_min_alpha, condition_num_to_use_SD)
    #########################################################

    #plot cost vs iterations
    Bild = false
    Bild_format = "PNG" #Defined for PNG and SVG
    plot_results(Bild, Bild_format, MK, Number_of_iterations)
    return MK, p_var_reslut
end

#include("../Model/parameter_values.jl")
#p_0_var = p_var
#MK, p_var_result = main_FWD_optimization(p_0_var)
#println(p_var_result)
#println(MK)
```

E.6.3 Optimering av parametervektorer i en lista

Code/Parameter_Estimation/optimize_all_p_0_values.jl

```
using CSV
using DataFrames
using LinearAlgebra
using DifferentialEquations
using Plots
using DiffEqSensitivity
using SteadyStateDiffEq
using ForwardDiff
using ProgressMeter
using ModelingToolkit
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")
include("../Model/ODE_methods.jl")
include("../../Data/exp_data.jl")
include("Optimization.jl")

"""
    main_Op_p_0()

    Optimize the cost for multiple parametervectors from p_0_values.csv and
    puts the result in a CSV file
"""
function main_Op_p_0()
    include("../Model/parameter_values.jl")

    df = CSV.read("Intermediate/p_0_values.csv", DataFrame)
    List_of_p_that_didnt_work = []

    for i in 1:(length(df[1,:])-1)
        println(" I : ",i)

        p_names_var = first.(p_var)

        df = CSV.read("Intermediate/p_0_values.csv", DataFrame)

        p_0_var = df[:, Symbol("p_0_value_$i")][1:end-1]
```

```
        p_var_temp = [Pair(p_names_var[1], p_0_var'[1])]
        for item in range(2, length(p_0_var))
            B = Pair(p_names_var[item], p_0_var'[item])
            p_var_temp = vcat(p_var_temp,B)
        end
        p_0_var = p_var_temp

        MK = []
        p_var_result = []
        try
            MK, p_var_result = Optimize_p(p_0_var)
        catch
            append!(List_of_p_that_didnt_work,i)
            continue
        end

        parameters_and_MK = vcat(first.(p_var),Num("Cost"))

        values_of_p_and_MK = vcat(last.(p_var_result),MK[end])
        Prior_values_of_p_and_MK = vcat(last.(p_0_var),MK[1])

        if i == 1
            df = DataFrame(Parameter = parameters_and_MK,
                        p_0_value_1 = Prior_values_of_p_and_MK,
                        Optimum_from_p_0_value_1 = values_of_p_and_MK)
            CSV.write("Intermediate/optimization_of_p0.csv",df)
        else
            df = CSV.read("Intermediate/optimization_of_p0.csv", DataFrame)
            #print(" length_p_0_v: ",length(df[:, Symbol
            # ("Optimum_from_p_0_value_1")]))
            df[!,Symbol("p_0_value_$i")] = Prior_values_of_p_and_MK
            df[!,Symbol("Optimum_from_p_0_value_$i")] = values_of_p_and_MK
            CSV.write("Intermediate/optimization_of_p0.csv",df)
        end
    end

    println("List_of_p_that_didnt_work: ",List_of_p_that_didnt_work)
end

"""
    Optimize_p(p_0_var)

    Optimize the cost for p_0_var.
"""
function Optimize_p(p_0_var)
    MK, p_var_reslut = main_FWD_optimization(p_0_var)
    return MK, p_var_reslut
end

main_Op_p_0()
```

E.7 Störningsexperiment

E.7.1 Funktioner för att simulera störningsexperiment

Code/Perturbation/perturbation_methods.jl

```
include("../Model/ODE_methods.jl")
include("../Model/parameter_values.jl")

"""
    Steady_state_pertubation_solver(model_inputs_preshift, model_inputs_postshift, output_variable)
    Steady_state_pertubation_solver(p_changes, model_inputs_preshift, model_inputs_postshift, output_variable)

Calculate the steady state values both for pre- and postshift nutrient conditions. Values are returned as a Pair{Float64,
"""
function Steady_state_pertubation_solver(model_inputs_preshift, model_inputs_postshift, output_variable)

    u0_SS_preshift = Steady_state_solver(p_const, p_var, model_inputs_preshift)
    u0_SS_postshift = Steady_state_solver(p_const, p_var, model_inputs_postshift)

    index = Get_index(u_lookup_table, string(output_variable))

    return (u0_SS_preshift[index] => u0_SS_postshift[index])
end

function Steady_state_pertubation_solver(p_changes, model_inputs_preshift, model_inputs_postshift, output_variable)

    p_conc_copy = copy(p_conc)

    # implement parameter changes
```

```
    for i in range(1, length(p_changes))
        p_conc_copy[Get_index(p_conc_lookup_table, string(first.(p_changes)[i]))] =
        first.(p_changes)[i] => last.(p_changes)[i]
    end

    u0_SS_preshift = Steady_state_solver(p_conc_copy, model_inputs_preshift)
    u0_SS_postshift = Steady_state_solver(p_conc_copy, model_inputs_postshift)

    index = Get_index(u_lookup_table, string(output_variable))

    return (u0_SS_preshift[index] => u0_SS_postshift[index])
end

"""
    Minmaxnorm_shifts(shifts_array)
    Minmaxnorm_shifts(shifts_array, min, max)

Normalizes the results of simulations of a type of shift according to ‘Minmaxnorm()‘, using
either min/max retrived from the shift value or as function inputs.
"""
function Minmaxnorm_shifts(shifts_array)

    for i in 1:length(shifts_array)
        c = shifts_array[i]
        min = minimum([c.first, c.second])
        max = maximum([c.first, c.second])

        norm_vals = Minmaxnorm([c.first, c.second], min, max)
        shifts_array[i] = norm_vals[1] => norm_vals[2]
    end

    return shifts_array
end

function Minmaxnorm_shifts(shift, min, max)

    for i in 1:length(shift)
        c = shift[i]
        norm_vals = Minmaxnorm([first.(c) last.(c)], min, max)
        shift[i] = norm_vals[1] => norm_vals[2]
    end

    return shift
end

"""
    Minmaxnorm_shifts(shifts_array)
    Minmaxnorm_shifts(shifts_array, min, max)

Normalizes the data according to ‘Minmaxnorm()‘, using either min, max retrived from the data or as function inputs
"""
function Minmaxnorm_data(data)

    min = minimum([first.(data) last.(data)])
    max = maximum([first.(data) last.(data)])

    for i in 1:length(data)
        c = data[i]
        norm_vals = Minmaxnorm([first.(c) last.(c)], min, max)
        data[i] = norm_vals[1] => norm_vals[2]
    end

    return data
end

function Minmaxnorm_data(data, min, max)

    for i in 1:length(data)
        c = data[i]
        norm_vals = Minmaxnorm([first.(c) last.(c)], min, max)
        data[i] = norm_vals[1] => norm_vals[2]
    end

    return data
end
```

E.7.2 Simuleringar av störningsexperiment

Code/Perturbation/perturbation_experiments.jl

```
# Creates the simulations from the Perturbation-Experiment for both wild typ and mutant
```

```
include("perturbation_methods.jl")

# Perturbation simulations Snf1
Snf1_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_glucose_starvation]), last.(nutrient_shifts[index_glucose_starvation]), Snf1)
Snf1_mutant_shift = Steady_state_pertubation_solver(
    [Sak_T => 0.0], first.(nutrient_shifts[index_glucose_starvation]),
    last.(nutrient_shifts[index_glucose_starvation]), Snf1
    )

# Gis1
Gis1_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_glucose_addition]), last.(nutrient_shifts[index_glucose_addition]), Gis1
    )
Gis1_mutant_shift = Steady_state_pertubation_solver(
    [Sch9_T => 0.0], first.(nutrient_shifts[index_glucose_addition]),
    last.(nutrient_shifts[index_glucose_addition]), Gis1
    )

# Nth1 (Trehalas)
Tre_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_glucose_starvation]), last.(nutrient_shifts[index_glucose_starvation]), Trehalase
    )
Tre_mutant_shift = Steady_state_pertubation_solver(
    [PKA_T => 0.3], first.(nutrient_shifts[index_glucose_starvation]),
    last.(nutrient_shifts[index_glucose_starvation]), Trehalase
    )

# cAMP
cAMP_shift_pde = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_glucose_addition]), last.(nutrient_shifts[index_glucose_addition]), cAMP
    )
cAMP_mutant_pde_shift = Steady_state_pertubation_solver(
    [PDE_T => 0.0], first.(nutrient_shifts[index_glucose_addition]),
    last.(nutrient_shifts[index_glucose_addition]), cAMP
    )
cAMP_shift_ras = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_glucose_addition]),
    last.(nutrient_shifts[index_glucose_addition]), cAMP
    )
cAMP_mutant_ras_shift = Steady_state_pertubation_solver(
    [Ras_T => 0.0, w_pka_camp => 0.0], first.(nutrient_shifts[index_glucose_addition]),
    last.(nutrient_shifts[index_glucose_addition]), cAMP
    )

# Rib
Rib_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_high_glutamine]), last.(nutrient_shifts[index_high_glutamine]), Rib
    )
Rib_mutant_shift = Steady_state_pertubation_solver(
    [Sch9_T => 0.0], first.(nutrient_shifts[index_high_glutamine]),
    last.(nutrient_shifts[index_high_glutamine]), Rib)

# Sch9 wild type
Sch9_lst4_lst7_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_high_glutamine]), last.(nutrient_shifts[index_high_glutamine]), Sch9
    )
Sch9_gtr1_gtr2_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_high_glutamine]), last.(nutrient_shifts[index_high_glutamine]), Sch9
    )
Sch9_gtr1_2_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_high_glutamine]), last.(nutrient_shifts[index_high_glutamine]), Sch9
    )

# Sch9 mutants
Sch9_lst4_lst7_mutant_shift = Steady_state_pertubation_solver(
    [EGOGAP_T => 0.0], first.(nutrient_shifts[index_high_glutamine]),
    last.(nutrient_shifts[index_high_glutamine]), Sch9
    )
Sch9_gtr1_gtr2_mutant_shift = Steady_state_pertubation_solver(
    [EGO_T => 0.0, w_torc_ego => 0.0, w_torc_egoin => 0.0, w_torc_glut => 0.5],
    first.(nutrient_shifts[index_high_glutamine]), last.(nutrient_shifts[index_high_glutamine]), Sch9
    )

Sch9_gtr1_2_mutant_shift = Steady_state_pertubation_solver(
    [EGO_T => 0.0, w_torc_ego => 0.0, w_torc_egoin => 0.0], first.(nutrient_shifts[index_high_glutamine]),
    last.(nutrient_shifts[index_high_glutamine]), Sch9
    )

# Gcn4
Gcn4_shift = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_nitrogen_starvation]), last.(nutrient_shifts[index_nitrogen_starvation]), Gcn4
    )
Gcn4_mutant_shift = Steady_state_pertubation_solver(
```

```
    [Gcn2_T => 0.0], first.(nutrient_shifts[index_nitrogen_starvation]),
    last.(nutrient_shifts[index_nitrogen_starvation]), Gcn4
    )

# Gln3
Gln3_shift_sit = Steady_state_pertubation_solver(
    first.(nutrient_shifts[index_nitrogen_starvation]), last.(nutrient_shifts[index_nitrogen_starvation]), Gln3)
Gln3_mutant_shift_sit = Steady_state_pertubation_solver(
    [w_gln_sit => 0.0, TORC1_T => 0.0], first.(nutrient_shifts[index_nitrogen_starvation]),
     last.(nutrient_shifts[index_nitrogen_starvation]), Gln3
     )

# Normalization of Rib and cAMP simulations
cAMP_shifts = [cAMP_shift_ras, cAMP_shift_pde, cAMP_mutant_ras_shift, cAMP_mutant_pde_shift]
min_cAMP = minimum([first.(cAMP_shifts) last.(cAMP_shifts)])
max_cAMP = maximum([first.(cAMP_shifts) last.(cAMP_shifts)])

Rib_shifts = [Rib_shift, Rib_mutant_shift]
min_Rib = minimum([first.(Rib_shifts) last.(Rib_shifts)])
max_Rib = maximum([first.(Rib_shifts) last.(Rib_shifts)])


cAMP_shifts_wt = [cAMP_shift_pde, cAMP_shift_ras]
cAMP_shifts_wt_norm = Minmaxnorm_shifts(cAMP_shifts_wt, min_cAMP, max_cAMP)

Rib_shifts_wt = [Rib_shift]
Rib_shifts_wt_norm = Minmaxnorm_shifts(Rib_shifts_wt, min_Rib, max_Rib)

raw_shifts_wt = [Snf1_shift, Gis1_shift, Tre_shift, Sch9_gtr1_2_shift, Gcn4_shift, Gln3_shift_sit,
                 Sch9_gtr1_gtr2_shift, Sch9_lst4_lst7_shift
]

# Normalisation of the expeimental data
# The first pair is wild typ and the next pair is mutant for each simulation
exp_data_Rib = [150.0 => 600.0, 100.0 => 100.0]
exp_data_Rib_norm = Minmaxnorm_data(exp_data_Rib)

exp_data_cAMP_pde = [0.25 => 1.1, 0.25 => 3.8]
exp_data_cAMP_pde_norm = Minmaxnorm_data(exp_data_cAMP_pde)

exp_data_cAMP_ras = [0.25 => 1.1, 0.25 => 0.25]
exp_data_cAMP_ras_norm = Minmaxnorm_data(exp_data_cAMP_ras, 0.0, 3.8)

exp_data_Snf1 = [0.05 => 2.25, 0.01 => 0.01]
exp_data_Snf1_norm = Minmaxnorm_data(exp_data_Snf1)

exp_data_Gis1 = [700.0 => 35.0, 100.0 => 10.0]
exp_data_Gis1_norm = Minmaxnorm_data(exp_data_Gis1)

exp_data_Tre = [100.0 => 25.0, 50.0 => 25.0]
exp_data_Tre_norm = Minmaxnorm_data(exp_data_Tre)

exp_data_Sch9_gtr1_2 = [5.0 => 55.0, 5.0 => 55.0]
exp_data_Sch9_gtr1_2_norm = Minmaxnorm_data(exp_data_Sch9_gtr1_2)

exp_data_Gcn4 = [7.3 => 100.0, 11.0 => 38.0]
exp_data_Gcn4_norm = Minmaxnorm_data(exp_data_Gcn4)

exp_data_Gln3_sit = [461.0 => 6248.0, 0.0 => 789.0]
exp_data_Gln3_sit_norm = Minmaxnorm_data(exp_data_Gln3_sit)

exp_data_wt_norm = vcat(
    exp_data_Rib_norm[1], exp_data_cAMP_pde_norm[1], exp_data_cAMP_ras_norm[1],
    exp_data_Snf1_norm[1], exp_data_Gis1_norm[1], exp_data_Tre_norm[1], exp_data_Sch9_gtr1_2_norm[1],
     exp_data_Gcn4_norm[1], exp_data_Gln3_sit_norm[1]
     )

# Plotting the results
using Plots
using StatsPlots

default(dpi=300)
# default(ytickfont=7)
# default(xtickfont=7)
# default(titlefont=8)

######## Wild type ############
shifts_wt = vcat(Rib_shifts_wt_norm, cAMP_shifts_wt_norm, raw_shifts_wt)

shift_names_wt = ["\$\\mathrm{Rib}\$", "\$\\mathrm{cAMP}\$", "\$\\mathrm{cAMP}\$",
 "\$\\mathrm{Snf1}\$", "\$\\mathrm{Gis1}\$", "\$\\mathrm{Nth1}\$",
 "\$\\mathrm{Sch9_{gtr1-2}}\$", "\$\\mathrm{Gcn4}\$", "\$\\mathrm{Gln3}\$",
 "\$\\mathrm{Sch9_{gtr1-gtr2}}\$","\$\\mathrm{Sch9_{lst4-lst7}}\$"
```

```
]


shift_magnitude_wt = AbstractFloat[]
# Calculates the diffrence between steady state values and adding them to the vector
for i in range(1, length(shifts_wt))
    push!(shift_magnitude_wt, last.(shifts_wt[i]) - first.(shifts_wt[i]))
end

# Instead of using the highest and lowest value from the data and simulation, 0 and 100 is used as scaling for these,
# since the unit is percentage
exp_data_wt_special_raw = [2.0 => 73.0, 2.0 => 73.0] # Sch9_gtr1_gtr2, Sch9_lst4_lst7
exp_data_wt_special_norm = Minmaxnorm_shifts(exp_data_wt_special_raw, 0, 100.0)

exp_data_wt_raw = [
    150.0 => 600.0, 0.25 => 1.1, 0.25 => 1.1, 0.05 => 2.25, 700.0 => 35.0,
    100.0 => 25.0, 5.0 => 55.0, 7.3 => 100.0, 461.0 => 6248.0
    ]

# Combinde all experimental data
exp_data_wt = vcat(exp_data_wt_norm, exp_data_wt_special_norm)

 # Calculates the diffrence between the values for the experimental data
exp_shift_magnitude_wt = AbstractFloat[]
for i in range(1, length(exp_data_wt))
    push!(exp_shift_magnitude_wt, last.(exp_data_wt[i]) - first.(exp_data_wt[i]))
end

# Creats a vector with the diffrence from pre- and post-shift for both experimental data and simulations ready to plot
shift_values_wt = [shift_magnitude_wt exp_shift_magnitude_wt]

plot11 = groupedbar(
    shift_values_wt, group=repeat(["Simulering", "Experiment"], inner = length(shift_names_wt)), xlims=(-1.05, 1.05),
    yticks=(1:length(shift_names_wt), shift_names_wt), bar_position=:group, legend=:topleft, ytickfont=6,
    bar_width = 0.65, framestyle=:box, orientation=:horizontal
)

# title!("Vildtyp")
# xlabel!("Differens mellan station rv rden")
# ylabel="Signalmolekyler"
# display(plot11)

file_name = "perturb_wt"
savefig(plot11, pwd()*"/Results/Perturbation_reconstructed/"*file_name*".png")
savefig(plot11, pwd()*"/Results/Perturbation_reconstructed/"*file_name*".pdf")

####### Mutants #######
# Do the same as above, but for the mutants

# Scales the data according to the Minmaxnorm function
# cAMP
cAMP_shifts_mutant = [cAMP_mutant_pde_shift, cAMP_mutant_ras_shift]
cAMP_shifts_mutant_norm = Minmaxnorm_shifts(cAMP_shifts_mutant, min_cAMP, max_cAMP)

# Rib
Rib_shifts_mutant = [Rib_mutant_shift]
Rib_shifts_mutant_norm = Minmaxnorm_shifts(Rib_shifts_mutant, min_Rib, max_Rib)

# Placing the mutant simulations i order for the plot
raw_shifts_mutant = [Snf1_mutant_shift, Gis1_mutant_shift, Tre_mutant_shift, Sch9_gtr1_2_mutant_shift,
Gcn4_mutant_shift, Gln3_mutant_shift_sit, Sch9_gtr1_gtr2_mutant_shift, Sch9_lst4_lst7_mutant_shift]

shifts_mutant = vcat(Rib_shifts_mutant_norm, cAMP_shifts_mutant_norm, raw_shifts_mutant)

shift_names_mutant = ["\$\\mathrm{Rib_{sch9}}\$", "\$\\mathrm{cAMP_{pde1}}\$", "\$\\mathrm{cAMP_{ras1ras2bcy1}}\$",
 "\$\\mathrm{Snf1_{sak1tos3elm1}}\$", "\$\\mathrm{Gis1_{sch9}}\$", "\$\\mathrm{Nth1_{tpk1tpk2}}\$",
 "\$\\mathrm{Sch9_{gtr1-2}}\$", "\$\\mathrm{Gcn4_{gcn2}}\$", "\$\\mathrm{Gln3_{sit}}\$",
 "\$\\mathrm{Sch9_{gtr1-gtr2}}\$","\$\\mathrm{Sch9_{lst4-lst7}}\$"]

# Calculates the diffrence between steady state solutions, add them to an epty vector
shift_magnitude_mutant = AbstractFloat[]
for i in range(1, length(shifts_mutant))
    push!(shift_magnitude_mutant, last.(shifts_mutant[i]) - first.(shifts_mutant[i]))
end

# Instead of using the highest and lowest value from the data and simulation, 0 and 100 is used as scaling factors
exp_data_mutant_special_raw = [22.0 => 12.0, 1.0 => 19.0] # Sch9_gtr1_gtr2_mutant, # Sch9_lst4_lst7_mutant
exp_data_mutant_special_norm = Minmaxnorm_shifts(exp_data_mutant_special_raw, 0.0, 100.0)

exp_data_mutant_raw = [100.0 => 100.0, 0.25 => 3.8, 0.25 => 0.25, 0.01 => 0.01, 100.0 => 10.0,
50.0 => 25.0, 5.0 => 55.0, 11.0 => 38.0, 0.0 => 789.0]

exp_data_mutant_norm = vcat(
```

```
    exp_data_Rib_norm[2], exp_data_cAMP_pde_norm[2], exp_data_cAMP_ras_norm[2],
    exp_data_Snf1_norm[2], exp_data_Gis1_norm[2], exp_data_Tre_norm[2], exp_data_Sch9_gtr1_2_norm[2],
    exp_data_Gcn4_norm[2], exp_data_Gln3_sit_norm[2]
    )

exp_data_mutant = vcat(exp_data_mutant_norm, exp_data_mutant_special_norm)

# Calculates the diffrence between steady state values
exp_shift_magnitude_mutant = AbstractFloat[]
for i in range(1, length(exp_data_mutant))
    push!(exp_shift_magnitude_mutant, last.(exp_data_mutant[i]) - first.(exp_data_mutant[i]))
end

# Creats the final vector, then plot results
shift_values_mutant = [shift_magnitude_mutant exp_shift_magnitude_mutant]

plot12 = groupedbar(
    shift_values_mutant, group=repeat(["Simulering", "Experiment"], inner = length(shift_names_mutant)),
    yticks=(1:length(shift_names_mutant), shift_names_mutant), xlims=(-1.05, 1.05), bar_position=:group, bar_width=0.65,
    legend=:topleft, xlabel="Differens mellan station rv rden", ytickfont=6, orientation=:horizontal, framestyle=:box
)
title!("Mutanter")
# ylabel="Signalmolekyler"
# display(plot12)

file_name = "perturb_mut"
savefig(plot12, pwd()*"/Results/Perturbation_reconstructed/"*file_name*".png")
savefig(plot12, pwd()*"/Results/Perturbation_reconstructed/"*file_name*".pdf")
```

E.8 Kod till känslighetsanalys

E.8.1 Värmekarta över normer av känslighetsvektorer

Code/Sensitivity_Analysis/heat_map_norms.jl

```
using Plots
using Plots.PlotMeasures
include("sensitivity_analysis.jl")

"""
Plotting heat map of sensitivity vector norms.
"""

abs_senses = hcat(sens_vector_norms_J, sens_vector_norms_1, sens_vector_norms_2)

x_labels = p_var_lookup_table

y_labels = ["Jalihal","Parametervektor 1","Parametervektor 2"]
width = 1000
height = 300


Plots.heatmap(abs_senses',xticks = (1:81,x_labels),yticks = (1:3,y_labels),xrotation=90,
size = (width,height),bottom_margin = 75px)
```

E.8.2 Värmekarta över parvisa beroenden i parametervektorn

Code/Sensitivity_Analysis/heat_map_rel.jl

```
using Plots
using Plots.PlotMeasures
include("sensitivity_analysis.jl")

"""
Plotting heat map of pairwise sensitivity vector dependencies.
"""

x_labels = p_var_lookup_table
y_labels = x_labels
width = 1000
height = 1000

Plots.heatmap(rel_J,xticks = (1:81,x_labels),yticks = (1:81,y_labels),xrotation=90,
size = (width,height),bottom_margin = 75px)
Plots.heatmap(rel_1,xticks = (1:81,x_labels),yticks = (1:81,y_labels),xrotation=90,
size = (width,height),bottom_margin = 75px)
Plots.heatmap(rel_2,xticks = (1:81,x_labels),yticks = (1:81,y_labels),xrotation=90,
size = (width,height),bottom_margin = 75px)
```

E.8.3 Känslighetsanalys

Code/Sensitivity_Analysis/sensitivity_analysis.jl

```
using LinearAlgebra
include("../../Data/exp_data.jl")
include("../Model/ODE_methods.jl")
include("../Model/parameter_values.jl")
include("../Model/ODE_functions.jl")
include("shift_info_SA.jl")
include("../../Results/Parameter_values/Results_from_optimization.jl")

"""
    dy_dp(param,out_index,i)

    Compute the sensitivities of out signal out_index in the point param.
"""
function dy_dp(param,out_index,i)
    sim_param = param
    pre_p_const = p_const
    for (p_key,value) in all_pre_p_const_changes[i]
        pre_p_const[Get_index(p_const_lookup_table,p_key)] = value
    end
    post_p_const = p_const
    for (p_key,value) in all_post_p_const_changes[i]
        post_p_const[Get_index(p_const_lookup_table,p_key)] = value
    end
    for (p_key,value) in all_pre_p_var_changes[i]
        sim_param[Get_index(p_var_lookup_table,p_key)] = value
    end

    u0 = Steady_state_solver(last.(pre_p_const),sim_param,pre_shifts[i])

    sens_sol = sensitivity_solver(u0,post_shifts[i],all_tvals[i],last.(post_p_const),sim_param)
    if sens_sol.retcode != :Success
        throw(ErrorException("instability"))
    end

    sens_mat = extract_local_sensitivities(sens_sol)

    dy_dp = []
    for t in 1:length(all_tvals[i])
        push!(dy_dp, [])
        for p in 1:length(param)
            push!(dy_dp[t], sens_mat[2][p][out_index,t])
        end
    end

    return dy_dp
end

"""
    sens_mat(p)

    Compute the sensitivity matrix in the point p.
"""
function sens_mat(p)
    sens_mat = []
    for i in 1:length(out_indices)
        block = dy_dp(p,out_indices[i],i)
        append!(sens_mat,block)
    end
    sm = zeros(81,86)
    for i in 1:86
        sm[:,i] = sens_mat[i]
    end
    return sm
end

"""
    sum_sens_vecs(sens_mat)

    Compute the sensitivity vector norms and the sum of them for sensitivity matrix sens_mat.
"""
function sum_vec_sens(sens_mat)
    s = 0
    v = []
    for i in 1:length(sens_mat[:,1])
        n = norm(sens_mat[i,:])
        s += n
        append!(v,n)
    end
    return s,v
```

```
end


"""
    cos_rel_senses(sens_mat)

    Compute a matrix of cosines of the angles between all pairs of
    sensitivity vectors in sensitivity matrix sens_mat.
"""
function cos_rel_senses(sens_mat)
    m = zeros(81,81)
    for i in 1:81
        for j in 1:81
            n = norm(sens_mat[i,:])*norm(sens_mat[j,:])
            if n == 0.0
                m[i,j] = -1.0
            else
                m[i,j] = abs(dot(sens_mat[i,:],sens_mat[j,:])/n)
            end
        end
    end
    return m
end

sens_Jalihal = sens_mat(last.(p_var))
sens_alt1 = sens_mat(last.(p_var_alt_1))
sens_alt2 = sens_mat(last.(p_var_alt_2))

cond_nr_J = cond(sens_Jalihal)
cond_nr_1 = cond(sens_alt1)
cond_nr_2 = cond(sens_alt2)

rank_J = rank(sens_Jalihal)
rank_1 = rank(sens_alt1)
rank_2 = rank(sens_alt2)

abs_sens_J,sens_vector_norms_J = sum_vec_sens(sens_Jalihal)
abs_sens_1,sens_vector_norms_1 = sum_vec_sens(sens_alt1)
abs_sens_2,sens_vector_norms_2 = sum_vec_sens(sens_alt2)

rel_J = cos_rel_senses(sens_Jalihal)
rel_1 = cos_rel_senses(sens_alt1)
rel_2 = cos_rel_senses(sens_alt2)

out = "k_J = "*string(cond_nr_J)*"\n
k_1 = "*string(cond_nr_1)*"\n
k_2 = "*string(cond_nr_2)*"\n
r_J = "*string(rank_J)*"\n
r_1 = "*string(rank_1)*"\n
r_2 = "*string(rank_2)*"\n
sum_sens_vecs J = "*string(abs_sens_J)*"\n
sum_sens_vecs 1 = "*string(abs_sens_1)*"\n
sum_sens_vecs 2 = "*string(abs_sens_2)*"\n
sens_vecs_norms J = "*string(sens_vector_norms_J)*"\n
sens_vecs_norms 1 = "*string(sens_vector_norms_1)*"\n
sens_vecs_norms 2 = "*string(sens_vector_norms_2)*"\n
sens_mat_J = "*string(sens_Jalihal)*"\n
sens_mat_1 = "*string(sens_alt1)*"\n
sens_mat_2 = "*string(sens_alt2)*"\n
rel_sens_J = "*string(rel_J)*"\n
rel_sens_1 = "*string(rel_1)*"\n
rel_sens_2 = "*string(rel_2)*"\n
rel_sens_J = "*string(rel_J)*"\n
rel_sens_1 = "*string(rel_1)*"\n
rel_sens_2 = "*string(rel_2);

open("Results/Parameter_values/sensitivities_out.txt","w") do file
    write(file,out)
end
```

E.8.4 Skiftningsinformation till känslighetsanalys

Code/Sensitivity_Analysis/shift_info_SA.jl

```
include("../../Data/exp_data.jl")

# 1=Glutamine, Cyr1=2, 3=Ras, 4=EGO, 5=EGOGAP, 6=cAMP
# 7=PDE, 8=Sak 9=TORC1, 10=Snf1, 11=PKA, 12=Sch9
# 13=Gcn2, 14=Gcn4, 15=eIF, 16=Gln3, 17=Gln1, 18=Rtg13
# 19=Gis1, 20=Mig1, 21=Dot6 22=Tps1, 23=Trehalase
# 24=Protein, 25=Rib
```

```
out_indices = [10,6,6,12,12,12,12,12,20,25]

all_tvals = [t_Snf1, t_cAMP, t_sch9Delta_cAMP, t_Sch9P_glutamine_H, t_Sch9P_glutamine_L,
t_Sch9_gtr1Delta, t_Sch9_glucose_starve, t_Sch9_glucose_relief, t_Mig1_glucose_relief, t_Rib_rap]

pre_shifts = [(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 0.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)]

post_shifts = [(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0),
(Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0)]

all_pre_p_const_changes = [[],[],Dict("Sch9_T" => (Sch9_T => 0.0)),[],[],
Dict("EGO_T" => (EGO_T => 0.0)),[],[],[],[]]

all_post_p_const_changes = [[],[],Dict("Sch9_T" => (Sch9_T => 0.0)),[],[],[],[],[],[],
Dict("TORC1_T" => (TORC1_T => 0.0))]

all_pre_p_var_changes = [[],[],[],[],[],Dict("w_torc_ego" => 0.0,
"w_torc_egoin" => 0.0),[],[],[],[]]
```

E.9 Tidsseriesimuleringar

E.9.1 Glukostillsättning

Code/Time_course/Glucose_addition.jl

```
# Creates the plot for cAMP and Sch9 for the wildtype

using DifferentialEquations
using ModelingToolkit
using Plots

default(dpi = 300)
default(titlefontsize=13)

include("../Model/ODE_functions.jl")

include("../Model/parameter_values.jl")

# Pre-shift => ATP, Carbon = 0, glucose starvation
include("../Model/ODE_methods.jl")

# Returns steady state parameters that are used as u0 (start values for solving the ODE)
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))

# Post-shift, Glucose addition => ATP, Carbon = 1, glucose additon after starvation
tspan_cAMP = (0.0, 3.0) # [min]

# Solve the ODEs under the give time span and values on p
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan_cAMP, p_const, p_var)

include("../../Data/exp_data_norm.jl")

# Create plot for cAMP and Sch9 separately with experimental data for each
plot3 = scatter(t_cAMP, data_cAMP, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

plot!(sol, vars = cAMP, legend = false, color=color_jalihal, lw=2.0, show=false)
xlabel!("t [min]")
ylabel!("cAMP")
ylims!((0.0, 1.02))
xlims!((-0.02, last(tspan_cAMP)*1.02))
title!("Glukostillsättning")
display(plot3)

file_name = "cAMP_gluc_add_reconstr"
```

```
savefig(plot3, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot3, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")

# Separate solution for Sch9 since it needs a bigger time span
tspan_Sch9 = (0.0, 30.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan_Sch9, p_const, p_var)

# Plot the solutions for Sch9
plot4 = scatter(
    t_Sch9_glucose_relief, data_Sch9_glucose_relief, markersize = 4.5,
    markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )
plot!(sol, vars = Sch9, legend = false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("Sch9")
ylims!((0.0, 1.02))
xlims!((-0.3, last(tspan_Sch9)*1.02))
title!("Glukostillsättning")
display(plot4)

file_name = "Snf1_gluc_starve_reconstr"
savefig(plot4, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot4, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.2 Glukostillsättning, Mig1

Code/Time_course/Glucose_addition_Mig1.jl

```
# Creates the plot Mig1 under glucose addition with pre- and postshift variables

using DifferentialEquations
using ModelingToolkit
using Plots

default(titlefontsize=13)
default(dpi=300)

include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

# Pre-shift => ATP, Carbon = 0 which is glucose starvation
include("../Model/ODE_methods.jl")

# Return the steady state for the parameters with glucose satarvation
u0_SS = Steady_state_solver(p_const, p_var, (ATP => 0.0, Carbon => 0.0, Glutamine_ext => 1.0))


# Post-shift => ATP, Carbon = 1 which is clucos addition after sarvation
tspan = (0.0, 20.0) # [min]

# Solve the ODEs for the u0 from glucose addition over tspan
sol = ODE_solver(u0_SS, (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Includes the experimental data for Mig1 glucose relief and plots the data as a scatter-plot
include("../../Data/exp_data_norm.jl")
scatter(
    t_Mig1_glucose_relief, data_Mig1_glucose_relief, markersize = 4.5,
    markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )

# Plot the results for Mig1
plot1 = plot!(sol, vars=log(10, Mig1/(1e-5 + 1.0-Mig1)), color = color_jalihal, lw=2.0, legend=false)
xlabel!("t [min]")
ylabel!("log(nMig1/cMig1)")
ylims!((1.0, 1.6))
xticks!([0.0, 10.0, 20.0])
yticks!([1.0, 1.2, 1.4, 1.6])
title!("Glukostillsättning")
display(plot1)

file_name = "Mig1_gluc_add_reconstr"
savefig(plot1, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot1, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.3 Glukostillsättning Sch9Δ

Code/Time_course/Glucose_addition_Sch9.jl

```
# Creates the plot cAMP under glucose addition with pre- and postshift variables
```

```julia
using DifferentialEquations
using ModelingToolkit
using Plots

default(dpi = 300)
default(titlefontsize=13)

include("../Model/ODE_functions.jl")

include("../Model/parameter_values.jl")

# Mutant Sch9_Delta => Sch9_T = 0, Creates a mutant by setting total amount of the protein to zero
p_const[Get_index(p_const_lookup_table, "Sch9_T")] = Sch9_T => 0.0

# Pre-shift => ATP, Carbon = 0, which is glucose starvation
include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))

# Post-shift => ATP, Carbon = 1, which is glucose addition
tspan = (0.0, 3.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Makes a scatter-plot from the experimental data from cAMP
include("../../Data/exp_data_norm.jl")
scatter(t_sch9Delta_cAMP, data_sch9Delta_cAMP, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

# Plot results from simulation and experimental data
plot2 = plot!(sol, vars=cAMP, legend=false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("cAMP")
ylims!((0, 1.1))
xlims!((-0.03, last(tspan)*1.01))
title!("Glukostillsttning, Sch9\\Delta")
display(plot2)

file_name = "cAMP_Sch9delta_gluc_starve_reconstr"
savefig(plot2, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot2, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.4 Glukossvältning

Code/Time_course/Glucose_starvation.jl

```julia
# Creates the plots for Glucose starvation for Snf1 and Sch9 separtaley since they have different time spans

using DifferentialEquations
using ModelingToolkit
using Plots

default(dpi = 300)
default(titlefontsize=13)

include("../Model/ODE_functions.jl")

include("../Model/parameter_values.jl")

# Pre-shift => Carbon, ATP = 1, glucose abundence
include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))

# Post-shift, Glucose starvation => Carbon, ATP = 0
tspan_Snf1 = (0.0, 61.0) # [min]

# Solves the ODEs under glucose starvation
sol_Snf1 = ODE_solver(u0_SS, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Snf1, p_const, p_var)

include("../../Data/exp_data_norm.jl")

# Creates the plot for Snf1 with experimental data from another file
plot5 = scatter(t_Snf1, data_Snf1, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)
plot!(sol_Snf1, vars=Snf1, legend=false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("Snf1")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan_Snf1)*1.02))
title!("Glukossvltning")
display(plot5)

file_name = "Snf1_gluc_starve_reconstr"
savefig(plot5, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot5, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

```
tspan_Sch9 = (0.0, 30.0) # [min]

# Solves the ODEs for Sch9 in the same way as for Snf1 but with a different tspan
sol_sch9 = ODE_solver(u0_SS, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Sch9, p_const, p_var)

# Plot solution with exp data from a different file
plot6 = scatter(
    t_Sch9_glucose_starve, data_Sch9_glucose_starve, markersize = 4.5,
    markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )
plot!(sol_sch9, vars=Sch9, legend=false, color=color_jalihal)
xlabel!("t [min]")
ylabel!("Sch9")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan_Sch9)*1.02))
title!("Glukossvältning")
display(plot6)

file_name = "Sch9_gluc_starve_reconstr"
savefig(plot6, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot6, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.5 Kvävetillsättning (glutamin)

Code/Time_course/Glutamine_addition.jl

```
# Creates plots for high and low nitrogen addition after starvation, with Sch9 as readout

using DifferentialEquations
using ModelingToolkit
using Plots

default(dpi = 300)
default(titlefontsize=13)

include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

# Pre-shift => Glutamine_ext = 0, nitrogen starvation
include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0))

# Low glutamine => Glutamine_ext = 0.3
tspan = (0.0, 30.0) # [min]

# Solve the ODEs under additon of low amounts of glutamine as nitrogen source
sol_low = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan, p_const, p_var)

include("../../Data/exp_data_norm.jl")

# Creates the plot for low glutamine addition after starvation for Sch9
plot8 = scatter(
    t_Sch9P_glutamine_L, data_Sch9P_glutamine_L, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )

plot!(sol_low, vars = Sch9, legend = false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("Sch9")
ylims!((0, 1.1))
xlims!((-0.5, last(tspan)*1.02))
title!("Kvävetillsättning (låg, 0.3)")
display(plot8)

file_name = "Sch9_nit_add_low_reconstr"
savefig(plot8, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot8, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")

# High Glutamine => Glutamine_ext = 1.0
# Solves the same ODEs with high amounts of glutamine
sol_high = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

plot9 = scatter(
    t_Sch9P_glutamine_H, data_Sch9P_glutamine_H, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )
plot!(sol_high, vars = Sch9, legend = false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
xlims!((-0.5, last(tspan)*1.02))
ylabel!("Sch9")
title!("Kvävetillsättning (hög, 1.0)")
display(plot9)
```

```
file_name = "Sch9_nit_add_high_reconstr"
savefig(plot9, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot9, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.6 Kvävetillsättning, gtr1Δ

Code/Time_course/Glutamine_addition_gtr1.jl

```
# Creates plot for  Sch9 with a mutation gtr1_Delta under Glutamine addition

using DifferentialEquations
using ModelingToolkit
using Plots

default(dpi = 300)
default(titlefontsize=13)

include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")
include("../Model/ODE_methods.jl")

# Add the mutant gtr1_Delta by setting EGO_T = 0, w_torc_ego = 0, w_torc_egoin = 0
p_const[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 0.0
p_var[Get_index(p_var_lookup_table, "w_torc_ego")] = w_torc_ego => 0.0
p_var[Get_index(p_var_lookup_table, "w_torc_egoin")] = w_torc_egoin => 0.0

# Pre-shift => Glutamine_ext = 0, ATP, Carbon = 1, nitogen starvation

# Returns u0 (initial conditions) from the pre-shift steady state
u0_SS = Steady_state_solver(p_const, p_var, (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

# High glutamine => Glutamine_ext = 1, nitrogen addition
tspan = (0.0, 30.0) # [min]
# Solve the ODEs for the post-shift changes
sol = ODE_solver(u0_SS, (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0), tspan, p_const, p_var)

include("../../Data/exp_data_norm.jl")

# Plot the solution with experimental data from a separate file
plot7 = scatter(
    t_Sch9_gtr1Delta, data_Sch9_gtr1Delta, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8
    )

plot!(sol, vars=Sch9, legend=false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("Sch9")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan)*1.02))
title!("Kv vetills ttning , gtr1\\Delta")
display(plot7)

file_name = "Sch9_nit_add_reconstr"
savefig(plot7, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot7, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.7 Rapamycinbehandling

Code/Time_course/Rapamycin_treatment.jl

```
# Creates the plot for Rapamycin treatment where Ribi is used to make the plot

using DifferentialEquations
using ModelingToolkit
using Plots

include("../Model/ODE_functions.jl")

include("../Model/parameter_values.jl")
# Pre-shift => No change in parameter values

include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))

# Post-shift: Rapamycin treatment, which means TORC1_T = 0.0
p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 0.0

# Steady-state at starvation which is the initial values for Glutamine addition
tspan = (0.0, 90.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)
```

```
include("../../Data/exp_data_norm.jl")

plot10 = scatter(t_Rib_rap, data_Rib_rap, markersize = 4.5, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

# Rel. RPL32 mRNA corresponds to (Rib divided by the pre-sfhit steady state value for Rib)
plot!(sol, vars=Rib/(1e-3+u0_SS[Get_index(u_lookup_table, "Rib(t)")]), legend=false, color=color_jalihal, lw=2.0)
xlabel!("t [min]")
ylabel!("Rel. RPL32 mRNA") # Corresponds to Rib relativt steady state
title!("Rapamycinbehandling (TORC1 = 0)")
display(plot10)

file_name = "Rib_rap_treat_reconstr"
savefig(plot10, pwd()*"/Results/Time_course_reconstructed/pixel_images/"*file_name*".png")
savefig(plot10, pwd()*"/Results/Time_course_reconstructed/vector_images/"*file_name*".pdf")
```

E.9.8 Rita alla återskapade resultat

Code/Time_course/plot_all.jl

```
# Plot all solutions from the sperate files in one
include("Glucose_addition_Mig1.jl")
include("Glucose_addition_Sch9.jl")
include("Glucose_addition.jl")
include("Glucose_starvation.jl")
include("Glutamine_addition_gtr1.jl")
include("Glutamine_addition.jl")
include("Rapamycin_treatment.jl")
include("../Perturbation/perturbation_experiments.jl")
```

E.9.9 Rita alla resultat i en figur

Code/Time_course/plot_all_reconstr_in_subplot.jl

```
# Plots all the reconstructed plots in a subplot

using DifferentialEquations
using ModelingToolkit
using Plots

default(titlefontsize=5)
default(xtickfont=4)
default(ytickfont=4)
default(guidefont=6)
default(ylabelfontsize=4)
default(xlabelfontsize=5)
default(markersize=1.4)
default(lw=1.1)

##### Glucose addition Mig1 #####
include("../Model/ODE_functions.jl")
include("../Model/parameter_values.jl")

# Pre-shift => ATP, Carbon = 0 which is glucose starvation
include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (ATP => 0.0, Carbon => 0.0, Glutamine_ext => 1.0))


# Post-shift => ATP, Carbon = 1 which is clucos addition after sarvation
tspan = (0.0, 20.0) # [min]
sol = ODE_solver(u0_SS, (ATP => 1.0, Carbon => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Includes the experimental data for Mig1 glucos relief and presnet as a scatter-plot
include("../../Data/exp_data_norm.jl")
scatter(t_Mig1_glucose_relief, data_Mig1_glucose_relief, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

# Plot the results for Mig1
plot1 = plot!(sol, vars=log(10, Mig1/(1e-5 + 1.0-Mig1)), color = color_jalihal, legend=false) # 1e-5 ensures denom != 0
xlabel!("")
ylabel!("log(nMig1/cMig1)")
ylims!((1.0, 1.6))
xticks!([0.0, 10.0, 20.0])
yticks!([1.0, 1.2, 1.4, 1.6])
title1 = "Glukostillsttning"

##### Glucose addition Sch9_delta #####
include("../Model/parameter_values.jl")
# Mutant Sch9_Delta => Sch9_T = 0, Creates a mutant by deleting Sch9_T
p_const[Get_index(p_const_lookup_table, "Sch9_T")] = Sch9_T => 0.0

# Pre-shift => ATP, Carbon = 0, which is glucos starvation
```

```
include("../Model/ODE_methods.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))

# Post-shift => ATP, Carbon = 1, which is glucos addition
tspan = (0.0, 3.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Makes a scatter-plot from the experimental data from cAMP
scatter(t_sch9Delta_cAMP, data_sch9Delta_cAMP, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

# Plot results from simulation and experimental data
plot2 = plot!(sol, vars=cAMP, legend=false, color=color_jalihal)
xlabel!("")
ylabel!("cAMP")
ylims!((0, 1.1))
xlims!((-0.03, last(tspan)*1.01))
yticks!([0.0, 0.50, 1.00])
title2="Glukostillsttning, Sch9\\Delta"

##### Glucose addition #####
include("../Model/parameter_values.jl")

# Pre-shift => ATP, Carbon = 0, glucose starvation
# Returns steady stat parameters that are used as u0 (start values for solving the ODE)
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0))

# Post-shift, Glucose addition => ATP, Carbon = 1, glucose additon after starvation
tspan_cAMP = (0.0, 3.0) # [min]
# Solve the ODEs under the give time span and values on p
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan_cAMP, p_const, p_var)

include("../../Data/exp_data_norm.jl")

# Creat plot for cAMP and Sch9 separately with experimental data in each
plot3 = scatter(t_cAMP, data_cAMP, markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

plot!(sol, vars = cAMP, legend = false, color=color_jalihal, show=false)
xlabel!("")
ylabel!("cAMP")
ylims!((0.0, 1.1))
xlims!((-0.02, last(tspan_cAMP)*1.02))
yticks!([0.0, 0.50, 1.00])
title3="Glukostillsttning"

tspan_Sch9 = (0.0, 30.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan_Sch9, p_const, p_var)

# Plot the solutions for Sch9
plot4 = scatter(t_Sch9_glucose_relief, data_Sch9_glucose_relief,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)
plot!(sol, vars = Sch9, legend = false, color=color_jalihal)
xlabel!("")
ylabel!("Sch9")
ylims!((0.0, 1.1))
yticks!([0.0, 0.50, 1.00])
xlims!((-0.3, last(tspan_Sch9)*1.02))
title4="Glukostillsttning"

##### Glucose starvation #####
# Pre-shift => Carbon, ATP = 1, glucose abundance
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))

# Post-shift, Glucose starvation => Carbon, ATP = 0
tspan_Snf1 = (0.0, 61.0) # [min]
# Solves the ODEs with parametrs for Glucose starvation for Snf1
sol_Snf1 = ODE_solver(u0_SS, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Snf1, p_const, p_var)

# Creates the plot for Snf1 with experimental data from another file
plot5 = scatter(t_Snf1, data_Snf1,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)
plot!(sol_Snf1, vars=Snf1, legend=false, color=color_jalihal)
xlabel!("")
ylabel!("Snf1")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan_Snf1)*1.02))
yticks!([0.0, 0.50, 1.00])
xticks!([0.0, 20.0, 40.0, 60.0])
title5="Glukossvltning"

tspan_Sch9 = (0.0, 30.0) # [min]
# Solves the ODEs for Sch9 in the same way as Snf1 but for a diffrent tspan
sol_sch9 = ODE_solver(u0_SS, (Carbon => 0.0, ATP => 0.0, Glutamine_ext => 1.0), tspan_Sch9, p_const, p_var)

# Plot solution with ex data from a different file
plot6 = scatter(t_Sch9_glucose_starve, data_Sch9_glucose_starve,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)
plot!(sol_sch9, vars=Sch9, legend=false, color=color_jalihal)
```

```
xlabel!("")
ylabel!("Sch9")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan_Sch9)*1.02))
yticks!([0.0, 0.50, 1.00])
title6="Glukossv ltning"

##### Glutamine addition gtr1_Delta #####
p_const[Get_index(p_const_lookup_table, "EGO_T")] = EGO_T => 0.0
p_var[Get_index(p_var_lookup_table, "w_torc_ego")] = w_torc_ego => 0.0
p_var[Get_index(p_var_lookup_table, "w_torc_egoin")] = w_torc_egoin => 0.0

# Pre-shift => Glutamine_ext = 0, ATP, Carbon = 1, glutamine starvation
# Returns the u0 (startvalues) from the pre-shift
u0_SS = Steady_state_solver(p_const, p_var, (Glutamine_ext => 0.0, Carbon => 1.0, ATP => 1.0))

# High glutamine => Glutamine_ext = 1, glutamine addition
tspan = (0.0, 30.0) # [min]
# Solve the ODEs for the post-shift changes
sol = ODE_solver(u0_SS, (Glutamine_ext => 1.0, Carbon => 1.0, ATP => 1.0), tspan, p_const, p_var)

# Plot the solution with experimental data from a seperate file
plot7 = scatter(t_Sch9_gtr1Delta, data_Sch9_gtr1Delta,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

plot!(sol, vars=Sch9, legend=false, color=color_jalihal)
xlabel!("")
ylabel!("Sch9")
ylims!((0.0, 1.1))
xlims!((-0.5, last(tspan)*1.02))
yticks!([0.0, 0.50, 1.00])
title7="Kv vetills ttning , gtr1\\Delta"

##### Glutamine addition #####
include("../Model/parameter_values.jl")
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.0))

# Low glutamine => Glutamine_ext = 0.3
tspan = (0.0, 30.0) # [min]
# Solve the ODEs with low additon of glutamin as post shift
sol_low = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 0.3), tspan, p_const, p_var)

# Creates the plot for low glutamine addition after starvation for Sch9
plot8 = scatter(t_Sch9P_glutamine_L, data_Sch9P_glutamine_L,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

plot!(sol_low, vars = Sch9, legend = false, color=color_jalihal)
xlabel!("")
ylabel!("Sch9")
ylims!((0, 1.1))
xlims!((-0.5, last(tspan)*1.02))
yticks!([0.0, 0.50, 1.00])
title8="Kv vetills ttning (l g , 0.3)"

sol_high = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Plot the solution for high glutamine addition, with experimental data form seperate file
plot9 = scatter(t_Sch9P_glutamine_H, data_Sch9P_glutamine_H,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)
plot!(sol_high, vars = Sch9, legend = false, color=color_jalihal)
xlabel!("t [min]")
xlims!((-0.5, last(tspan)*1.02))
ylabel!("Sch9")
ylims!((0.0, 1.1))
yticks!([0.0, 0.50, 1.00])
title9="Kv vetills ttning (h g , 1.0)"

##### Rapamycin treatment #####
u0_SS = Steady_state_solver(p_const, p_var, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0))

# Post-shift: Rapamycin treatment which means TORC1_T = 0.0
p_const[Get_index(p_const_lookup_table, "TORC1_T")] = TORC1_T => 0.0

# Steady-state at starvation which is the initial values for Glutamine addition
tspan = (0.0, 90.0) # [min]
sol = ODE_solver(u0_SS, (Carbon => 1.0, ATP => 1.0, Glutamine_ext => 1.0), tspan, p_const, p_var)

# Rel. RPL32 mRNA corresponds to (Rib/steady-state value for Rib)

# Plot the results from the simulations and data from a seperate file
plot10 = scatter(t_Rib_rap, data_Rib_rap,  markercolor=RGB(0.35, 0.4, 1), markerstrokewidth=0.8)

plot!(sol, vars=Rib/(1e-3+u0_SS[Get_index(u_lookup_table, "Rib(t)")]), legend=false, color=color_jalihal, )
xlabel!("t [min]")
ylabel!("Rel. RPL32 mRNA") # Corresponds to Rib relative to it's steady-state value
yticks!([0.0, 0.50, 1.00])
ylims!((0.0, 1.1))
```

```
title10="Rapamycin (TORC1 = 0)"

title11="Vildtyp"
title12="Mutanter"

##### Perturbation #####
include("../Perturbation/perturbation_experiments.jl")

plot_list = [plot1, plot2, plot3, plot4, plot5, plot6, plot7, plot8, plot9, plot10, plot11, plot12]
l = @layout [
    grid(5,2){0.6w} grid(2,1)
]

titles=[title1, title2, title3, title4, title5, title6, title7, title8, title9, title10, title11, title12]

plot_all = plot(
    plot_list..., layout=l, legend = false,
    title=["\\($i\\) "*title for j in 1:1, (i, title) in enumerate(titles)]
)
display(plot_all)

file_name = "reconstructed_subplot"
savefig(plot_all, pwd()*"/Results/"*file_name*".png")
savefig(plot_all, pwd()*"/Results/"*file_name*".pdf")
```

E.10 Rita alla resultat

Code/plot_all_results.jl

```
include("Model_analysis/simulation_methods.jl")
sol_array_gluc_add = Generate_sol_array(index_glucose_addition, (0.0, 15.0))
plot_simulation_result(
    true, sol_array_gluc_add, cAMP, (0.0, 15.0), (0.0, 1.05), "Glukostillsättning", "cAMP_gluc_add", :right
    )
plot_simulation_result(
    true, sol_array_gluc_add, PKA, (0.0, 15.0), (0.0, 1.05), "Glukostillsättning", "PKA_gluc_add", :bottomright
    )

# Dessa bör köras sist, eftersom de ändrar många defaultvärden på Plots
include("Time_course/plot_all_reconstr_in_subplot.jl")
include("Model_analysis/plot_all_sims_in_subplot.jl")
```

E.11 Experimentell data

E.11.1 Rå experimentell data

Data/exp_data.jl

```
# Raw time series data. Normalization of the raw data is done in exp_data_norm
# Only manipulation done here is conversion of unit of time to minutes.

# Glucose starvation (Snf1)
t_Snf1=[0.39, 5.55, 30.56, 60.64] # min
raw_data_Snf1=[0.03, 0.26, 0.32, 0.302]

# Glucose addition (cAMP)
t_cAMP=[0.16, 15.27, 30.66, 44.88, 59.95, 74.84, 89.72, 104.2, 119.75, 149.39, 180.06]./60 # min
raw_data_cAMP=[0.079, 0.632, 0.822, 1.197, 1.227, 0.928, 0.617, 0.617, 0.632, 0.209, 0.282]

# Glucose addition, Sch9delta mutation (cAMP)
t_sch9Delta_cAMP=[0.15, 14.97, 30.09, 44.76, 59.71, 74.74, 89.51, 104.54, 119.59, 149.79, 180.28]./60 # min
raw_data_sch9Delta_cAMP=[0.052, 0.112, 0.67, 0.259, 0.484, 0.375, 0.264, 0.279, 0.301, 0.275, 0.259]

# High glutamine addition (Sch9)
t_Sch9P_glutamine_H=[0.06, 0.36, 0.73, 1.05, 3.07, 4.12, 5.06, 8.05, 11.03, 15.07, 30.06] # min
raw_data_Sch9P_glutamine_H=[5.92, 10.23, 34.29, 41.14, 40.59, 39.82, 32.99, 13.09, 17.25, 35.4, 52.57]

# Low glutamine addition (Sch9)
t_Sch9P_glutamine_L=[0.46, 1.09, 1.57, 2.04, 3.01, 4.05, 5.03, 8.05, 11.04, 15.01, 29.98] # min
raw_data_Sch9P_glutamine_L=[10.23, 49.55, 45.84, 27.66, 6.55, 5.49, 5.22, 5.27, 4.82, 4.89, 5.23]

# Glucose addition, gtr1Delta mutation (Sch9)
t_Sch9_gtr1Delta=[0.15, 0.7, 1.04, 1.51, 2.06, 3.05, 4.04, 5.06, 8.06, 11.0, 15.04, 30.02] # min
raw_data_Sch9_gtr1Delta=[7.96, 8.48, 8.48, 7.44, 7.44, 7.31, 7.44, 7.96, 9.64, 12.49, 25.7, 51.72]

# Glucose starvation (Sch9)
t_Sch9_glucose_starve=[0.0,2.5,5.0,15.0,30.0] # min
raw_data_Sch9_glucose_starve=[1.0,0.34,0.13,0.15,0.20]
```

```
# Glucose relief (Sch9)
t_Sch9_glucose_relief=[0.0,2.0,5.0,15.0,30.0] # min
raw_data_Sch9_glucose_relief=[0.20,0.63,0.92,0.91,1.0]

# Glucose relief (log(nMig1/cMig1))
t_Mig1_glucose_relief=[0.0, 67.0, 104.0, 130.0, 233.0, 334.0, 466.0, 606.0, 771.0, 919.0, 1088.0]./60 # min
raw_data_Mig1_glucose_relief=[1.070, 1.233, 1.363, 1.428, 1.490, 1.465, 1.432, 1.425, 1.416, 1.400, 1.402]

# Rapamycin treament (Rel. Rib)
t_Rib_rap=[0.0,15.0,30.0,60.0,90.0] # min
raw_data_Rib_rap=[1.00,0.43,0.19,0.09,0.07]
```

E.11.2 Normaliserad experimentell data

Data/exp_data_norm.jl

```
# The file explains how the experimental data is normalized in order to create the  time course plots
include("exp_data.jl")
include("../Code/Model/ODE_methods.jl")

color_data = RGB(0.35, 0.4, 1)

# For Mig1 there is no normalization done, due to the logarithmic scale in the data
data_Mig1_glucose_relief = raw_data_Mig1_glucose_relief

# data for cAMP uses min and max taken from all data for cAMP
data_sch9Delta_cAMP = Minmaxnorm(raw_data_sch9Delta_cAMP, 0.052, 1.227)
data_cAMP = Minmaxnorm(raw_data_cAMP, 0.052, 1.227)

# Snf1, sch9 uses the minmax function
data_Snf1 = Minmaxnorm(raw_data_Snf1)

data_Sch9_glucose_relief = Minmaxnorm(raw_data_Sch9_glucose_relief)
data_Sch9_glucose_starve = Minmaxnorm(raw_data_Sch9_glucose_starve)

# data for Sch9 with unit %phosphorylation has min and max taken from all data for Sch9
data_Sch9_gtr1Delta = Minmaxnorm(raw_data_Sch9_gtr1Delta, 4.82, 52.27)
data_Sch9P_glutamine_L = Minmaxnorm(raw_data_Sch9P_glutamine_L, 4.82, 52.57)
data_Sch9P_glutamine_H = Minmaxnorm(raw_data_Sch9P_glutamine_H, 4.82, 52.57)

# Rib uses minmaxnorm function to normalize data
data_Rib_rap = Minmaxnorm(raw_data_Rib_rap)
```

E.12 Parametervektorer som undersökts

Results/Parameter_values/plot_all_results.jl

```
include("../../Code/Model/ODE_functions.jl")
include("../../Code/Model/parameter_values.jl")

using Plots

#Alt 1
Color_alt_1 = RGB(249/255,212/255,35/255)
label_alt_1 = "Parametervektor 1"

p_var_alt_1 = [gammacyr => 18.322882081228986, w_cyr_glu => 3.6633999531005634, w_cyr_snf => 0.1285302238461551,
w_cyr => 0.6059280389339637, w_dot => 0.29859496480262826, w_dot_sch_pka => 0.26984973500579434,
w_ego => 0.14425669637415833, w_ego_basal => 0.006814200479995382, gammaego => 14.736736141864862,
w_ego_gap => 3.084279203114428, gammagap => 0.6663448945308855, w_gap_torc => 65.85992935027903,
w_gap_N => 34.2766745380838, gamma_gcn2 => 3.4453269070855477, w_gcn_torc => 3.306379713332646,
w_gcn => 0.23454636628836736, w_gcn4_gcn2_trna => 0.4912494787320278, w_gcn4 => 1.0432305907464907,
tRNA_sensitivity => 114.66813114510322, tRNA_total => 1.2406350409199844, w_gis => 2.050475783412456,
w_gis_pka => 7.217918860351613, w_gis_sch => 0.4425895833082018, w_gln1_gln3 => 1.395808403337666,
w_gln1 => 0.4670875108930438, gammagln1 => 0.06914367796507906, w_gln_snf => 1.5788171694107367,
w_gln_sit => 0.2404956660111842, w_gln3 => 1.603042453826415, gammagln3 => 0.040164304185710965,
k_acc_pro => 0.0013118240019047525, k_acc_glu => 0.05229669457960585, k_degr => 0.09599631569415247,
k_acc_nh4 => 0.0024859229170995766, w_mig_snf => 0.8162274919437411, w_mig => 7.815869018524566,
gamma_mig => 1.164435581740471, w_mig_pka => 6.336797432203219, gammapde => 0.20574815477230357,
w_pde_pka => 3.9405784158373662, w_pde => 0.9996420795693384, w_pka => 0.13971421470413295,
w_pka_sch9 => 41.48203870342469, w_pka_camp => 132.76026480145333, gammapka => 1.5340964099477878,
k_pr => 0.08020678151736915, w_ras_pka => 2.7963243302203162, gammaras => 1.1147866300004579,
w_ras_glu => 0.5929948563300336, w_ras => 0.012974111183497556, k_mRNA_degr => 0.07544679796135792,
k_transcription => 0.15269009808367842, w_rtg => 0.06984179043085359, w_rtg_torc => 0.6110744127682198,
w_sak => 0.21523434075177722, w_sak_pka => 0.8123596239993751, w_sch9_torc => 1.7534961165216625,
w_sch9 => 1.4098716830272526, gammasch9 => 2.9517438169783246, w_snf_sak => 1.132649229920907,
gammasnf => 0.7243545642175336, w_snf_glc => 1.4227670694144834, w_snf => 0.32464632554922157,
w_torc_egoin => 0.6320329127410665, w_torc_ego => 1.0293030153698601, w_torc => 0.33908276222471906,
w_torc_snf => 0.5627260805104367, w_torc_glut => 1.5539874103707414, gammator => 4.767784720571355,
w_tps_pka => 1.1821915364111435, gammatps => 1.0502044269195572, w_tps => 0.01421454474969244,
```

```
w_tre => 0.7169530050152036, gammatre => 0.7687385062810923, w_tre_pka => 8.412748392489899,
k_camp_cyr => 3.8238530492568787, k_camp_deg => 0.024634668345843064, k_camp_pde => 10.062551872385388,
w_eif_gcn2 => 0.418875925117752, gammaeif => 1.3510931647965105, w_eif => 2.9038537977882704]

p_conc_alt1 = vcat(p_const, p_var_alt_1)

#Alt2
Color_alt_2 = RGB(255/255,156/255,91/255)
label_alt_2 = "Parametervektor 2"

p_var_alt_2 = [gammacyr => 7.37813213215515, w_cyr_glu => 4.0997184519675685, w_cyr_snf => 0.15618803057097008,
w_cyr => 1.185464238821411, w_dot => 0.4111950209759053, w_dot_sch_pka => 0.1503256183734567,
w_ego => 0.17637124054254413, w_ego_basal => 0.01210236974294936, gammaego => 28.126358175661938,
w_ego_gap => 3.237868986459061, gammagap => 0.5612427739487568, w_gap_torc => 74.83866008764363,
w_gap_N => 11.874790200511052, gamma_gcn2 => 4.090316901581333, w_gcn_torc => 0.9896133346495836,
w_gcn => 0.11452929251704766, w_gcn4_gcn2_trna => 1.0378678579818184, w_gcn4 => 0.8080253187256702,
tRNA_sensitivity => 51.475300198167155, tRNA_total => 1.4165830774039265, w_gis => 0.9598913559028704,
w_gis_pka => 2.2531749789281474, w_gis_sch => 0.39962868796936074, w_gln1_gln3 => 0.42656401845447395,
w_gln1 => 0.23205314705787253, gammagln1 => 0.0748697315567821, w_gln_snf => 3.88332897305385,
w_gln_sit => 0.499946743845947, w_gln3 => 0.8291893830375339, gammagln3 => 0.05684170975136388,
k_acc_pro => 0.0009580425315539306, k_acc_glu => 0.0382698796749601, k_degr => 0.06109799788770438,
k_acc_nh4 => 0.0010633379425814037, w_mig_snf => 1.1045407402569298, w_mig => 10.682656304179545,
gamma_mig => 0.6301801949743072, w_mig_pka => 2.0702815313415317, gammapde => 0.24563706382038622,
w_pde_pka => 2.740532823678456, w_pde => 0.24578161145053465, w_pka => 0.07651914481742461,
w_pka_sch9 => 21.26180902280174, w_pka_camp => 92.33570214540178, gammapka => 2.0363410326848514,
k_pr => 0.036864618632475865, w_ras_pka => 1.7059571669418776, gammaras => 1.5457443238793085,
w_ras_glu => 0.4173960535668054, w_ras => 0.020403077111805366, k_mRNA_degr => 0.07321278881387466,
k_transcription => 0.30586390645462325, w_rtg => 0.11476698861371079, w_rtg_torc => 0.5694031108190232,
w_sak => 0.21376050520530604, w_sak_pka => 0.37341585131038835, w_sch9_torc => 1.8291717826570042,
w_sch9 => 0.6809949019873669, gammasch9 => 2.848067541686701, w_snf_sak => 1.6514510153607276,
gammasnf => 0.7872467094399845, w_snf_glc => 1.04913300835961, w_snf => 0.15981101514355836,
w_torc_egoin => 0.21074325679400258, w_torc_ego => 0.45325285966873063, w_torc => 0.3985465334961566,
w_torc_snf => 0.41082901282469547, w_torc_glut => 0.8002347014939227, gammator => 5.217018084103047,
w_tps_pka => 0.434029686353165 97, gammatps => 0.33650873776774165, w_tps => 0.027677950465241026,
w_tre => 1.0427720464617893, gammatre => 0.2491200987057159, w_tre_pka => 2.8906249963284982,
k_camp_cyr => 6.834825028889938, k_camp_deg => 0.052875098172279325, k_camp_pde => 11.19861069411881,
w_eif_gcn2 => 0.21385697180334579, gammaeif => 0.5726657082381318, w_eif => 3.718324521854869]

p_conc_alt2 = vcat(p_const, p_var_alt_2)

#Jalihal
Color_jalihal = RGB(59/255,129/255,131/255)
label_jalihal = "Jalihal"

p_var_jalihal = [gammacyr => 8.96, w_cyr_glu => 5.13, w_cyr_snf => 0.12, w_cyr => 1.35, # Cyr1
w_dot => 0.29, w_dot_sch_pka => 0.16, # Dot6
w_ego => 0.28, w_ego_basal => 0.01, gammaego => 50.66, w_ego_gap => 2.21, # EGO
gammagap => 0.56, w_gap_torc => 88.33, w_gap_N => 7.76, # EGOGAP
gamma_gcn2 => 4.71, w_gcn_torc => 1.29, w_gcn => 0.12, # Gcn2
w_gcn4_gcn2_trna => 1.53, w_gcn4 => 0.74, tRNA_sensitivity => 74.51, tRNA_total => 2.47, # Gcn4
w_gis => 1.3, w_gis_pka => 3.3, w_gis_sch => 0.84, #Gis1
w_gln1_gln3 => 0.52, w_gln1 => 0.22, gammagln1 => 0.06, # Gln1
w_gln_snf => 3.9, w_gln_sit => 0.86, w_gln3 => 0.64, gammagln3 => 0.08, # Gln3
k_acc_pro => 0.0, k_acc_glu => 0.05, k_degr => 0.09, k_acc_nh4 => 0.0, # Glutamine
w_mig_snf => 1.21, w_mig => 10.64, gamma_mig => 0.66, w_mig_pka => 2.31, # Mig1
gammapde => 0.28, w_pde_pka => 2.89, w_pde => 0.38, # PDE
w_pka => 0.06, w_pka_sch9 => 17.5, w_pka_camp => 102.11, gammapka => 2.68, # PKA
k_pr => 0.02, # Protein
w_ras_pka => 1.87, gammaras => 1.82, w_ras_glu => 0.21, w_ras => 0.02, # Ras
k_mRNA_degr => 0.07, k_transcription => 0.24, # Rib
w_rtg => 0.19, w_rtg_torc => 0.88, # Rtg13
w_sak => 0.21, w_sak_pka => 0.38, # Sak
w_sch9_torc => 1.96, w_sch9 => 0.57, gammasch9 => 4.63, # Sch9
w_snf_sak => 1.52, gammasnf => 0.82, w_snf_glc => 1.15, w_snf => 0.54, # Snf1
w_torc_egoin => 0.3, w_torc_ego => 0.88, w_torc => 0.54, w_torc_snf => 0.44, w_torc_glut => 0.86, gammator => 7.55, # TORC
w_tps_pka => 0.57, gammatps => 0.47, w_tps => 0.05, # Tps1
w_tre => 1.07, gammatre => 0.34, w_tre_pka => 3.07, # Trehalase
k_camp_cyr => 10.87, k_camp_deg => 0.08, k_camp_pde => 14.12, # cAMP
w_eif_gcn2 => 0.28, gammaeif => 0.47, w_eif => 3.73] # eIF

p_conc_jalihal = vcat(p_const, p_var_jalihal)
```