CHALMERS

GÖTEBORGS UNIVERSITET

Dynamisk Modellering av Glukosavkännande signalvägar i *Saccharomyces Cerevisiae*

Dynamic Modelling of Glucose Sensing Pathways in *Saccharomyces Cerevisiae*

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Miranda Carlsson

Jonatan Eklöv

Ola Rosengren

Anna Svensson Fehér

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Dynamisk Modellering av Glukosavkännande signalvägar i *Saccharomyces Cerevisiae*

Kandidatarbete i matematik inom civilingenjörsprogrammet Bioteknik vid Chalmers

Miranda Carlsson Jonatan Eklöv

Ola Rosengren Anna Svensson Fehér

Handledare: Sebastian Persson

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Förord

Följande kandidatarbete är genomfört av medlemmarna Miranda Carlsson, Jonatan Eklöv, Ola Rosengren och Anna Svensson Fehér. Vi studerar allesammans civilingenjörsprogrammet inom bioteknik på Chalmers tekniska högskola och valde att genomföra ett projekt på institutionen för matematiska vetenskaper. Projektets fokus har varit att applicera matematisk modellering för analys av glukosnätverk hos jästceller och har handletts av Sebastian Persson, nuvarande doktorand för Cvijovic lab. Vår förhoppning med projektet var att lära oss mer om ämnet, dess potential för forskningsrelaterade frågor samt bli bättre på programmering.

Tidigt under projektets gång delades arbetets in i två större delar. Det ena innefattade att formulera systemet med ekvationer, implementera dessa och ramverket för att kunna simulera dem i kod samt konstruera en kostnadsfunktion som kan minimeras. Den andra delen handlade om att formulera och implementera en kontinuerlig optimeringsalgoritm. Anna och Jonatan har haft ansvar för den förstnämnda delen medan Miranda och Ola har arbetat med den sistnämnda. Båda delar innefattar skrivande av kod och där har Ola och Jonatan tagit en större roll än de övriga i respektive halva. Trots uppdelningen har många av projektets delar arbetats på och diskuterats gemensamt, desto mer när slutskedet närmats för arbetet.

Uppdelningen för projektet speglas även i skrivprocessen av rapporten. I tabellen nedan framgår koncist vem som har skrivit vad av rapportinnehållet. Samtliga gruppmedlemmar har bidragit med feedback och förbättringar genom hela rapportskrivandet.

Tabell 1: Bidragsrapport - Skrivandet av rapporten

Vem	**Avsnitt**
Miranda	Förord, Abstract, Inledning, 3.1.1, 3.5.1, 3.6, 4.0, 4.1.0-3, 4.2, 5.2, 6, Appendix G.3-G.5
Jonatan	3.1.0 3.1.1, 3.1.3, 3.2.0, 3.2.4, 3.3, 3.4, 3.6 4.3.0, 4.3.4-5 5.0, 5.1-5, 6, Figur 1, Figur 5, Appendix A, B, C, G.1, G.2
Ola	Populärvetenskaplig presentation, 3.5.0, 3.5.2, 3.6, 4.1.2 4.0 4.1.4-5, 4.2, 4.3, 5.4, 5.6, 6, Figur 6, Appendix E-F, Appendix G.3-G.5
Anna	Förord, Populärvetenskaplig presentation, Sammandrag, 2, 3.1.2-3 3.2.0, 3.2.1-3, 4.3.1-3, 5.1, Figur 2-4, Appendix A, C, D, G.2

Utöver arbetet relaterat till rapporten har en loggbok förts över medlemmars enskilda bidrag och prestationer. Loggning har skett i form av en dagbok där arbetsbörda och aktiviteter presenterats i tabellform till examinatorerna. Dagboken har lämnats in veckovis och ansvaret för inlämningen har bestämts genom en turordning och därefter via ruljangs. Även en individuell, daglig tidslogg har förts av vardera medlem.

Populärvetenskaplig presentation

Hälsa och högt välbefinnande är en viktig aspekt att ta hänsyn till för hållbar utveckling globalt. I de globala målen framtagna av FNs utvecklingsprogram UNDP räknas god hälsa som en förutsättning för att kunna medverka i utvecklingen av samhället [1]. Trots att det skett framsteg inom vissa områden, förväntas antalet drabbade av olika metabolismrelaterade diagnoser som exempelvis fetma och diabetes typ 1 och 2 stiga [2]. Eftersom livslängden på jordens befolkning dessutom ökar, kan man även förvänta sig att åldersrelaterade sjukdomar kommer bli allt vanligare [3]. För att förhindra negativa hälsotrender krävs djupare förståelse kring vad som sker i människan på cellnivå under sjukdomstillstånd och åldrande. Därför är det intressant att studera cellulära processer som exempelvis glukosreglering och metabolism.

Till detta ändamål har jästceller (från *Saccharomyces cerevisiae*) visat sig vara av stor användning [4], då många av dess grundläggande funktioner liknar de som kan observeras i människoceller. Jästceller är välstuderade organismer samt lättodlade, vilket gör dem till lämpliga modellorganismer. Detta i kombination med etablerade analysmetoder gör att stora mängder information om jäst kan utvinnas experimentellt. För att effektivt kunna utnyttja denna information och kunna ta fram välgrundande slutsatser om jästecellers molekylära processer, krävs därför kompletterande metoder. Ett vanligt sätt att hantera och analysera denna sortens biologisk data är genom matematisk modellering. Matematisk modellering är synonymt med att applicera matematiska uttryck och ekvationer för att beskriva ett verkligt fenomen, exempelvis ett biologiskt system. Modellerna kan ge oss en insikt i hur komplexa system fungerar, exempelvis hur betydande komponenter varierar över tid [5].

Under projektet har modellering använts för att studera koncentrationsförändringar av proteiner, som påverkar hur mycket glukos jästceller tar upp. Nätverket har studerats främst med andledning att några av koncentrationerna för systemet saknat experimentell data. Modelleringen har genomförts genom att sätta upp ett matematiskt system för ingående komponenter. Systemet har därefter lösts, där lösningen visar hur systemet beter sig över tid. Modellens uppskattningar jämfördes därefter med experimentellt framtagen data för att se hur väl beräkningarna överensstämde med verkligheten. Deras differens har sedan försökt minimeras med hjälp av optimeringsalgoritmer.

Arbetet genomfördes med ambitionen att skapa en modell för jästcellens glukoshantering och ta fram resultat för detta, likt de som redan etablerats i artikeln *A quantitative model of glucose signaling in yeast reveals an incoherent feed forward loop leading to a specific, transient pulse of transcription* av S. Kuttykrishnan [6]. Till följd av lite data kunde artikelns modellkonstruktion inte återskapas på ett övertygande sätt. Metoden som användes lyckades inte bestämma parametrarnas värden, vilket gjorde direkt jämförelse mot artikeln omöjlig. Arbetet diskuterar istället metodval för utformningen av modellen och hur andra val hade kunnat motverka bristfälliga uppskattningar.

Sammandrag

Det blir allt vanligare att människor insjuknar i metaboliska sjukdomar som diabetes. För att motverka denna negativa hälsotrend krävs kunskap om cellulära processer för glukosreglering. Denna kunskap kan fås genom att studera interaktioner mellan substanser i komplexa biologiska system. I följande arbete var syftet att konstruera en dynamisk modell som beskriver nätverket för glukosreguleringen i *Saccharomyces cerevisiae*, utifrån en befintlig modell. Arbetets modell skapades genom att formulera ordinära differentialekvationer (ODE:er) som beskriver nätverkets koncentrationsförändringar. ODE-systemet löstes med en kostnadsfunktion, där en optimeringsalgoritm implementerades från grunden för att estimera okända parametrar i nätverket. Estimeringen lyckades inte på ett övertygande sätt bestämma parametervärdena och en diskussion fördes därför angående huruvida mängden tillgänglig data eller andra metodval hade kunnat förbättra modellen. Slutsatsen blev att större set data och en kraftigare optimeringsalgoritm troligtvis hade resulterat i mindre avvikande prediktioner.

Abstract

As the prevalence of metabolic diseases such as diabetes is increasing, more knowledge of the cellular processes governing glucose regulation is required to combat the trend. This understanding can be obtained by studying interactions of substances in complex biological systems. The aim of this project was to construct a dynamic model that describes the glucose regulation network in *Saccharomyces cerevisiae*, based on an existing model. The project's model was produced by formulating ordinary differential equations (ODEs) corresponding to concentration changes in the network. The system of ODEs was then solved by a cost function where an optimization algorithm was implemented to estimate unknown parameters in the network. The estimation failed in determining the parameter values in a convincing manner, and it was thereby discussed whether the amount of available data or other methodological approaches could have improved the model. It was concluded that the use of larger sets of data and a superior optimization algorithm most likely would have resulted in less deviant predictions.

Innehåll

1 Inledning

Metaboliska sjukdomsfall, såsom diagnoser av diabetes och fetma, ökar i världen [2, 7]. Till följd av det växer även intresset för kunskap om metabolism, då dessa sjukdomar påverkar både individ och samhälle kritiskt. Detta innebär exempelvis försämrade livssituationer och ökade vårdkostnader [8, 9]. Sockerarten glukos är en av kroppens främsta energikällor och hålls i en frisk individ under konstant reglering [10]. Denna reglering är en viktig funktion inom glukosmetabolism, vilket är den biologiska processen där kolhydrater bryts ner till glukos [11, 12]. Processen är vårt mest effektiva sätt att utvinna energi. Däremot kan olika glukosnivåer påverka immunsystemets celler kroniskt vilket kan medföra sjukdomar som diabetes, Alzheimers och cancer [10]. Förståelse kring hur cellers glukosreglering fungerar är därav av stor vikt för den mänskliga hälsan.

Från ett etiskt perspektiv uppstår svårigheter kring arbete på mänskliga cellstammar. Detta beror på att det är otydligt huruvida forskningen blir moraliskt försvarbar. Dessutom är cellerna komplexa och kostnadsmässigt ineffektiva att forska på, i jämförelse med andra alternativ som kan underhållas till en låg kostnad [4]. En bra kompromiss blir att använda välstuderade modellorganismer, vars glukosmetabolism kan jämföras med människan. En välanvänd organism är bagerijäst. Även efter en miljard års evolution har många av genfunktionerna bevarats mellan jästceller och människan, däribland metabolism av glukos [13]. Det är därför givande att studera biologiska system i jäst för att erhålla en förståelse för motsvarande processser i människan.

Genom förståelse för den här typen av system på en molekylär nivå möjliggörs manipulation av dem för att bidra till en ny önskad funktion [6]. Dessutom är den kinetiska responsen, dvs. hur snabbt cellen reagerar på en förändring, intressant att undersöka då det kan ha betydelse för cellens överlevnadsförmåga [14]. Om en cell reagerar för långsamt till ett ändrat näringstillstånd kan den bli utkonkurrerad av snabbare celler medan en cell som reagerar för snabbt kan förlora energi på responsen, vilket är ineffektivt.

Däremot kan reglerande nätverk som fysiologiskt svarar på externa och interna stimuli vara svårförstådda endast utifrån ett experimentellt tillvägagångssätt och intuitiva resonemang [15]. Detta genererar ett behov för matematisk modellering för att förstå systemen. Ofta används ordinära differentialekvationer (ODE:er) för att modellera dynamisk signalering i celler [16]. ODE:erna brukar formuleras genom att skriva om biokemiska interaktioner till hastighetsekvationer.

I det här arbetet undersöks en modell som beskriver koncentrationsförändringar av gener och protein intracellulärt i jästceller, givet olika externa sockerhalter. Modellen är skapad av Kuttykrishnnan m.fl. och beskrivs i artikeln *A quantitative model of glucose signaling in yeast reveals an incoherent feed forward loop leading to a specific, transient pulse of transcription* [6]. Målet med det här arbetet kommer vara att försöka återskapa modellen och undersöka om vi får liknande resultat. Det givna nätverket känner av glukos, som specifikt styr över gener som kodar för glukostransportörer i *Saccharomyces cerevisiae*. Systemet har två signalvägar varav den första svarar med induktion av generna till följd av ökad glukoshalt och den andra svarar med repression, vilket innebär att genen inte längre transkriberas. De två signalvägarna är sammanlänkade och högst dynamiska, vilket innebär att genuttrycken inte är konstanta och är beroende av externa och interna signaler.

2 Syfte

Syftet med följande projekt är att konstruera en dynamisk modell som beskriver den transkriptionella och protein baserade regleringen i glukosregleringsnätverket hos *S. cerevisiae* vid olika koncentrationer av glukos. Modellen skall bygga på Ordinära Differentialekvationer, ODE:er, som svarar mot koncentrationsförändringar av proteiner och gener i nätverket.

Slutmålet är att kunna jämföra den modell som tas fram under projektet med den befintliga som beskrivs i artikeln *A quantitative model of glucose signaling in yeast reveals an incoherent feed*

forward loop leading to a specific, transient pulse of transcription av Kuttykrishnan m.fl., där den experimentella datan som skall användas hämtas från artikeln [6]. Genom att jämföra modellerna kan det verifieras hur väl fungerande artikelns modell är för att uppskatta de okända parametrarna. Detta är relevant eftersom parametrarna påverkar resultat och slutsatser utifrån modellen, som i sin tur kan ligga i grund för hypoteser för frågeställningar inom området.

3 Teori

Inledningsvis ges en biologisk grund för att sätta arbetet i en bredare kontext samt möjliggöra förståelse av modellkonstruktion på flera nivåer. Därefter beskrivs den matematiska bakgrunden som används i konstruerandet och lösandet av optimeringsproblemet vars mål är att bestämma de okända parametervärdena.

Genomgående i texten betecknar x en skalär, $\boldsymbol{x}$ en vektor och $\boldsymbol{X}$ en matris. På liknande sätt är $f(x)$ en reellvärd funktion och $\boldsymbol{f}(x)$ en vektorvärd funktion.

3.1 Biologisk bakgrund

I följande stycken benämns protein, mRNA och gennamn i regel som akronymer. Som exempel beskrivs proteinet *Glucose-responsive transcription factor* som Rgt1, dess mRNA som mRGT1 och dess gen som RGT1. En mer detaljerad bild över generna i detta arbete fås via databasen *Saccharomyces Genome Database* [17].

En begreppslista över andra biologiska termer finns att tillgå i Appendix A.

3.1.1 Glukosreglering

Glukos och andra sockerarter är viktiga energikällor och byggnadsblock för alla organismer. Trots det kan höga glukoshalter vara problematiskt, som för människan där det kan bidra till exempelvis diabetes [18]. Reglering av halten glukos är därmed kritisk och det finns flertalet system för att styra den. På en större skala i människan bibehålls glukosnivå genom ett nätverk bestående av interaktioner mellan bukspottskörteln, levern, fettvävnad, muskler och hjärnan [10]. På cellulär nivå styrs det av en grupp proteiner. Dessa är intressanta att studera i mänskliga celler men också i andra organismer.

3.1.2 Jästen *Saccharomyces cerevisiae*

Bagerijäst (*Saccharomyces cerevisiae*) har flera fördelar som gör den till en lämplig modellorganism. Den har en inre cellstruktur som påminner om den som går att observera hos andra eukaryoter som djur och växter [4]. Likheten mellan de intracellulära organellerna gör att studier om jästen kan öka förståelsen för biologiska processer, även hos eukaryoter. Detta skulle kunna innefatta exempelvis metabolism, signalering och responsmekanismer.

3.1.3 Hexostransportörernas funktion och sammanfattning av glukosnätverket

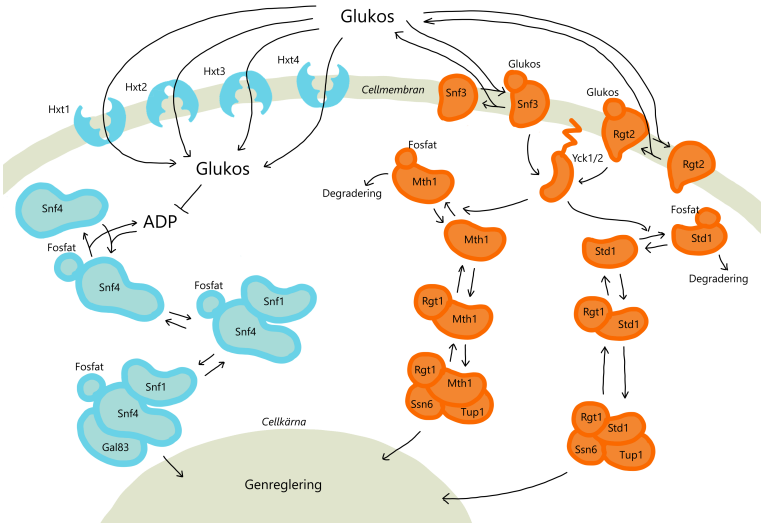
S. cerevisiae transporterar glukos och andra hexoser in i cellen med hjälp av hexostransportörer, benämnda Hxt1 till Hxt7 [19]. Glukosupptagningshastigheten är starkt korrelerad till antalet hexostransportörer, vilket innebär att glukosupptagningshastigheten kan styras väl genom att ändra transkriptionshastigheten av HXT-generna.

Det sju HXT-generna har liknande funktion men är aktiva och effektiva i olika glukoskoncentrationer med betydande överlapp [19]. Vid modellering räcker det därför att bara ett antal av generna inkluderas [6]. I Kuttykrishnans m.fl. artikel används HXT1-4 vilka även kommer användas i detta arbete.

Transkriptionshasitigheten av HXT-generna styrs av två kopplade nätverk. Induktionsnätverket ökar uttrycket av HXT-generna vid hög glukoskoncentration utanför cellerna och repressionsnätverket minskar uttrycket när energibehovet är lågt [6].

Induktionsnätverk mynnar ut i repressorn Rgt1 vilken nedreglerar uttryck av flertalet HXT-gener. För att fungera som repressor behöver Rgt1 binda de generella trankriptionsfaktorerna Ssn6 och Tup1 samt antingen Std1 eller Mth1. En effektiv degradering av Std1 och Mth1 initieras av fosforylering via kinaserna Yck1 eller Yck2 vilka blir aktiva när membranproteinerna Rgt2 och Snf3 har glukos bundna till sig.

Repressionsnätverket nedreglerar HXT-generna när cellen har ett lågt energibehov [6]. Med lågt energibehov menas att cellen har hög halt ATP vilket är ekvivalent med låg halt ADP. Det repressiva delnätverket utför nedreglering orsakad av repressorn Mig1. När glukosnedbrytning sker i delsystemet genom metabolism, hindras kinasproteinet Snf1 från att fosforylera Mig1. När Mig1 inte fosforyleras korrekt, blir konsekvensen att Mig1 nedreglerar bland annat de tre hexostransportörerna HXT2-4. Systemet i sin helhet illustreras i Figur 1.



Figur 1: Schematisk överblick av *Saccharomyces cerevisiaes* glukosregleringsnätverk. Till vänster (blått) är repressionsnätverket vilket nedreglerar uttrycket av Hxt-generna och till höger (orange) är induktionsnätverket vilket istället uppreglerar det. Pilar visar generellt övergångar mellan olika ämnen, men representerar från Yck1/2 hastighetspåverkan för reaktioner. Pilen från glukos till ADP visar att hög intracellulär glukoshalt minskar mängden ADP.

3.2 Dynamisk modellering av intracellulära processer

Inom biologi är det viktigt att kunna analysera system bestående av metaboliter, proteiner och gener, här gemensamt kallat substanser, via matematiska modeller. Detta är viktigt eftersom experimentella tillvägagångssätt är dyra, svåra och begränsade i vad de kan göra. Matematisk modellering möjliggör att kunna studera reaktioner och fenomen som inte går att undersöka explicit samt konsekvenserna av komplexa interaktioner. För biologiska system som ändras över tid görs detta lämpligen med kopplade ordinära differentialekvationer [20]. Dessa är normalt på formen

$$\frac{\mathrm{d}[X]}{\mathrm{d}t} = \text{Produktion av X} - \text{Förbrukning av X}, \tag{1}$$

där X är en substans i systemet. Lösningar av dessa system, som i regel görs via numerisk väg, kan ge en insikt om hur systemet fungerar och möjliggör förutsägelse av dess beteende.

Samtliga biokemiska reaktioner, studerade på nivån av endast ett reaktionssteg, kan uttryckas med hjälp av massverkans lag [20]. Den säger att hastigheten för en reaktion är proportionell mot reaktanternas koncentration, alltså koncentrationerna av de substanser som förbrukas. Som exempel kan reaktionen där två X slås ihop med varandra och bildar Y formuleras enligt,

$$r = k[X]^2, \tag{2}$$

där r är reaktionshastigheten, k är hastighetskonstanten och $[X]$ är koncentrationen av X [21]. Lagens universalitet gör den mycket användbar och därför används den i vår modell för degradering och translation.

Att modulera alla reaktioner med massverkans lag är däremot opraktiskt då vi måste känna till exakta reaktionsflödet samt hastighetskonstanten för varje steg i alla reaktioner. Därför används ofta olika förenklande antaganden. I *S. cerevisiaes* glukosnätverk finns fyra sorters biokemiska reaktioner, utöver redan presenterade degradering och translation som vår modell uppmärksammar [6]. I följande avsnitt presenteras *Michealis-Mentens ekvation* och *Hills ekvation* som förenklande ekvationer samt *Quasi steady state* (*QSSA*) som förenklande antagande, vilka används till att formulera en användbar modell.

3.2.1 Michaelis Menten ekvationen - förenklande modell för enzym-substratreaktioner

Michaelis-Mentens ekvation är viktig för forskning inom det biokemiska fältet [22]. Ekvationen togs fram genom forskning på enzymet *Invertas*, i syfte att beskriva dess enzymatiska aktivitet. Ekvationen blir ett sätt att förenkla följande enzym-substratreaktion

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} ES \xrightarrow{k_2} E + P,$$

där S symboliserar substratet som reversibelt binds in till ett enzym E:s aktiva säte [23]. De två enheterna bildar tillsammans ett komplex som i schemat benämns *ES*. Vid reaktionen bildas sedan P och enzym E frisläppas igen. Förloppet har tre tillhörande hastighetskonstanter. k_1 är hastighetskonstanten för uppkomsten av komplexet ES från totalmängden substrat och det obundna enzymet [22]. Processen som resulterar i komplexet, gör att substratet delas in i kategorierna bundet och obundet substrat. Genom att anta ett substratöverskott för reaktionen kan dock andelen bundet substrat försummas och totalmängden substrat anses konstant. För reaktionen motsvarar sedan k_{-1} hastighetskonstanten för det komplexbundna substratet och enzymet att återgå till sina initialtillstånd E och S. k_2 svarar slutligen mot den katalytiska hastighetskonstanten för den irreversibla reaktionen att omvandla komplexet till produkt och obundet enzym.

För reaktionen kan Michaelis-Mentens ekvation presenteras enligt följande

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{V_{max} \cdot [S]}{K_s + [S]}\ , \tag{3}$$

där V_{max} indikerar maximala hastigheten för allt enzym att bli bundet som komplex, K_m Michaelis-Menten-konstanten och slutligen S koncentrationen för substratet [22]. Uttrycket tas fram genom härledningen i Appendix D och sker under antagandena att reaktionen har ett substratöverskott

samt att delreaktionen för komplexbildningen sker under stationärt tillstånd [24]. I projektet används ekvationen för att härleda fram uttryck för HXT-genernas transport av glukos över cellmembranet.

3.2.2 Modell för genreglering - Hills ekvation

Ekvationer liknande Michaelis-Mentens ekvation används i många sammanhang inom biologi och medicin. En bred generalisering av dessa är Hills ekvation, presenterad nedan mellan de två generella variablerna x och y [25]:

$$y = \frac{y_{max} \cdot x^{\alpha}}{c^{\alpha} + x^{\alpha}}. \tag{4}$$

Ekvationen har utöver det tre generiska parametrar; y_{max} , c och α.

I projektet studeras ett nätverk med gener som har direkt samverkan för genregleringen. Eftersom transkription resulterar i produktion av mRNA kan ett systems tillstånd beskrivas genom dess koncentration vid en tid t i systemet [26]. För analys där man betraktar celler med identiska mekanismer för uttryck av en samstämmig uppsättning gener, kan därför dess mRNA-produktion uttryckas genom en första ordningens hastighetsekvation:

$$\frac{\mathrm{d}[mRNA]}{\mathrm{d}t} = V_{max} \cdot c - k[mRNA]. \tag{5}$$

I ekvationen beskriver V_{max} den producerade DNA-koncentrationen per sekund givet att samtliga gener bidrar till transkriptionen. k symboliserar degraderingshastigheten för mRNAt medan c är den andel av generna som utför transkriptionen. I djupare detalj kan c uttryckas

$$c = \left(1 - \prod_{i \in R^{+}} \frac{1}{(1+\theta_i)^{S_i}}\right) \prod_{i \in R^{-}} \frac{1}{(1+\theta_i)^{S_i}}, \tag{6}$$

där R^{+} och R^{-} är de gener som verkar för produktion av transkriptionsfaktorerna som blir aktivatorer respektive repressorer för transkriptionen [26]. θ_i motsvarar styrkan som transkriptionsfaktorn i binder till genen, vilket kan likställas med hur effektiv repressionen eller aktiveringen är. S_i är antalet platser faktorn kan binda.

Under förutsättningen att nätverket uteslutande har repressorer och att dessa enbart besitter en lämplig bindingsplats kan uttrycket förkortas till:

$$\frac{\mathrm{d}[mRNA]}{\mathrm{d}t} = V_{max} \prod_{i=1}^{n} \frac{1}{1+\theta_i[R_i]} - k[mRNA]. \tag{7}$$

Uttrycket används för att kunna modellera repressorernas inverkan på transkriptionen av de diverse mRNAn som finns i glukosnätverket.

3.2.3 Quasi steady state - förenklande hastighetsantagande för snabba reaktioner

Hastigheten r för biokemiska reaktioner kan i hög grad variera beroende på vilken sorts reaktion som studeras. Skillnader i tidsskalor för reaktionerna gör simuleringen för systemets beteende över

tid väldigt svårberäknad utan att tillämpa förenklingar [20]. Därför används Quasi steady-state antagandet *QSSA*. Det antagandet säger är att beroende variabler för systemet kan antas vara i stationärt tillstånd med avseende på andra systemvariabler, givet att reaktionen för den berörda variabeln sker under hög hastighet [27]. Antagandet innebär att en differentialekvation ersätts med en jämnviktsreaktion. I arbetet tillämpas *QSSA* för bindningar mellan två proteinmolekyler samt för fosforylering. I systemet är de beroende variablerna koncentrationer.

3.3 Modellkonstruktion

Informationen presenterad i föregående avsnitt är en grund på vilken det går att utveckla en numeriskt lösbar ODE-modell. Denna kommer att ha formen

$$\boldsymbol{C}_t = \mathbf{ODEmodell}(\boldsymbol{C}_0, \boldsymbol{\theta},\ t), \tag{8}$$

och kan bestämma slutkoncentrationerna $\boldsymbol{C}_t$ vid tiden t utifrån parametervärdena $\boldsymbol{\theta}$ och initialkoncentrationerna $\boldsymbol{C}_0$. Ett stort problem är dock att det i många modeller existerar parametrar $\boldsymbol{\theta}_e$ vilka inte kan bestämmas experimentellt utan måste estimeras. Hur detta görs presenteras i kommande avsnitt.

3.4 Härledning av minimeringsproblem för okända parametrar

Värdet på parametrarna $\boldsymbol{\theta}_e$, introducerade i avsnittet ovan, ska bestämmas sådant att modellen beter sig så likt som möjligt som den experimentella datan. För att göra det introduceras här ett minimeringsproblem vilket löser detta. Avsnittet är baserat på *Notes for Predictive Modeling* av E. García-Portugués och är anpassat till arbetets sammanhang [28].

Parametrarna med okända värden $\boldsymbol{\theta}_e$ i en modell går att estimera om det finns data över hur modellen beter sig. De okända parametrarna betraktas lämpligen som stokastiska variabler där ett experiment är en sampling. Sannolikheten för att observera koncentrationerna $\boldsymbol{y} = \{y_1, \ldots, y_i\}$ givet initialkoncentrationenerna $\boldsymbol{x}$ och parametervärdena $\boldsymbol{\theta}_e$ kan formuleras enligt följande, om vi antar normalfördelningar med oberoende fel och samma varians [28].

$$\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\mathbf{ODEmodell}(\boldsymbol{x}, \boldsymbol{\theta}_e), \sigma) \tag{9}$$

Observera att $\boldsymbol{x}$ kan innefatta andra substanser än $\boldsymbol{y}$ och att $\mathcal{N}(f(\boldsymbol{x}), \sigma)$ är en normalfördelning med standardavvikelsen σ och väntevärde enligt funktionen $f(\boldsymbol{x})$.

Om vi studerar j observationer $\boldsymbol{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j\}$ med initialvillkor $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j\}$ kan vi formulera ett uttryck för den totala sannolikheten enligt följande

$$\boldsymbol{Y}|\boldsymbol{X} \sim \prod_{i=0}^{j} \mathcal{N}(\mathbf{ODEmodell}(\boldsymbol{x}_i, \boldsymbol{\theta}_e), \sigma)\,. \tag{10}$$

Vi vill att vår modell stämmer överens med observerad data så bra som möjligt. En mer exakt formulering är att vi vill finna $\boldsymbol{\theta}_e$ sådant att en stokastisk variant av vår modell ska ha så stor sannolikhet som möjligt för observerad data. Därför definierar vi likelihoodfunktionen $\mathcal{L}$ som värdet på täthetsfunktionen för $\boldsymbol{Y}|\boldsymbol{X}$ med parametervärderna $\boldsymbol{\theta}_e$:

$$\mathcal{L}(\theta) := \prod_{i=0}^{j} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(\boldsymbol{y}_i - \mathbf{ODEmodell}(\boldsymbol{x}_i,\, \boldsymbol{\theta}_e))^2}. \tag{11}$$

Vår maximum-likelihood estimation fås genom att finna maximum av $\mathcal{L}(\boldsymbol{\theta}_e)$, vilket är ekvivalent med att finna maximum av dess logaritm $\ell(\boldsymbol{\theta}_e)$, då logaritmen är en monotont ökande funktion. Detta tar bort problemet med de små värden likelihood funktionen normalt har. $\mathcal{L}(\boldsymbol{\theta}_e)$ får uttrycket

$$\ell(\boldsymbol{\theta}_e) = -\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\sum_{i=0}^{j}\left(\boldsymbol{y}_i - \mathbf{ODEmodell}(\boldsymbol{x}_i,\, \boldsymbol{\theta}_e)\right)^2. \tag{12}$$

Konstanterna påverkar inte optimeringen, förutom för proportionalitetskonstanternas tecken, vilket möjliggör att optimeringsproblemet att minimera kostnadsfunktionen kan formuleras som

$$\min_{\boldsymbol{\theta}_e} \sum_{i=0}^{j}\left(\boldsymbol{y}_i - \mathbf{ODEmodell}(\boldsymbol{x}_i,\, \boldsymbol{\theta}_e)\right)^2. \tag{13}$$

Detta problem saknar generell analytisk lösning då beräknandet av kostnadsfunktionen involverar differentialekvationer som saknar explicita lösningar. Därmed krävs en kontinuerlig optimeringsalgoritm.

3.5 Kontinuerlig optimering

För en funktion, så som kostnadsfunktionen (13) presenterad i Avsnitt 3.4, finns numeriska metoder för att finna dess minimum. Två vanliga metoder för att optimera kontinuerliga funktioner är Newtons metod och Steepest descent [29].

3.5.1 Steepest descent

Steepest descent-metoden är en av de äldsta och enklaste metoderna för att minimera en generell olinjär funktion. Strategin går ut på att funktionsvärdet minskar om vi går i riktning av den negativa gradienten [30]. Metoden kan allmänt beskrivas enligt,

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_i \cdot \big(-\nabla f(\boldsymbol{\theta}_i)\big), \tag{14}$$

där α_i är steglängden och gradienten av funktionen betecknas av $\nabla f(\boldsymbol{\theta}_i)$ [31]. $\boldsymbol{\theta}_i$ är värdet på den punkt som startas ifrån och $\boldsymbol{\theta}_{i+1}$ är det nya $\boldsymbol{\theta}$-värdet. En betydande nackdel med Steepest descent är att metoden har en mycket långsam konvergeringshastighet [30]. Utmaningen blir då att finna en passande steglängd så att konvergeringshastigheten blir så bra som möjligt. Även om metoden är långsam är en stor fördel att den alltid garanterar en nedåtgående riktning.

3.5.2 Newtons metod

Newtons metod utnyttjar, till skillnad från Steepest descent, andra ordningens approximation av funktionen vid en punkt för att hitta en nedåtstigande riktning [29]. Algoritmen Newtons metod använder är följande:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{x}_i - \alpha_i \cdot \left(\nabla^2 f(\boldsymbol{\theta}_i)\right)^{-1} \cdot \nabla f(\boldsymbol{\theta}_i), \tag{15}$$

där $\nabla^2 f(\boldsymbol{\theta}) \in \mathbb{R}^{k \times k}$ är hessianen för funktionen f, $\nabla f(\boldsymbol{\theta}) \in \mathbb{R}^k$ är gradienten av funktionen f och k är längden av vektorn $\boldsymbol{\theta}$. Härledningen hittas i Appendix E.

För minimering fungerar metoden endast om hessianen är positivt definit för varje punkt, då en negativt definit hessian medför att algoritmen kommer gå mot ett lokalt maximum. Om detta är fallet för en stor del av parameterrummet till funktionen blir därför Steepest descent en lämpligare metod att använda.

För Steepest decent och Newtons metod krävs värden på gradienten, respektive hessianen. En mycket användbar metod för att beräkna dem är automatisk differentiering vilken presenteras i Appendix B. Med dessa verktyg kan minimeringsproblemet (13) lösas. Dock ger denna lösning ingen insikt i hur pålitliga de estimerade parametrarna från optimeringen är. Till detta behövs en identifieringsanalys.

3.6 Identifierbarhetsanalys

När parametrarna i stora biologiska modeller är kalibrerade för att bäst anpassas till den experimentella datan, kan mer och viktig information av modellen ges genom identifierbarhetsanalys [32]. Identifierbarhetsanalys av parametrar bedömer om det är teoretiskt möjligt att estimera specifika parametrar, och om dessa inte går att estimera kallas de icke-identifierbara [33].

Profile likelihood är en effektiv identifierbarhetsanalysmetod för att upptäcka dessa icke-identifierbara parametrar. Dessutom kan metoden hjälpa oss att förstå olika typer av icke-identifierbarhet [32]. Metoden går ut på att en parameter $\theta_i \in \boldsymbol{\theta}_e$ sätts till ett konstant värde som är förskjutet från det optimerade värdet. Därefter optimeras kostnadsfunktionen $\chi^2(\boldsymbol{\theta})$ med avseende på resterande parametrar. Detta kan beskrivas enligt

$$\chi^2_{PL} = \min_{\theta_{j \neq i}} \left[\chi^2(\boldsymbol{\theta})\right], \tag{16}$$

där χ^2_{PL} representerar profile likelihood-funktionen [32]. Detta upprepas för värden både över och under estimeringen av θ_i så att profilering fås kring optimum av θ_i. Detta erhåller hur kostnadsfunktionen varierar kring θ_i vilket innebär att vi kan erhålla ett konfidensintervall för θ_i. Vi kan nu definiera två olika typer av icke-identifierbara parametrar.

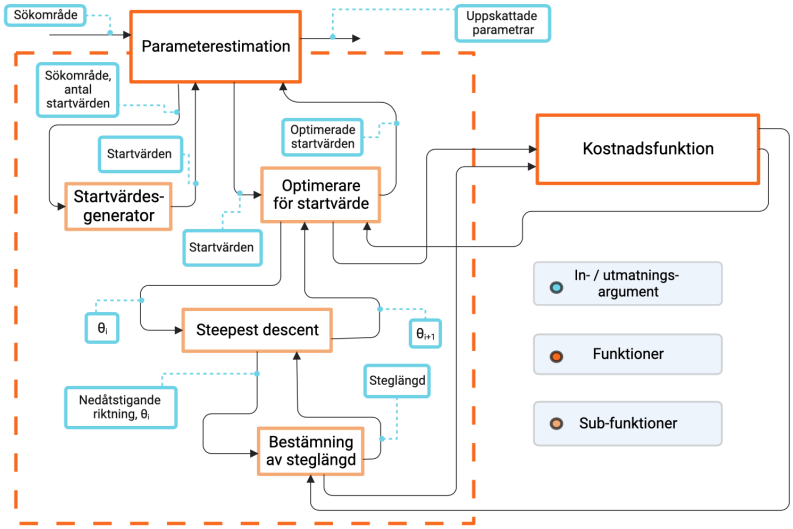
En parameter kan dels vara strukturellt icke-identifierbar, vilket innebär att en ändring av parametern inte nödvändigtvis förändrar hur modellen ser ut då en ekvivalent förändring kan erhållas av att ändra andra parametrar [34]. Strukturellt icke-identifierbara parametrar kan inte definieras alls då de har ett oändligt konfidensintervall $[-\infty, \infty]$ och även är oberoende av hur mycket data som samlats in [32].

Parameterar kan också vara praktiskt icke-identifierbar, vilket sker då det inte finns tillräckligt med, eller för dålig kvalité, av data för att bestämma den [32]. Till skillnad från strukturellt icke-identifierbara parametrar är en parameter praktiskt icke-identifierbar om konfidensregionen för parametern är oändligt utsträckt i minst en riktning.

4 Metod

Arbetsgången i projektet har bland annat innefattat att förstå och analysera modellen från artikeln av Kuttykrishnan m.fl., att formulera ODE:er och att optimera parametervärden. En stor del av arbetet har gått till att strukturera och skriva kod som illustreras schematiskt i Figur 2 och 3, vilket presenteras mer detaljerat i kommande delavsnitt.

All kod är skriven i programmeringsspråket Julia (version 1.8.5) [35]. Källkoden finns att erhålla i Appendix G samt på GitHub via `https://github.com/oerp0315/kandidatarbete`.



Figur 2: Översiktlig struktur över parameterestiomationen samt hur den interagerar med kostnadsfunktionen. Varje funktion tar in pilarna till vänster och returnerar pilarna åt höger. Pilarna under en funktion går till funktioner som kallas på, när den evalueras. När kostnadsfunktionen kallas på beräknas antingen dess funktionsvärde eller dess gradient.

4.1 Optimering av modellens parametrar

Parameterestimationen har gennomförts genom att finna minimum av kostnadsfunktionen. Denna process visualiseras i Figur 2. Utifrån startvärden inom ett visst område har Steepest descentmetoden använts för att hitta en punkt med lägre funktionsvärde, varav steglängden har anpassats för vardera steg som tagits. Beräkningarna för optimeringen har skett i logaritmerad skala.

4.1.1 Användning av Steepest descentmetoden

För att optimera modellens parametervärden användes Steepest descentmetoden. Detta beror på att vi fann att en stor majoritet av punkterna i funktionen inte hade en hessian som var positivt definit, vilket diskuteras i Avsnitt 3.5.

För att hitta nästa punkt i en nedåtgående riktning med användning av metoden, utgicks (14) ifrån. Den nya punkten $\boldsymbol{\theta}_{i+1}$ hittades med

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \cdot \boldsymbol{d}, \tag{17}$$

där $\boldsymbol{\theta}_i$ representerar den nuvarande punkten, α steglängden och $\boldsymbol{d}$ riktningen för steget. Riktningen

beräknades genom att dividera den negativa gradienten med normen av gradienten vilket gav oss en enhetsvektor i den mest nedåtgående riktningen. Om den nya punkten befann sig utanför de satta gränserna projekterades den tillbaka till gränsvärdet.

För att beräkna gradienten användes ForwardDiff.jl (version 0.10.35) i Julia [36].

4.1.2 Bestämning av steglängd

Ett steg i en punkts nedåtgående riktning garanterar inte ett mindre funktionsvärde då en minimipunkt kan passeras. För att undvika ett för långt steg halverades steglängden för varje iteration tills att kravet av ett mindre funktionsvärde för den nya punkten uppfylldes. Steglängden α hade ett startvärde på 1.0. Om ingen av de testade steglängderna uppfyllt det angivna kravet efter 50 iterationer avbröts försöket.

4.1.3 Termineringskrav

Det maximala antalet steg som tilläts per startvärde var 1000 och för att undvika att många onödiga steg togs användes termineringskrav som kunde avbryta optimeringen tidigare. De tre termineringskraven som implementerades är följande:

$$||\nabla f(\boldsymbol{\theta}_{i+1})|| \leq \varepsilon(1 + |f(\boldsymbol{\theta}_{i+1})|) \tag{18a}$$
$$f(\boldsymbol{\theta}_i) - f(\boldsymbol{\theta}_{i+1}) \leq \varepsilon(1 + |f(\boldsymbol{\theta}_{i+1})|) \tag{18b}$$
$$||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1}|| \leq \varepsilon(1 + |\boldsymbol{\theta}_{i+1}|), \tag{18c}$$

där L^2-normen användes [29]. $\boldsymbol{\theta}_i$ är punkten vid den aktuella iterationen inom optimeringen för ett specifikt startvärde och $\boldsymbol{\theta}_{i-1}$ är punkten vid föregående iteration. ε sattes i implementeringen till 10^{-3}. Det första termineringskravet (18a) uppnås om gradienten är för liten. Termineringskrav två (18b) jämför funktionsvärdet med det för föregående punkt medan det tredje (18c) avgör om steglängden anses för liten. Om minst två av tre termineringkrav uppnåddes avbröts optimeringen för det aktuella startvärdet.

4.1.4 Begränsningar i sökområdet

För att söka efter ett minimum i kostnadsfunktionen krävs ett specificerat område där optimeringen sker. Inledningsvis sattes en övre och en undre gräns på samtliga parametrar till 10^{-3} respektive 10^3. Gränserna för parametrarna justerades därefter utefter vad parametrarna hos minimipunkten tenderade mot. Exempelvis om en parameters optimala värde från en optimering låg precis vid dess övre eller undre gräns utökades gränsen i den riktningen för att ge parametern mer utrymme. Detta för att sökområdet skulle innehålla kostnadsfunktionens globala minimum.

4.1.5 Generering av startvärden

Då det inte är garanterat att en minimipunkt som ges av optimeringsalgoritmen är en global minimipunkt är det väsentligt att utgå från många olika startpunkter. Dessa bör vara väl utspridda för att med större sannolikhet lyckas hitta det globala minimat. Latin hypercybe-sampling är den metod som användes för att få slumpade punkter jämnt utspridda i parameterrymden [37]. Detta genomfördes genom att spannet av vardera variabel delades in i lika stora delar och att ett värde valdes slumpvis från varje intervall. Därefter parades de slumpade punkterna från varje variabel

ihop på ett sådant sätt att varje variabel blev oberoende av varandra. Det innebar att varje genererad punkt blev unik för ett specifikt intervall från varje variabel.

De värden som returnerades av metoden kunde därefter användas som startpunkter i minimeringsalgoritmen. För parameterestimationen genererades 1000 startvärden med hjälp Random.jl för att generera slumpade tal.

4.2 Identifierbarhetsanalys med profile likelihood

För att profilera kostnadsfunktionen genom profile likelihood hölls en parameter i taget konstant medan resterande parametrar optimerades med samma metod som beskrivs i Avsnitt 4.1. Efter att parametrarna optimerats minskades värdet på den konstanta parametern med en adaptiv steglängd och de övriga parametrarna optimerades på nytt. Steglängden bestämdes utifrån sambandet

$$\chi^2(\boldsymbol{\theta}_{last} + \boldsymbol{\theta}_{step}) - \chi^2(\boldsymbol{\theta}_{last}) \approx q \cdot \Delta_\alpha, \tag{19}$$

där $\boldsymbol{\theta}_{last}$ motsvarar parametervärdet från den senaste iterationen och $\boldsymbol{\theta}_{step}$ den steglängd för parametern som profileras [32]. Differensen för funktionsvärdena ska alltså infalla sig vid tröskelvärdet Δ_α multiplicerat med en faktor q. Tröskelvärdet som användes är en χ^2-fördelning med 95% signifikans med en frihetsgrad, vilket gav

$$\Delta_\alpha = \chi^2(\alpha, df) = \chi^2(0.05, 1) = 3.84. \tag{20}$$

Faktorn q sattes till 0.1 [32]. Detta upprepades maximalt 100 gånger eller tills att parametern gått utanför sitt konfidensintervall som definierades av θ givet att

$$\chi^2(\boldsymbol{\theta}) - \chi^2(\hat{\boldsymbol{\theta}}) < \Delta_\alpha, \tag{21}$$

där $\hat{\boldsymbol{\theta}}$ är de optimerade parametrarna. Därefter upprepades samma process men med en ökning istället för minskning av parametervärdet.

Den initiala steglängden som testades för när det konstanta parametervärdet skulle ändras, var 10^{-3} multiplicerat med det aktuella parametervärdet. Detta för att göra steglängden relativ till storleken av parametern. Om differensen mellan funktionsvärdena i (19) var för liten ökades steglängden tvåfaldigt och likaså halverades den när differensen var för stor.

4.3 Kostnadsfunktionen och ODE:lösare

I styckena ovan har kostnadsfunktionen beskrivits som en abstrakt funktion vi vill finna minimum av. Detta avsnitt beskriver dess struktur och hur funktionen fungerar. Kopplat till det beskrivs även hur vi kan lösa ODE-systemet över tid.

Avsnitt 3.4 introducerade kostnadsfunktionen vi vill finna minimum av som

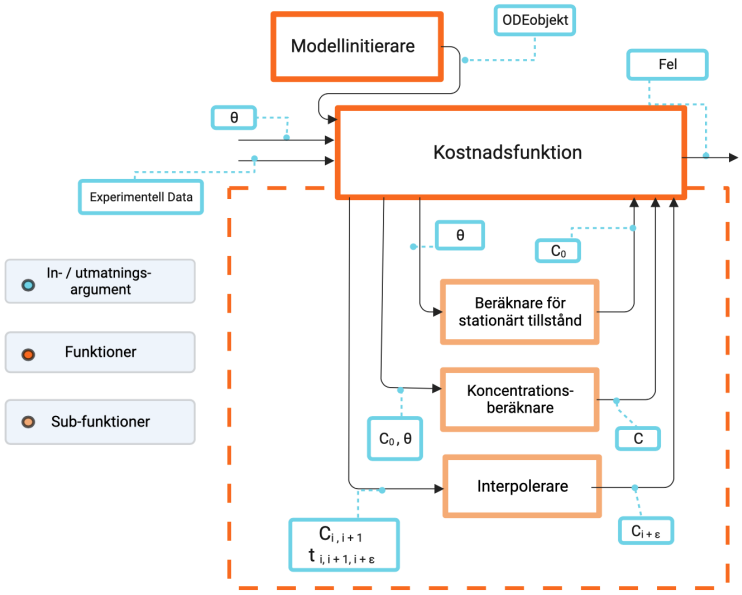
$$f(\boldsymbol{\theta}) = \sum_{i=0}^{j} \left(\boldsymbol{y}_i - \mathbf{ODEmodell}(\boldsymbol{x}_i, \boldsymbol{\theta})\right)^2, \tag{22}$$

där $\boldsymbol{x}_i$ är initialkoncentrationerna för alla substanser, $\boldsymbol{y}_i$ är slutkoncentrationerna för några substanser och j är antalet experimentella förhållanden med avseende på tid, initialkoncentration och

variationer på systemet.

Den experimentella datan $\boldsymbol{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j\}$ tillhör fyra mängder. De första två mängderna innehåller koncentrationerna av mHXT1, mHXT2, mHXT3 och mHXT4 vid regelbundna tidpunkter efter spädning med 0,1% respektive 0,2% glukos. De andra två har mätt koncentrationen av mHXT4 över tid då 0,1% glukos tillsatts till celler där antingen genen för Rgt1 eller Mig2 slagits ut. Utslagningen innebär att proteinerna inte längre produceras.

Kostnadsfunktionens struktur presenteras i stora drag i Figur 3. Övriga funktioner presenteras mer detaljerat i kommande stycken.



Figur 3: Översiktlig struktur av kostnadsfunktionen och de primära funktionerna den använder. Varje funktion tar in pilarna till vänster och returnerar pilen åt höger. Pilarna under en funktion går till funktioner som kallas på när den evalueras. Kostnadsfunktionen kallar på funktionerna under den i ordning nedifrån och upp.

4.3.1 Modellinitierare

Denna funktion initierade ett ODE-system vilket användes av andra funktioner för att beräkna koncentrationerna. I projektet formulerades differentialekvationer för systemets reaktioner i delnätverken för induktion och repression. Dessa sågs därefter över i syfte att förkorta bort eller förenkla ekvationerna och därmed minska komplexiteten. Simplifieringarna berördes detaljerat i Avsnitt 3.2.

ODE-erna formulerades i strukturen för ModellingToolkit.jl (verision 8.55.1) [38]. I denna definierades 11 parametrar och den extracellulära glukosen, där parametrarna motsvarar systemets okända värden. Dessa går att observera i Figur 4 som systemets blåmfärgade pilar. Värdet på de kända parametrarna togs från den data som fanns publicerad i artikeln av Kuttykrishnan m.fl. Funktionen *structural_simplify* från ModellingToolkit.jl användes slutligen för att skriva om ODE:rna så att systemet blev automatiskt lösbart för ODE-lösaren.

För att kunna testa systemet när gener slagits ut, introducerades två kontrollparametrar. Dessa sattes till ett värde av noll när tidsderivatan av mRGT1 respektive mMIG2 skulle ha tidsderivatan noll, och i övriga fall till värdet ett.

4.3.2 Koncentrationsberäknare

Målet med att etablera *Koncentrationsberäknaren* var att få returnerat systemets substanskoncentrationer över tid, genom att låta en ODE-lösare lösa det framtagna ODE-systemet. Detta under förutsättningen att parametrar och initialvillkor uppdaterades för systemet vid varje upprepning, samtidigt som lösarens arbetstid varierats.

Funktionen tog in ODE-systemet och uppdaterade dess parametervärden, initialkoncentrationer och sluttid med *Remake* ur DifferentialEquations.jl (version 7.7.1) [39]. Därefter löstes problemet över tid med ODE-lösaren Rodas5P med en absolut och relativ tolerans på 10^{-8}.

4.3.3 Interpolerare

För att uppfatta hur väl modellens parameterprediktioner förhöll sig till verkligheten krävdes en jämförelse mellan den experimentella datan och modellens prediktioner. Eftersom mätdatan var begränsad till ett fåtal tidpunkter, introducerades en interpolerare i syfte att kunna få fram koncentrationer för nya tidpunkter att inkludera i jämförelsen. I modellens *Interpolerare* matades två beräknade koncentrationer ($C_{i,\,i+1}$) in med dess motsvarande tidpunkter ($t_{i,\,i+1}$) tillsammans med en ytterligare tidpunkt ($t_{i+\epsilon}$), för att kunna returnera en linjärinterpolerad koncentration vid $t_{i+\epsilon}$.

4.3.4 Beräknare för stationärt tillstånd

För att kunna lösa ODE-systemet kräver ODE-lösaren initialkoncentrationerna av alla substanser. Dessa är inte kända. För att lösa detta läts systemet gå till stationärt tillstånd med en extracellulär glukoshalt satt till noll likt experimentella datan.

Initialkoncentrationerna beräknades av en separat funktion vilken tog in parametervärden och beräknade koncentrationerna då systemet nått stationärt tillstånd. Detta genomfördes genom att starta med samtliga initialkoncentrationer som noll och köra systemet utan extracellulär glukoshalt. Systemet kördes tills det var tillräckligt nära stationärt tillstånd, det vill säga att derivatan var nära noll. Villkoret för detta formuleras enligt

$$\frac{1}{\text{Antal subs.}}\left(\sum_{i\,\in\,\text{subs.}}\left(\frac{\mathrm{d}C_i}{\mathrm{d}t}\frac{1}{C_i\text{reltol.} + \text{abstol.}}\right)^2\right)^{1/2} < 1, \tag{23}$$

där relativ och absolut tolerans sattes till 100 gånger lägre än gränserna för andra ODE-lösaren, alltså $10^{-8}/100$.

Stoppandet av ODE-lösaren implementerades med en *DiscreteCallback* från DifferentialEquations.jl (version 7.7.1) vilken inkluderas i ModellingToolkit.jl [40].

4.3.5 Anpassning av kostnadsfunktionen och dess sub-funktioner för att möjliggöra optimering

I optimeringen beräknades derivatan av kostnadsfunktionen med hjälp av automatisk differentiering vilket innebär att funktionen måste kunna ta in dualtal och propagera dessa till funktionsvärdet. Då DiffrentialEquations.jl kan hantera dualtal innebar det endast att alla koncentrationer, samt extra parametrar alltså glukoskoncentration och kontrollparametrarna, konverterades till samma typ som parametrarna kostnadsfunktionen tog in.

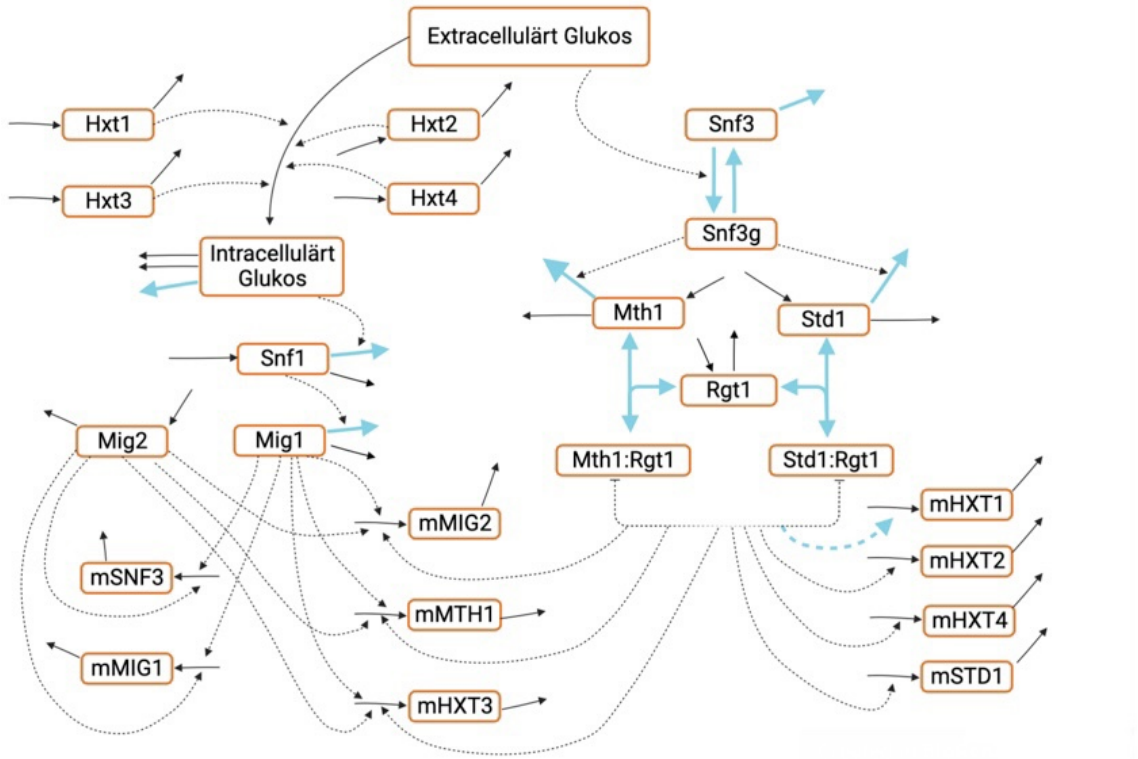
5 Resultat

Resultatet är upplagt i två delar, där den första presenterar modellen som utvecklats för att beskriva regleringsnätverket för glukosupptagning av *S. cerevisiae*. Den andra delen visar resultaten för parameterestimationen av de okända parametrarna i systemet.

5.1 Sammanfattning av systemets kinetik

Utifrån de antaganden och förenklingar som presenteras i Avsnitt 3.2 kunde uttryck för glukosregleringen hos jästen formuleras. Uttrycken har formulerats utifrån delnätverkens sex olika sorters reaktioner: degradering, translation, transmembrantransport, transkription, protein-protein interaktion och fosforylering formuleras. När vi formulerade ekvationerna behövde dock några fler antaganden tillämpas för att få ihop kompletta differentialekvationer för samtliga komponenter i regleringen.

Komplett data saknades för membranproteinerna Snf3 och Rgt2, vilka besitter samma funktion i induktionsnätverket. Problemet åtgärdades därför genom att endast inkludera Snf3 i modellen och komplettera avsaknad data med degraderings- och translateringshastighet tagen från Rgt2. Yck1 och Yck2 är nästa delsteg i induktionsnätverket. Kinaserna är relativt ostuderade och är helt uteslutna från den artikel av Kuttykrishnan m.fl. som studerats. Eftersom ingen data finns att erhålla för paret förenklades dessa ur modellen genom att anta stationärt tillstånd. ADP i det repressiva delnätverket togs därefter ur modellen med argumentet att aktiveringen av ATP och ADP står i jämvikt i systemet. Jämvikten innebär att mängden intracellulärt glukos är direkt korrelerad till mängden ADP. På det sättet kunde uttrycket för ADP tas ur modellen. Systemet som erhölls efter de slutgiltiga antagandena hade 23 differentialekvationer och en algebraisk ekvation. Dessa illustreras schematiskt i Figur 4 och finns nedtecknade i Appendix C.



Figur 4: Schematisk bild över ODE-systemet utifrån glukosnätverkets kinetik. Det högra flödet symboliserar induktionnätverket och det vänstra det repressiva. Hela enkelriktade pilar i figuren representerar reaktioner där substanser bildas, degraderas eller bildar en annan substans. Streckade pilar innebär att de influerar hastigheten av en annan reaktion och dubbelriktade att de representerar en jämnviktsreaktion. Blåa pilar representerar reaktioner med ett okänt parametervärde som därför måste uppskattas. Observera att hastigheten för bildning av proteiner är proportionell mot mängden av dess mRNA vilket inte illustreras i figuren.

I Figur 4 finns 11 reaktioner vilka alla har en parameter vars värde saknas. Dessa reaktioner illustreras i figuren som blåa pilar och parametrarna benämns genomgående i rapporten som $\boldsymbol{\theta}_e$. Alla $\boldsymbol{\theta}_e$ måste bestämmas för att kunna använda modellen. Parametrarna $\theta_{Mig1:Snf3}$ och $\theta_{Mig2:MIG1}$ från Kuttykrishnan m.fl. är inte mer exakt givna än 0.000 vilket gör att även dessa inkluderats som okända parametrar i vissa modeller [6].

5.2 Implementering av testmodell

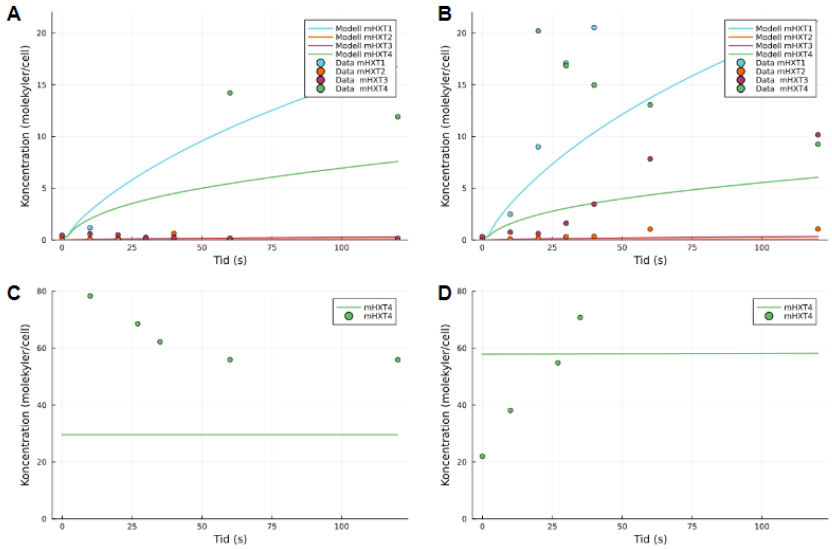
Som steg i byggandet av vår modell konstruerades enklare modeller för att testa delar av systemet. En mycket enkel modell och optimering på denna presenteras utförligt i Appendix F. Från optimeringen för den här modellen erhölls parametervärden som var mycket nära de sanna värdena. En identifierbarhetsanalys av parametrarna visar att de är praktiskt icke-identifierbara vilket förväntades då parametrarna är starkt korrelerade.

5.3 Estimering av modellparametrar

För optimering av parametrarna i glukosregleringsmodellen har olika varianter av modellen och datan använts. Först kördes koden med exkludering av datan från muterade celler och i alla följande tester inkluderades den. Då alla experiment replikerats en gång testades därefter att annvända

deras medelvärde istället för båda två. Detta gjordes då datan från de muterade cellerna inkluderades. Slutligen genomfördes samma sak men med en utökad variant av modellen som inkluderade $\theta_{Mig1:Snf3}$ och $\theta_{Mig2:MIG1}$ som okända parametrar. Ingen av genererade parameterestimationer följde experimentell data på något betydande sätt. Nedan presenteras resultaten för sista modellen som testades.

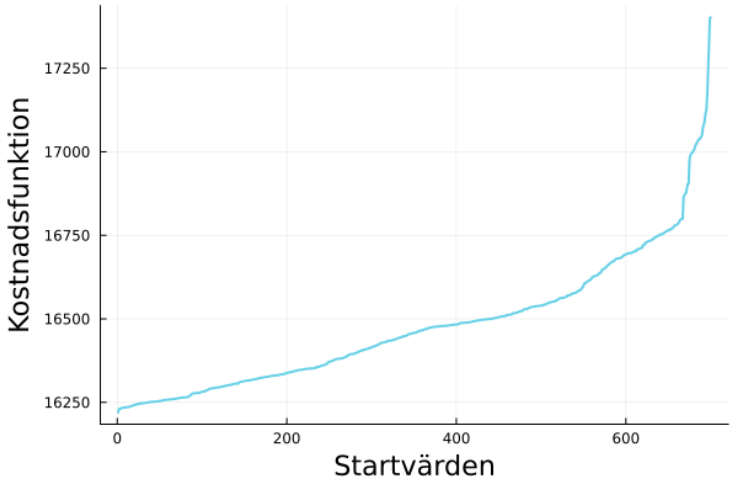
Första kolumnen i Tabell 2 visar bästa estimationen på optimeringen. Denna användes till att lösa ODE-systemet över tid, vilken sedan har jämförts med experimentell data i Figur 5. För en välfungerande modell bör skillnaden mellan modellen av experiment vara liten.



Figur 5: Jämförelse av modell och datan den har optimerats på för de fyra generna HXT1-4. De fyra experimenten är **A)** 0.1% glukos tillsatt, **B)** 0.2% glukos tillsatt, **C)** 0.1% glukos tillsatt till kultur med RGT1 utslagen samt **D)** 0.1% glukos tillsatt till kultur med MIG2 utslagen. För en bra modell ska modellen röra sig nära experimenten.

5.4 Konvergens av optimum för parameteruppskattningen

Figur 6 visar en överblick över de erhållna funktionsvärdena som nåddes för varje startpunkt, sorterade i storlek av kostnad. Notera att antalet startvärden har trunkerats till de 650 lägsta av alla 1000. Om globala optimum hade hittats, förväntades flera av startvärdena nå samma punkt och därmed samma kostnad, vilket figuren inte visar. Detta då förutsättningarna hade resulterat i platta regioner för grafen.



Figur 6: Överblick över 650 av de 1000 bästa funktionsvärdena. De optimerade funktionsvärdena har sorterats i storleksordning för att kunna se trender i var de optimerade punkterna hamnar. Är kurvan plan någonstans i grafen har troligtvis en del av startvärdena konvergerat till samma minimum.

Även de bästa estimationerna som erhölls gav stor spridning i parametrarnas värden (Tabell 2). Detta ger en ytterligare indikation på att parametervärdena inte konvergerar till ett globalt minimum.

Tabell 2: Jämförelse av de sex bästa estimationerna som erhölls från de 1000 startvärdena. Notera att parametrarna har olika enheter, det väsentliga är jämförelsen av samma parameter mellan olika optimerade parametervärden

Parameter	**Parameterärde**						*
$k_{a,Snf3}$	0.797	1.46	1.2	97.1	0.0795	180.0	-
$k_{i,Snf3g}$	68.9	50.6	73.1	850000.0	63.3	211000.0	-
$k_{i,Std1}$	7000.0	8660.0	5240.0	1560.0	8990.0	149.0	5.98e-8
$k_{i,Mth1}$	0.0159	0.0521	0.0847	0.35	0.00047	0.00457	0.01
$K_{Mth1:Rgt1}$	17.3	75.3	57.8	271.0	0.607	6.51	0.01
$K_{Std1:Rgt1}$	0.00867	0.0309	7.77	0.0703	8.2	2.62	0.002
$k_{i,Snf1}$	0.000124	0.000674	0.00374	2.47e-6	0.00327	8.59e-8	1.21
$k_{i,Mig1}$	304.0	77200.0	1410.0	180.0	5830.0	70.6	4.97
$k_{p,ATP}$	112.0	3.43	454.0	2.77	197.0	0.481	9.99e9
$\theta_{Mig1,Snf3}$	0.000308	6.35e-9	1.49e-6	0.00055	1.75e-8	0.00125	0.000 **
T_{mHXT1}	235.0	154.0	2180.0	77.2	324.0	136.0	0.130
$\theta_{\text{activation}}$	0.000136	1.32e-5	0.00322	2.51e-7	0.000171	2.1e-7	3.01
$\theta_{Mig2,MIG1}$	24000.0	133000.0	108.0	685000.0	6.37e6	409.0	0.000 **
Funktionsvärde	16220.24	16229.03	16231.5	16231.99	16232.16	16233.45	-

* Estimerade parametrar från artikeln av Kuttykrishnan m.fl.
** Dessa parametrar är direkt bestämda i artikeln och inte optimerade.

5.5 Jämförelse av beräkningshastighet

Sista modellen som testades tog 13 timmar på en Intel Core i7-13700K med 32GB ram. Test av de olika delberäkningarna presenteras i Tabell 3.

Tabell 3: Tiden olika operationer som är del av optimering tog. Varje gång kostnadsfunktionen evalueras löses ODE-systemet 7 olika gånger. Testerna genomfördes på en Intel Core i7-13700K med 32GB ram.

Beräkning	**Tid** (s)
Lösning av ODE	0.00559
Kostnadsfunktionens värde	0.0162
Kostnadsfunktionens gradient	0.137
Kostnadsfunktionens hessian	5.06

5.6 Indentifierbarhetsanalys

Som observerat ger parameterestimationen mycket instabila minimipunkter. Därav förfaller syftet med en identifierbarhetsanalys. Av denna anledning tillämpades inte profile likelihood.

6 Diskussion

Under projektet var ambitionen att bygga en modell för glukosnätverket hos *S. cerevisiae*. Förhoppningen var att få bra estimerade värden för de okända parametrarna, nära de koncentrationer som fanns presenterade som experimentell data i artikeln av Kuttykrishnnan m.fl., som vi kunde observera inte följde den experimentella datan (Fig. 5). Även för de optimeringskörningar som producerade lägst värden för kostnadsfunktionen skilde sig parametervärdena mycket (Tab. 2). Konvergensen av minimipunkten för parameteruppskattningen (Fig. 6) visar också att startvärdena inte konvergerade. Det finns olika områden vilka kan ligga som grund till att parameterestimationen ej verkar gått att lösa. Dessa är en dålig modellkonstruktion, begränsning av tillgänglig datamängd och en ineffektiv optimering.

Den första möjliga förklaringen till att parameteruppskattningen inte har gett önskade resultat är valet av optimeringsalgoritm. Den valda algoritmen, Steepest descent, har fördelen att den alltid går i en nedåtgående riktning. Vardera steg är effektivt att beräkna då metoden endast kräver gradienten och inte hessianen, som tar ca 37 gånger längre tid att beräkna (Tab. 3). Denna stora tidskillnad förväntades då gradienten har en linjär komplexitet medan hessianen har en kvadratisk sådan. Däremot har Steepest descent en långsam konvergeringshastighet då det är en linjär approximation och kostnadsfunktionen är högst olinjär. Dess långsamma konvergeringshastighet innebär att väldigt många steg kan behöva tas för att nå ett optimum. För optimeringen skedde ingen konvergens till något minimum, varken ett globalt eller lokala (Fig. 6). Detta indikerar att optimeringen av varje startvärde slutar för tidigt, vilket regleras av termineringskraven (Avsnitt 4.1.3). Alltså är en möjlig förklaring till att minimum inte hittades för kostnadsfunktionen att optimeringen inte tilläts gå tillräckligt långt då termineringkraven nåddes, vilket är en konsekvens av Steepest descent-metodens karaktär. För att nå ett minimum kan termineringskraven göras mindre strikta, dock skulle detta kunna leda till en mycket långsam optimering då många steg behöver tas. Alternativet är att välja en bättre optimeringsmetod där mer effektiva steg tas. Detta minskar risken att termineringskraven möts då ett minimum ännu inte nåtts.

Newtons metod anses i många fall bättre än Steepest descent i och med den snabbare konvergeringshastigheten. Trots detta användes inte metoden då den kräver att hessianen för samtliga

punkter i optimeringen är positivt definita för att hitta ett minimum i kostnadsfunktionen. I den ursprungliga optimeringskoden användes båda metoderna. Hessianen beräknades vid varje steg och därefter valdes metod utefter hessianens definitivitet. När vi testade algoritmen för färre punkter uppmärksammades att endast en väldigt liten andel av hessianerna som beräknades var positivt definita. Därför beslutades det att använda enbart Steepest descent-metoden. Att bara använda den metoden var motiverat utifrån att hessianen annars behövde beräknas utifrån varje iteration, vilket är väldigt beräkningstungt (Tab. 3), oavsett vilken metod som användes.

Vid val av optimeringsalgoritm blir det väsentligt att göra en avvägning över hur många steg som måste tas samt hur snabba stegen är. De två metoderna som diskuterats är bra på olika sätt och ingen av dem blir därav optimal att använda. Ett alternativ till metoderna hade varit att använda Newtons metod men med en Gauss-Newton-approximation av hessianen. I den metoden används jacobianen [41], vilket är en matris med första ordningens partiella derivator, för approximationen. Då det är just första och inte andra ordningens derivator som beräknas blir beräkningstiden för approximationen i samma storleksordning som för gradienten (Tab. 3). Detta är markant mycket snabbare än tiden det tar att beräkna hessianen vilket gör varje steg betydelsevärt snabbare än för en vanlig Newtons metod. Däremot är det värt att notera att då det endast är en approximation av hessianen som användes ger metoden generellt inte bättre resultat än vad användningen av en sann hessian hade gjort.

Då Newtons metod, och varianter av den, har förmågan att ta för långa steg kan en trust region-metod användas [42]. En trust-region undviker den komplikationen genom att begränsa hur långt bort nästa iteration kan hamna i förhållande till den nuvarande iterationen. I den algoritm som användes i projektet för att hitta en steglängd, hittades en godtagbar steglängd i en given sökriktning. Sökningen för nästa punkt höll sig till den specifika riktningen. Detta skiljer sig från en trust region-metod som möjliggör en förändring av sökriktningen när stegen minskar i storlek [43]. Den flexibla avgränsningen medför även att definitiviteten av den approximerade hessianen inte behöver tas hänsyn till vilket underlättar avsevärt [43]. Den här kombinationen av Newtons metod med en trust-region samt en Gauss-Newton approximation är en välanvänd metod som visat sig resultera i en effektiv optimerare för biokemiska reaktioner i cellulära nätverk [44].

Ett annat potentiellt problem kan ligga i strukturen på modellen. I modellskapandet gjordes avvägningar på vilken detaljnivå systemet ska beskrivas. En högre detaljnivå innebär att mer komplicerade interaktioner och beteenden kan beskrivas, vilket också medför att modellen blir mer användbar. Om modellen vi utvecklat har för låg detaljnivå är det möjligt att det inte existerar några parametervärden sådana att modellen kan bete sig som den experimentella datan. Modellen av Kuttykrishnnan m.fl. följer datan någorlunda väl vilket innebär att modellen som konstruerades i detta arbete också borde kunna göra det, då samma data har använts och modellerna är mycket lika [6]. Alltså borde inte en för låg detaljnivå i vår modell vara grunden till att modellen inte följer datan (Fig. 5).

Om modellen istället har en hög detaljnivå ökar antalet parametrar i modellen som behöver bestämmas, vilket medför att optimeringen blir svårare. Detta är till följd av att en ökningen i antalet dimensioner av parameterrymden medför en exponentiellt växande mängd möjliga kombinationer av variabler [45]. Vid ett stort antal parametrar finns en betydande risk att introducera strukturellt icke-identifierbara parametrar, det vill säga parametrar vilka vid flera olika värden genererar samma modellobservationer. Trots att dessa parametrar inte går att entydigt bestämma, kommer en optimering med strukturellt icke-identifierbara parametrar kunna generera modeller vilka följer experimentell data väl. Verktyg som analyserar systemet utan att behöva ta hänsyn till datan existerar men ligger utanför detta arbetets omfattning [46].

Generellt är modellkonstruktion en iterativ process där fler parametrar introduceras när modellen inte verkar kunna beskriva beteenden, och tas bort när de är överflödiga. Den begränsade tidsskalan i arbetet samt att skapande av experimentell data varit utanför projektets ramar, har begränsat modellkonstruktionerna som varit möjliga.

Ett ytterligare problem kan ligga i att mängden tillgänglig data är för liten för att bestämma para-

metrarna. Parametrar som inte kan bestämmas från tillgänglig data är praktiskt icke-identifierbara, vilket just av dess definition är ett betydande problem för parameterestimationen. Detta är en potentiell orsak till att parameterestimationen inte lyckades då den tillgängliga datan kan ha varit otillräcklig. Detta skulle då kunna åtgärdas genom att inkludera mer data.

En annan komplikation i den använda datan är att varje experimentell mätning upprepades två gånger. Det här ökar det absoluta värdet av kostnadsfunktionen samt minskar relativa skillnaden på värdet av kostnadsfunktionen genom att introducera ett stort fel för alla värden. Detta hade en signifikans för bland annat ett av termineringsvilkoren vid Steepest descent (18b) vilken stannade om relativa skillnaden på värdet av kostnadsfunktionen var för liten. Vid de sista parameterestimationerna användes därför medelvärden av tillgänglig data.

Vi har nu diskuterat olika möjligheter till att modellens okända parametrar inte kunde bestämmas. Resonemangen visar på att parameterestimation för modeller över biologiska system är något som är svårt att göra ur flera avseenden. På grund av att det finns många typer av modeller, krävs olika anpassningar för att kunna bestämma okända parametrar i dessa modeller. Detta medför att delar av parameteruppskattningen kan behöva implementeras från grunden och således tillkommer risker för misstag som relativt enkelt leder till felaktiga resultat. Dessutom minskas reproducerbarheten och möjligheten för att återanvända kod, vilket bidrar till utmaningen att utföra korrekta modelleringar. Svårigheten medför ett behov för standarder kring den här typen av modellering. Ett exempel på en sådan standardisering är PEtab, där ett parameteruppskattningsproblem kan formuleras i ett specifikt format som sedan stöds av många olika typer av mjukvaror [47]. Dock är svårigheten med att utforma den här typen av standarder att stor anpassningsbarhet krävs för att täcka det behov som finns. Trots svårigheten täcks det problem som möts i arbetet väldigt tydligt av standarden. Därför hade det funnits goda möjligheter att formulera problemet i en PEtab standard.

Under projektet hade användandet av en standardisering för parameteruppskattning kunnat underlätta arbetet kring optimeringsalgoritmen. Under arbetsprocessen implementerades en algoritm för optimeringen från grunden och där uppstod frågor kring metodval för algoritmen att finna kostnadsfunktionens minimum. Då det är svårt att veta vilken algoritm som är optimal för sin modell underlättar det med ett standardbibliotek där dessa kan varieras utan att själv behöva implementera koden. Dessutom skulle det tillfört möjligheten för andra personer att testa och vidareutveckla den framtagna modellen.

Sammanfattningsvis kan anledningen till att parameterestimeringen inte gått som önskats bero på många olika faktorer. Dessa inkluderar modellstrukturen, bristen på data och ett dåligt val av optimeringsalgoritm. I framtida projekt hade det varit intressant att undersöka om bättre resultat hade erhållits med ett annat val av optimeringsalgoritm samt en annan modellstruktur. Om optimeringen hade gett identifierbara parametrar hade det varit möjligt att verifiera, alternativt falsifiera, modellparametrarna som Kuttykrishnan m.fl. erhöll. Modellens reproducerbarhet är relevant eftersom modellen kan ligga till teoretisk grund för övrig forskning inom området. Om den här typen av modeller kan ta fram estimationer som svarar mot de som är mätbara för cellulära processer, kan slutsatserna användas för ökad förståelse för näringssignalering och metabolism. Insikterna kan därefter hjälpa oss att bättre förstå och förebygga metaboliska sjukdomar för att främja global hälsa.

Referenser

1. *Mål 3: God Hälsa och Välbefinnande* okt. 2022. https://www.globalamalen.se/om-globala-malen/mal-3-halsa-och-valbefinnande/.
2. Chalkiadaki, A. & Guarente, L. Sirtuins mediate mammalian metabolic responses to nutrient availability. *Nature Reviews Endocrinology* **8,** 287–296. https://www.nature.com/articles/nrendo.2011.225 (2012).
3. *Life expectancy at birth (years)* https://www.who.int/data/gho/data/indicators/indicator-details/GHO/life-expectancy-at-birth-(years).
4. Stewart, G. i *Encyclopedia of Food Microbiology* (utg. Batt, C. A. & Tortorello, M. L.) 2. utg. 10 Febuary 2023, 309–315 (Academic Press, Oxford, 2014). ISBN: 978-0-12-384733-1. https://www.sciencedirect.com/science/article/pii/B9780123847300002925.
5. Motta, S. & Pappalardo, F. Mathematical modeling of biological systems. *Briefings in Bioinformatics* **14,** 411–422. ISSN: 1467-5463. eprint: https://academic.oup.com/bib/article-pdf/14/4/411/479256/bbs061.pdf. https://doi.org/10.1093/bib/bbs061 (okt. 2012).
6. Kuttykrishnan, S., Sabina, J., Langton, L. L., Johnston, M. & Brent, M. R. A quantitative model of glucose signaling in yeast reveals an incoherent feed forward loop leading to a specific, transient pulse of transcription. *Proceedings of the National Academy of Sciences* **107,** 16743–16748 (2010).
7. Yang, J., Ueharu, H. & Mishina, Y. Energy metabolism: A newly emerging target of BMP signaling in bone homeostasis. *Bone* **138,** 115467 (2020).
8. Cawley, J. *m. fl.* Direct medical costs of obesity in the United States and the most populous states. *Journal of managed care & specialty pharmacy* **27,** 354–366. https://pubmed.ncbi.nlm.nih.gov/33470881/ (2021).
9. Zhang, P. *m. fl.* Global healthcare expenditure on diabetes for 2010 and 2030. *Diabetes research and clinical practice* **87,** 293–301 (2010).
10. Von Ah Morano, A. E., Dorneles, G. P., Peres, A. & Lira, F. S. The role of glucose homeostasis on immune function in response to exercise: The impact of low or higher energetic conditions. *Journal of Cellular Physiology* **235,** 3169–3188. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcp.29228. https://onlinelibrary.wiley.com/doi/abs/10.1002/jcp.29228.
11. Alberts, B. *m. fl. Molecular Biology of the Cell* 6. utg., 1–1465 (Garland Science, 2014).
12. AKHURST, T. *Chapter 17 - The Role of Nuclear Medicine in the Diagnosis and Management of Hepatobiliary and Pancreatic Diseases* 4. utg., 234–265. ISBN: 978-1-4160-3256-4. https://www.sciencedirect.com/science/article/pii/B9781416032564500272 (W.B. Saunders, Philadelphia, 2007).
13. Kachroo, A. H. *m. fl.* Systematic humanization of yeast genes reveals conserved functions and genetic modularity. *Science* **348,** 921–925. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4718922/ (2015).
14. Kalisky, T., Dekel, E. & Alon, U. Cost–benefit theory and optimal design of gene regulation functions. *Physical Biology* **4,** 229. https://dx.doi.org/10.1088/1478-3975/4/4/001 (nov. 2007).
15. Tyson, J. J., Chen, K. C. & Novak, B. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. *Current Opinion in Cell Biology* **15,** 221–231. ISSN: 0955-0674. https://www.sciencedirect.com/science/article/pii/S0955067403000176 (2003).
16. Kreutz, C. A new approximation approach for transient differential equation models. *Frontiers in Physics* **8,** 70. https://www.frontiersin.org/articles/10.3389/fphy.2020.00070/full (2020).
17. *Saccharomyces Genome Database* https://www.yeastgenome.org/.

18. Lee, D., Son, H. G., Jung, Y. & Lee, S.-J. V. The role of dietary carbohydrates in organismal aging. *Cellular and Molecular Life Sciences* **74,** 1793–1803. https://link.springer.com/article/10.1007/s00018-016-2432-6 (2017).

19. Reifenberger, E., Freidel, K. & Ciriacy, M. Identification of novel HXT genes in Saccharomyces cerevisiae reveals the impact of individual hexose transporters on glycolytic flux. en. *Mol. Microbiol.* **16,** 157–167 (april 1995).

20. Ciliberto, A., Capuani, F. & Tyson, J. J. Modeling networks of coupled enzymatic reactions using the total quasi-steady state approximation. *PLoS computational biology* **3,** e45 (2007).

21. Kenakin, T. The mass action equation in pharmacology. *British Journal of Clinical Pharmacology* **81,** 41–51. https://doi.org/10.1111/bcp.12810 (dec. 2015).

22. Srinivasan, B. A guide to the Michaelis–Menten equation: steady state and beyond. *The FEBS journal* **289,** 6086–6098 (2022).

23. English, B. P. *m. fl.* Ever-fluctuating single enzyme molecules: Michaelis-Menten equation revisited. *Nature chemical biology* **2,** 87–94 (2006).

24. Schnell, S. Validity of the Michaelis–Menten equation–steady-state or reactant stationary assumption: that is the question. *The FEBS journal* **281,** 464–472 (2014).

25. Goutelle, S. *m. fl.* The Hill equation: a review of its capabilities in pharmacological modelling. *Fundamental & clinical pharmacology* **22,** 633–648 (2008).

26. Goutsias, J. & Lee, N. Computational and Experimental Approaches for Modeling Gene Regulatory Networks. *Current Pharmaceutical Design* **13,** 1415–1436. https://doi.org/10.2174/138161207780765945 (maj 2007).

27. Segel, L. A. & Slemrod, M. The quasi-steady-state assumption: a case study in perturbation. *SIAM review* **31,** 446–477 (1989).

28. García-Portugués, E. *Notes for Predictive Modeling* Version 6.9.11. ISBN 978-84-09-29679-8. https://bookdown.org/egarpor/PM-UC3M/ (2023).

29. Andréasson, N., Evgrafov, A. & Patriksson, M. *An Introduction to Continuous Optimization* ISBN: 9789144044552 (Professional Publishing Svc., 2005).

30. Meza, J. C. Steepest descent. *WIREs Computational Statistics* **2,** 719–722. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.117. https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.117.

31. George, S. & Sabari, M. Convergence rate results for steepest descent type method for nonlinear ill-posed equations. *Applied Mathematics and Computation* **294,** 169–179. ISSN: 0096-3003. https://www.sciencedirect.com/science/article/pii/S0096300316305690 (2017).

32. Raue, A. *m. fl.* Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Bioinformatics* **25,** 1923–1929 (2009).

33. Guillaume, J. H. *m. fl.* Introductory overview of identifiability analysis: A guide to evaluating whether you have the right type of data for your modeling purpose. *Environmental Modelling Software* **119,** 418–432. ISSN: 1364-8152. https://www.sciencedirect.com/science/article/pii/S1364815218307278 (2019).

34. Wieland, F.-G., Hauber, A. L., Rosenblatt, M., Tönsing, C. & Timmer, J. On structural and practical identifiability. *Current Opinion in Systems Biology* **25,** 60–69. ISSN: 2452-3100. https://www.sciencedirect.com/science/article/pii/S245231002100007X (2021).

35. Bezanson, J., Edelman, A., Karpinski, S. & Shah, V. B. Julia: A Fresh Approach to Numerical Computing. *SIAM Review* **59,** 65–98. https://epubs.siam.org/doi/10.1137/141000671 (2017).

36. Revels, J., Lubin, M. & Papamarkou, T. Forward-Mode Automatic Differentiation in Julia. *arXiv:1607.07892 [cs.MS].* https://arxiv.org/abs/1607.07892 (2016).

37. Atangana, A. i *Fractional Operators with Constant and Variable Order with Application to Geo-Hydrology* (utg. Atangana, A.) 15–47 (Academic Press, 2018). ISBN: 978-0-12-809670-3. https://www.sciencedirect.com/science/article/pii/B9780128096703000023.

38. Ma, Y. *m. fl. ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling* 2021. arXiv: 2103.05244 [cs.MS].

39. Rackauckas, C. & Nie, Q. DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software* **5** (2017).

40. Rackauckas, C. & Nie, Q. DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software* **5** (2017).

41. Chen, C., Reiz, S., Yu, C. D., Bungartz, H.-J. & Biros, G. Fast approximation of the Gauss–Newton Hessian matrix for the multilayer perceptron. *SIAM Journal on Matrix Analysis and Applications* **42,** 165–184. https://epubs.siam.org/doi/epdf/10.1137/19M129961X (2021).

42. Chen, P. Hessian matrix vs. Gauss–Newton hessian matrix. *SIAM Journal on Numerical Analysis* **49,** 1417–1435. https://epubs.siam.org/doi/pdf/10.1137/100799988 (2011).

43. Steven D. Brown, R. T. & Walczak, B. *Comprehensive Chemometrics: Chemical and Biochemical Data Analysis* ISBN: 978-0-12-381375-6 (Elsevier, 2009).

44. Fröhlich, F. & Sorger, P. K. Fides: Reliable trust-region optimization for parameter estimation of ordinary differential equation models. *PLOS Computational Biology* **18,** e1010322. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1010322 (2022).

45. Nocedal, J. & Wright, S. J. *High-Dimensional Optimization* (Springer Science & Business Media, 2006).

46. Wieland, F.-G., Hauber, A. L., Rosenblatt, M., Tönsing, C. & Timmer, J. On structural and practical identifiability. *Current Opinion in Systems Biology* **25,** 60–69. ISSN: 2452-3100. https://www.sciencedirect.com/science/article/pii/S245231002100007X (2021).

47. Schmiester, L. *m. fl.* PEtab—Interoperable specification of parameter estimation problems in systems biology. *PLoS computational biology* **17,** e1008646 (2021).

48. Peñuñuri, F., Carvente, O., Zambrano-Arjona, M., Peón, R. & Cruz-Villar, C. A. *Dual numbers for algorithmic differentiation* dec. 2019. https://www.redalyc.org/journal/467/46761359006/html/#redalyc_46761359006_ref30.

49. Wolfe, A. *Multivariable Dual Numbers Automatic Differentiation* febr. 2017. https://blog.demofox.org/2017/02/20/multivariable-dual-numbers-automatic-differentiation/.

A Appendix 1 - Begreppslista över biologiska termer

ATP	Ett ämne som fungerar som kortsiktig energikälla i många cellulära processer och som överlämnar energi genom att släppa en fosfatgrupp. ADP saknar fosfatgrupp medan ATP har den.
Degradering	Processen som ett ämne går igenom när det sönderfaller.
Enzym	Protein som genomför reaktioner.
Eukaryot	Gruppen organismer med cellkärna. Inkluderar djur, växter och svampar.
Fosforylering	Processen att sätta en fosfatgrupp på ett annat ämne. Fosforylering av ett protein ändrar ofta dess funktion.
Gen	En sekvens DNA som innehåller informationen för att göra ett specifikt protein.
Genuttryck	Hur mycket mRNA som bildas i en specifik celltyp.
Glukos	En sockerart som är en viktig energi- och byggkälla för många organismer.
Hexokinaser	Membranprotein som transporterar hexoser över cellmembranet.
Hexoser	Sockerarter med sex kolatomer i en ring, exempelvis glukos.
Katalys	När en kemisk eller fysikalisk förändringsprocess underlättas genom att ett ämne sänker aktiveringsenergin för reaktionen.
Komplex	Enhet med två eller fler komponenter, ofta proteiner.
Ligand	Liten molekyl som binder till protein och ändrar dess struktur.
Metabolism	Gemensamt namn för processer som tar upp näringsämnen, bryter ner dem för energi, bygger ihop dem till större beståndsdelar samt exporterar dem.
Modellorganism	Organism som studeras i syfte att få ökad biologisk förståelse för andra arter.
mRNA	En molekyl som innehåller information om hur ett protein ska se ut.
Organeller	Cellens inre delar, lokaliserade innanför cellmembranet. Kan betraktas som cellens organ.
Protein	Stora molekyler som har en variation av viktiga funktioner i organismer, bland annat inom metabolism.
Proteinkinaser	Enzymer som katalyserar proteinfosforylering.
Repressor	Protein som förhindrar transkription av gener.
Stationärt tillstånd	Ett tillstånd där inga observerbara förändringar inträffar i det kemiska/fysikaliska systemet. Kallas även steady-state.
Substrat	Den näring som en organism lever på.

B Appendix 2 - Automatisk Diffrentiering

En svårighet med att applicera Newtons metod och Steepest decent för mer komplicerade funktioner, är att beräkna dess derivator. Ett kraftigt verktyg för att lösa problemet är att använda automatisk differentiering [48].

Framåtackumulerande automatisk differentiering utnyttjar dualtal, vilka i stor likhet till komplexa talen kan definieras som $\hat{x} = a + \epsilon b$ med reella talen a, b där $\epsilon^2 = 0$ [48].

Taylorutvecklingen av analytiska funktionen $f : \mathbb{R} \to \mathbb{R}$ utvärderad runt $x + \epsilon$ kan skrivas som

$$f(x + \epsilon) = f(x) + f'(x)\epsilon + \cancelto{0}{O(\epsilon^2)} \tag{B.1}$$

och då $\epsilon^2 = 0$ kan högre ordningens termer styrkas. Ur detta kan vi se att vi får en evaluering av både funktionsvärdet och dess derivata i x när funktionen evalueras i $x + \epsilon$.

Dualtalen hanterar kedjeregeln väl så att $h(x) = f(g(x))$ ger $h(x + \epsilon) = f(g(x)) + h'(g(x))\epsilon$ vilket gör att nestade uttryck blir lätta att hantera.

För en analys av en variabelfunktion blir det således trivialt att beräkna derivatan givet att den är formulerad så den hanterar dualtal. Hanteringen av multivariabla funktioner $f : \mathbb{R}^n \to \mathbb{R}$ blir något mer komplicerat [49]. Enklast beräknas riktingsderivatorna separat för att sedan sättas ihop. Detta är mycket redundant och kan optimeras till stor grad.

C Appendix 3 - Systemets diffrentialekvationer

Nedan presenteras diffrentialekvationerna och den algebraiska ekvationen som använts för glukosnätverkets system.

$$\frac{\mathrm{dSnf3}}{\mathrm{d}t} = k_{i,Snf3g} \cdot \mathrm{Snf3g} + k_{t,Snf3} \cdot \mathrm{mSNF3} - k_{d,Snf3}\mathrm{Snf3} - \mathrm{Glucose}_{Ex} \cdot k_{a,Snf3} \cdot \mathrm{Snf3}$$

$$\frac{\mathrm{dSnf3g}}{\mathrm{d}t} = -k_{i,Snf3g} \cdot \mathrm{Snf3g} + \mathrm{Glucose}_{Ex} \cdot k_{a,Snf3} \cdot \mathrm{Snf3}$$

$$\frac{\mathrm{dStd1}}{\mathrm{d}t} = k_{t,Std1} \cdot \mathrm{mSTD1} - k_{d,Std1} \cdot \mathrm{Std1} - k_{i,Std1} \cdot \mathrm{Snf3g} \cdot \mathrm{Std1}$$

$$\frac{\mathrm{dMth1}}{\mathrm{d}t} = k_{t,Mth1} \cdot \mathrm{mMTH1} - k_{d,Mth1} \cdot \mathrm{Mth1} - k_{i,Mth1} \cdot \mathrm{Mth1} \cdot \mathrm{Snf3g}$$

$$\frac{\mathrm{dRgt1}}{\mathrm{d}t} = k_{t,Rgt1} \cdot \mathrm{mRGT1} - k_{d,Rgt1} \cdot \mathrm{Rgt1}$$

$$\mathrm{Rgt1}_{active} = K_{Mth1,Rgt1} \cdot \mathrm{Mth1} \cdot \mathrm{Rgt1} + K_{Std1,Rgt1} \cdot \mathrm{Rgt1} \cdot \mathrm{Std1}$$

$$\frac{\mathrm{dHxt1}}{\mathrm{d}t} = k_{t,Hxt1} \cdot \mathrm{mHXT1} - k_{d,Hxt1} \cdot \mathrm{Hxt1}$$

$$\frac{\mathrm{dHxt2}}{\mathrm{d}t} = k_{t,Hxt2} \cdot \mathrm{mHXT2} - k_{d,Hxt2} \cdot \mathrm{Hxt2}$$

$$\frac{\mathrm{dHxt3}}{\mathrm{d}t} = k_{t,Hxt3} \cdot \mathrm{mHXT3} - k_{d,Hxt3} \cdot \mathrm{Hxt3}$$

$$\frac{\mathrm{dHxt4}}{\mathrm{d}t} = k_{t,Hxt4} \cdot \mathrm{mHXT4} - k_{d,Hxt4} \cdot \mathrm{Hxt4}$$

$$\frac{\mathrm{dSnf1}}{\mathrm{d}t} = k_{t,Snf1} \cdot \mathrm{mSNF1} - k_{d,Snf1} \cdot \mathrm{Snf1} - k_{i,Snf1} \cdot \mathrm{Glucose}_{In} \cdot \mathrm{Snf1}$$

$$\frac{\mathrm{dMig1}}{\mathrm{d}t} = k_{t,Mig1} \cdot \mathrm{mMIG1} - k_{d,Mig1} \cdot \mathrm{Mig1} - k_{i,Mig1} \cdot \mathrm{Mig1} \cdot \mathrm{Snf1}$$

$$\frac{\mathrm{dMig2}}{\mathrm{d}t} = k_{t,Mig2} \cdot \mathrm{mMIG2} - k_{d,Mig2} \cdot \mathrm{Mig2}$$

$$\frac{\mathrm{dGlucose}_{In}}{\mathrm{d}t} = \frac{\mathrm{Glucose}_{Ex} \cdot V_{transport,Hxt1}}{\mathrm{Glucose}_{Ex} + K_{transport,Hxt1}} + \frac{\mathrm{Glucose}_{Ex} \cdot V_{transport,Hxt2}}{\mathrm{Glucose}_{Ex} + K_{transport,Hxt2}} +$$

$$\frac{\mathrm{Glucose}_{Ex} \cdot V_{transport,Hxt3}}{\mathrm{Glucose}_{Ex} + K_{transport,Hxt3}} + \frac{\mathrm{Glucose}_{Ex} \cdot V_{transport,Hxt4}}{\mathrm{Glucose}_{Ex} + K_{transport,Hxt4}} - k_{p,ATP} \cdot \mathrm{Glucose}_{In}$$

$$\frac{\mathrm{dmSNF3}}{\mathrm{d}t} = \frac{V_{mSNF3}}{(1 + \theta_{Mig1,Snf3} \cdot \mathrm{Mig1})\,(1 + \theta_{Mig2,Snf3} \cdot \mathrm{Mig2})} - k_{d,mSNF3} \cdot \mathrm{mSNF3}$$

$$\frac{\mathrm{dmSTD1}}{\mathrm{d}t} = \frac{V_{mSTD1}}{1 + \theta_{Rgt1,active,Std1} \cdot \mathrm{Rgt1}_{active}} - k_{d,mSTD1} \cdot \mathrm{mSTD1}$$

$$\frac{\mathrm{dmMTH1}}{\mathrm{d}t} = \frac{V_{mMTH1}}{(1 + \theta_{Mig1,MTH1} \cdot \mathrm{Mig1})\,(1 + \theta_{Mig2,MTH1} \cdot \mathrm{Mig2})\,(1 + \theta_{Rgt1,active,MTH1} \cdot \mathrm{Rgt1}_{active})} -$$
$$- k_{d,mMTH1} \cdot \mathrm{mMTH1}$$

$$\frac{\mathrm{dmRGT1}}{\mathrm{d}t} = V_{mRGT1} - k_{d,mRGT1} \cdot \mathrm{mRGT1}$$

$$\frac{\mathrm{dmHXT1}}{\mathrm{d}t} = \frac{V_{mHXT1}\left(T_{mHXT1} + \frac{\theta_{activation}(1-T_{mHXT1})\cdot \mathrm{Rgt1}}{1+\theta_{activation}\cdot \mathrm{Rgt1}}\right)}{1+\theta_{Rgt1,active,HXT1}\cdot \mathrm{Rgt1}_{active}} - k_{d,mHXT1}\cdot \mathrm{mHXT1}$$

$$\frac{\mathrm{dmHXT2}}{\mathrm{d}t} = \frac{V_{mHXT2}}{(1+\theta_{Mig1,HXT2}\cdot \mathrm{Mig1})\,(1+\theta_{Mig2,HXT2}\cdot \mathrm{Mig2})\,(1+\theta_{Rgt1,active,HXT2}\cdot \mathrm{Rgt1}_{active})} - \\ - k_{d,mHXT2}\cdot \mathrm{mHXT2}$$

$$\frac{\mathrm{dmHXT3}}{\mathrm{d}t} = \frac{V_{mHXT3}}{(1+\theta_{Mig1,HXT3}\cdot \mathrm{Mig1})\,(1+\theta_{Mig2,HXT3}\cdot \mathrm{Mig2})\,(1+\theta_{Rgt1,active,HXT3}\cdot \mathrm{Rgt1}_{active})} - \\ - k_{d,mHXT3}\cdot \mathrm{mHXT3}$$

$$\frac{\mathrm{dmHXT4}}{\mathrm{d}t} = \frac{V_{mHXT4}}{(1+\theta_{Mig1,HXT4}\cdot \mathrm{Mig1})\,(1+\theta_{Mig2,HXT4}\cdot \mathrm{Mig2})\,(1+\theta_{Rgt1,active,HXT4}\mathrm{Rgt1}_{active})} - \\ - k_{d,mHXT4}\cdot \mathrm{mHXT4}$$

$$\frac{\mathrm{dmMIG1}}{\mathrm{d}t} = \frac{V_{mMIG1}}{(1+\theta_{Mig1,MIG1}\cdot \mathrm{Mig1})\,(1+\theta_{Mig2,MIG1}\cdot \mathrm{Mig2})} - k_{d,mMIG1}\cdot \mathrm{mMIG1}$$

$$\frac{\mathrm{dmMIG2}}{\mathrm{d}t} = \frac{V_{mMIG2}}{(1+\theta_{Mig1,MIG2}\cdot \mathrm{Mig1})\,(1+\theta_{Mig2,MIG2}\cdot \mathrm{Mig2})\,(1+\theta_{Rgt1,active,MIG2}\cdot \mathrm{Rgt1}_{active})} - \\ - k_{d,mMIG2}\cdot \mathrm{mMIG2}$$

$$\frac{\mathrm{dmSNF1}}{\mathrm{d}t} = V_{mSNF1} - k_{d,mSNF1}\cdot \mathrm{mSNF1}$$

D Appendix 4 - Härledning av Michaelis-Mentens ekvation

Följande härledning är utförd utifrån artikeln *Validity of the Michaelis–Menten equation – steady-state or reactant stationary assumption: that is the question* av Santiago Schnell [24].

Massverkans lag kan appliceras för enzym-substrat-reaktionen presenterad i Avsnitt 3.2.1,

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} ES \xrightarrow{k_2} E \ + P,$$

för att få fram ett uttryck för hastigheten för enzym-substratkomplexet ES att bildas:

$$\frac{\mathrm{d}ES}{\mathrm{d}t} = k_1 \cdot E \cdot S - (k_{-1} + k_2) \cdot ES. \tag{D.1}$$

Genom att anta stationärt tillstånd för första delreaktionen kan koncentrationen av komplexet antas vara konstant och därför dess produktionshastighet ungefär noll:

$$\frac{\mathrm{d}ES}{\mathrm{d}t} \approx 0. \tag{D.2}$$

Antagandet ovan gör att följande konserveringslag,

$$E_0 = E + ES, \tag{D.3}$$

gäller för den irreversibla delen av reaktionen där komplex blir produkt och enzym. Genom att ersätta E i uttrycket ovan och anta stationärt tillstånd kan komplexet formuleras i termer av substrat enligt

$$ES = \frac{E_0 \cdot S}{K_m + S}, \tag{D.4}$$

där K_m motsvarar Michaelis-Menten konstanten:

$$K_m = \frac{k_{-1} + k_2}{k_1}. \tag{D.5}$$

I den irreversibla delen för reaktionen kan hastighetsförändringen för koncentrationen av produkten P, med hjälp av massverkans, lag formuleras som uttrycket nedan.

$$\frac{\mathrm{d}P}{\mathrm{d}t} = k_2 \cdot ES \tag{D.6}$$

Om ett substratöverskott antas för reaktionen kan man förutsätta att substratet som förbrukas i komplexbildningen blir försumbar mot totalmängden substrat. Detta resulterar i följande:

$$S_0 \approx S. \tag{D.7}$$

Genom att D.4och D.7 tillämpas för D.6 fås

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{V_{max} \cdot [S]}{K_s + [S]}, \tag{D.8}$$

som är den ekvation som presenteras i Avsnitt 3.2.1 som Michaelis-Menten-ekvationen. I uttrycket motsvarar konstanten V_{max} följande:

$$V_{max} = k_2 \cdot E_0. \tag{D.9}$$

Detta svarar mot maximala hastigheten för allt enzym att bli bundet som komplex.

E Appendix 5 - Härledning av Newtons metod

Funktionen f kan approximeras i närheten av punkten $\boldsymbol{x}_0$ med en andra gradens taylorutveckling enligt:

$$f(\boldsymbol{x}) \approx f_T(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \cdot \nabla f(\boldsymbol{x}_0) + \frac{1}{2} \cdot (\boldsymbol{x} - \boldsymbol{x}_0)^T \cdot \nabla^2 f(\boldsymbol{x}_0) \cdot (\boldsymbol{x} - \boldsymbol{x}_0), \tag{E.1}$$

där $\nabla^2 f(\boldsymbol{x}) \in \mathbb{R}^{k \times k}$ är Hessianen för funktionen f och $\nabla f(\boldsymbol{x}) \in \mathbb{R}^k$ är gradienten av funktionen f [29]. Givet den lokala approximationen f_T vid den aktuella punkten $\boldsymbol{x}_0$ vill vi hitta ett $\boldsymbol{x}$ sådant att f_T minimeras. Detta kan genomföras genom att ta gradienten på f_T, sätta den till 0 och lösa ut $\boldsymbol{x}$. Gradienten av f_T ger

$$\nabla f_T(\boldsymbol{x}) = \nabla f(\boldsymbol{x}_0) + \nabla^2 f(\boldsymbol{x}_0) \cdot (\boldsymbol{x} - \boldsymbol{x}_0), \tag{E.2}$$

varpå $\boldsymbol{x}$ kan lösas ut om $\nabla f_T(\boldsymbol{x}) = 0$. Detta ger slutligen

$$\boldsymbol{x} = \boldsymbol{x}_0 - \left(\nabla^2 f(\boldsymbol{x}_0)\right)^{-1} \cdot \nabla f(\boldsymbol{x}_0). \tag{E.3}$$

Uttrycket ovan kan generaliseras och en steglängd $\alpha_k \in [0, 1]$ för sökriktningen inkluderas:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \cdot \left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1} \cdot \nabla f(\boldsymbol{x}_k). \tag{E.4}$$

F Appendix 6 - Optimering samt indentifierbarhetsanalys på testmodell

I syfte att testa optimeringsalgoritmens funktion skapades en simpel modell där de sanna parametrarna var kända. Vi ställer upp en enkel reaktion

$$\mathrm{A} \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} \mathrm{B}, \tag{F.1}$$

där ämne A reagerar till ämne B med hastigheten k_1 och ämne B reagerar tillbaka till A med hastigheten k_{-1}. Vi kan definiera koncentrationsförändringen av ämne A respektive B enligt:

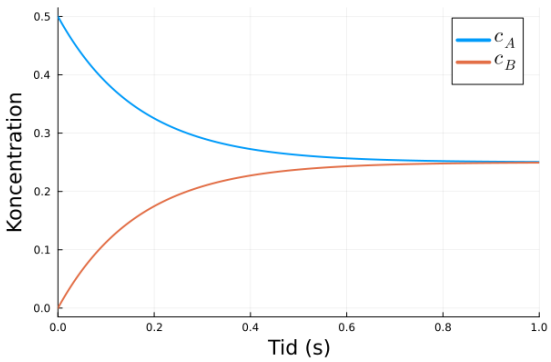
$$\begin{aligned} \frac{\mathrm{d}c_A}{\mathrm{d}t} &= k_{-1} \cdot c_B - k_1 \cdot c_A \\ \frac{\mathrm{d}c_B}{\mathrm{d}t} &= k_1 \cdot c_A - k_{-1} \cdot c_B. \end{aligned} \tag{F.2}$$

I detta system är k_1 och k_{-1} okända, vi kallar dessa θ_1 respektive θ_2. Genom att sätta ett värde på dessa parametrar och sätta startkoncentrationer $c_{A,0}$ och $c_{B,0}$ kan vi generera en datapunkt vid en tid tidpunkt som avviker normalfördelat från den sanna lösningen med en viss varians.

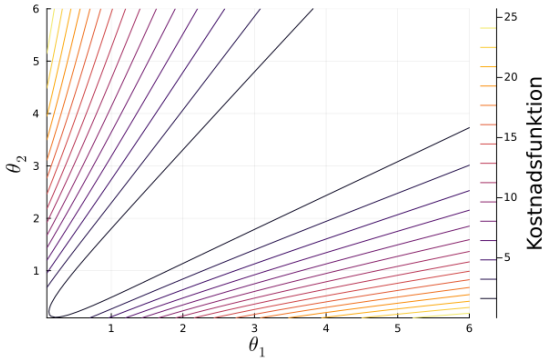
För att testa vår optimerare sätter vi

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \tag{F.3}$$

där vi genererar 50 experiment där startkoncentrationer $c_{A,0}$ och $c_{B,0}$ slumpas mellan 0 och 1 och experimenten körs till en slumpad tid mellan 0 och 1 sekunder. Den sanna lösningen för ett specifikt starttillstånd visas i Figur F.1.



Figur F.1: Exakta lösningen för testmodellen då $c_{A,0} = 0.5$ och $c_{B,0} = 0$



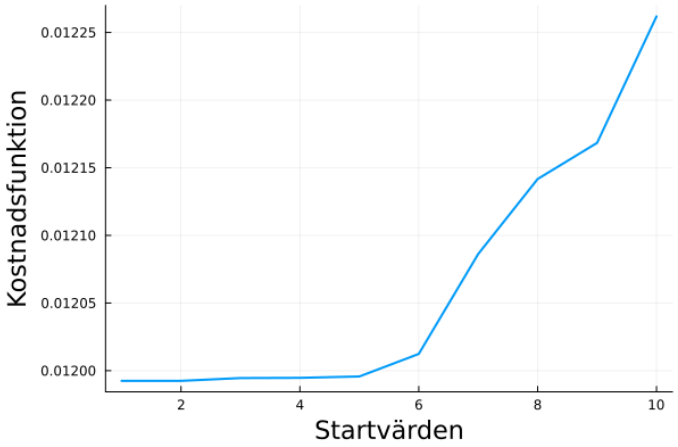
Figur F.2: Konturplot för kostnadsfunktionen i området är $0.01 \geq \theta_1 \geq 6$ och $0.01 \geq \theta_2 \geq 6$

Sökområdet är $0.01 \geq \theta_1 \geq 6$ och $0.01 \geq \theta_2 \geq 6$ och optimeringen körs på 10 genererade startvärden med varians 0.01 enligt latin hypercube-sampling.

Optimeringen erhåller

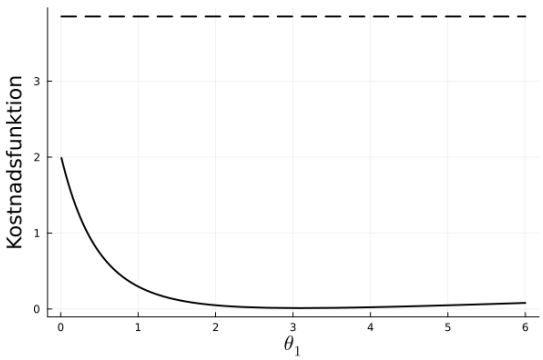
$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 3.091638162900869 \\ 3.1068427019656415 \end{bmatrix}, \tag{F.4}$$

vilket ligger nära de parametrar som den genererade data utgått från. Vi väntar oss att de estimerade parametrarna inte blir identiska till de sanna parametrarna då den genererade datan avviker från den sanna lösningen. I Figur F.3 kan vi se de optimerade värdena från startpunkterna, och kan observera att startpunkterna konvergerar till ett minimivärde.

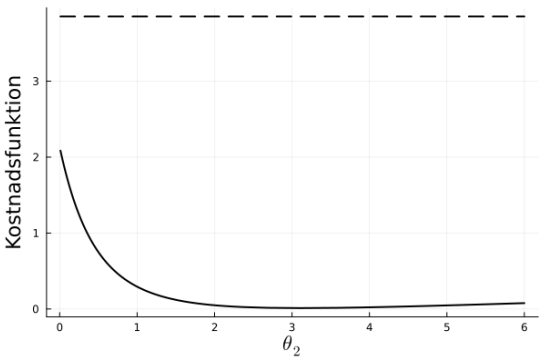


Figur F.3: Överblick över funktionsvärdet från optimeringarna från alla 10 startpunkter.

Figur F.4 och Figur F.5 visar en profile likelihood för den första respektive andra parametern. Figurerna har liknande form och ingen av graferna går tydligt uppåt till vänster om minimivärdet vilket innebär de är praktiskt icke-identifierbara. Orsaken till detta är att parametrarna är korrelerade och kan därför inte definieras var för sig. Däremot kan förhållandet mellan parametrarna fastställas (Fig. F.2). Även konturplotten demonstrerar parametrarnas icke-identifierbarhet då minimipunkten inte är innesluten. Anledningen till att parametrarna kan uppskattas är på grund av att parametrarnas värde påverkar hur snabbt jämviktsläget infinner sig, vilket den genererade experimentella datan reflekterar.



Figur F.4: Profilering av kostnadsfunktionen för θ_1, tröskelvärdet Δ_α ges av den streckade linjen.



Figur F.5: Profilering av kostnadsfunktionen för θ_2, tröskelvärdet Δ_α ges av den streckade linjen.

G Appendix 7 - Källkod

I nedstående avsnitt ges källkoden för testmodellen samt den riktiga modellen. Det ingår även annan kod som ingår i parameterestimationen så som en optimerare samt kod för att plotta erhållna resultat.

G.1 Modellkonstruktion av testmodell med grundkod för parameterestimation

```
using DifferentialEquations, ModelingToolkit, Plots, Random, Distributions
include("newton_minimize.jl")
include("profile_likelihood.jl")

# Object for experimental results
struct experiment_results
    c0::AbstractVector
    c_final::AbstractVector
    t_final::Number
end

"Model construction"
function model_2p_initialize()
    @parameters t    [1:2]      #Parametrar i modellen
    @variables c1(t) c2(t)   #Variabler i modellen
    D = Differential(t) #Definierar tecken för derivata

    equation_system = [D(c1) ~ -  [1] * c1 +   [2] * c2,
        D(c2) ~   [1] * c1 -   [2] * c2]  #Uttryck för systemet som
    differentialekvationer

    @named system = ODESystem(equation_system) #Definierar av som är
    systemet från diffrentialekvationerna
    system = structural_simplify(system) #Skriver om systemet så det blir lö
    sbart

    # Intialvärden som kommer skrivas över
    c0 = [0, 0]
     in  = [0, 0]

    u0 = [c1 => c0[1],
        c2 => c0[2]] #Definierar initialvärden

    p = [  [1] =>  in [1],
          [2] =>  in [2]] #Definierar värden för parametrarna

    tspan = (0.0, 10) #Tiden vi kör modellen under
    problem_object = ODEProblem(system, u0, tspan, p, jac=true)  #Definierar
     vad som ska beräknas
    return problem_object
end

"Solve the ODE system"
function model_solver(_problem_object,  in , c0, t_stop)
    problem_object = remake(_problem_object, u0=convert.(eltype( in ), c0),
    tspan=(0.0, t_stop), p= in )
    solution = solve(problem_object, Rodas5P(), abstol=1e-8, reltol=1e-8,
    maxiters=1e5)
    return solution
```

```
44  end
45
46  "Catch error"
47  function check_error(e)
48      if e isa BoundsError
49          @warn "Bounds error ODE solve"
50      elseif e isa DomainError
51          @warn "Domain error on ODE solve"
52      elseif e isa SingularException
53          @warn "Singular exeption on ODE solve"
54      else
55          rethrow(e)
56      end
57  end
58
59  "Calculate difference between experiments and model"
60  function cost_function(problem_object, log  , experimental_data::
        AbstractVector)
61         = exp.(log  )
62      error = 0
63      for data in experimental_data
64          success = true
65          try
66              sol = model_solver(problem_object,    , data.c0, data.t_final)
67              if !(sol.retcode == ReturnCode.Success || sol.retcode ==
        ReturnCode.Terminated)
68                  success = false
69              end
70              if success
71                  c_final_model = sol.u[end]
72                  error += sum((c_final_model - data.c_final) .^ 2)
73              end
74          catch e
75              check_error(e)
76              success == false
77          end
78          if success == false
79              return Inf
80          end
81      end
82      return error
83  end
84
85
86  "Run experiment"
87  function experimenter(problem_object, t_stop, c0, standard_deviation,  in )
88      solution = model_solver(problem_object,  in , c0, t_stop) #Genererar lö
        sningar
89      noise_distribution = Normal(0, standard_deviation) #Skapar error
90      return solution[:, end] + rand(noise_distribution, length(c0)) # Lägger
        till error
91  end
92
93  "Generate experimental data"
94  function random_dataset_generator(problem_object, number_of_experiments,
         in ; standard_deviation=0.01)
95      experimental_data = []
96      for i = 1:number_of_experiments
97          Random.seed!(10 * i)
98          t_final_data = rand() #Genererar slumpmässiga sluttider
```

```
99          c0_data = rand!(zeros(length(problem_object.u0))) #Genererar slumpmässiga intial koncentrationer
100         c_final_data = experimenter(problem_object, t_final_data, c0_data, standard_deviation, in )
101         current_data = experiment_results(c0_data, c_final_data, t_final_data)
102         push!(experimental_data, current_data)
103     end
104     return experimental_data
105 end
106
107 "Plot true solution"
108 function plot_exact_example(problem_object, in )
109     c0 = [0.5, 0.0] # Initial concentrations
110     sol = model_solver(problem_object, in , c0, 1) #Kör modellen
111     plot(sol, xaxis="Tid (s)", yaxis="Koncentration", label=[L"c_{A}" L"c_{B}"], lw=2, legendfontsize=15, labelfontsize=15) #Plottar lösningen
112     savefig("exact_ex.png")
113 end
114
115 "Plot model"
116 function plot_experiment(experimental_data)
117     for data in experimental_data
118         plot!(data.t_final * ones(length(data.c_final)), data.c_final, seriestype=:scatter) #Plottar lösningen
119     end
120     savefig("exp_data.png")
121 end
122
123 problem_object = model_2p_initialize()
124 experimental_data = random_dataset_generator(problem_object, 50, [3.0, 3.0])
125
126 bounds = [(0.01, 6), (0.01, 6)]
127 log_bounds = map(x -> (log(x[1]), log(x[2])), bounds)
128
129 f(x) = cost_function(problem_object, x, experimental_data)
130
131 # run the parameter estimation
132 x_min, f_min = p_est(f, log_bounds, 10, false)
133
134 plot_exact_example(problem_object, [3.0, 3.0])
135 plot_experiment(experimental_data)
136
137 # Define the initial parameter values
138 params = x_min
139
140 # Perform profile likelihood analysis for each parameter
141 num_points = 100
142 threshold = 3.84
143
144 # save threshold
145 CSV.write("profilelikelihood_results/threshold.csv", DataFrame(threshold=threshold))
146
147 run_profile_likelihood(params, 1000, bounds, num_points, threshold)
148
149 contourplot_2p()
```

G.2 Modellkonstruktion och grundkod för parameterestimation

```
using DifferentialEquations
using ModelingToolkit
using Plots
using Random
using Distributions
using ForwardDiff
using DataFrames
using CSV
include("newton_minimize.jl")
include("profile_likelihood.jl")

"Object for experimental results"
struct experiment_results
    glucose_conc::Number
    hxt_types::AbstractVector
    c::AbstractMatrix
    t::AbstractVector
end

Data01_glucose = [0.74 0.1 0.06 0.05 0.76 0.13 23.02 26.98
    1.83 0.52 0.1 0.06 0.85 0.33 29.55 36.75
    0.23 0.02 0.29 0.02 0.95 0.05 41.21 53.44
    0.08 0.05 0.19 0.29 0.24 0.14 31.34 44.63
    0.05 0.19 0.15 1.1 0.32 0.18 28.44 25.82
    0.09 0.03 0.06 0.07 0.27 0.09 12.34 16.08
    0.02 0.02 0.01 0.2 0.28 0.04 9.92 13.89]
Data02_glucose = [0.07 0.02 0.06 0.05 0.52 0.13 23.02 26.98
    3.92 1.05 0.09 0.09 0.78 0.71 21.19 24.59
    9.86 8.15 0.18 0.18 0.67 0.57 19.69 20.69
    19.08 15.12 0.32 0.32 1.78 1.46 16.35 17.33
    21.03 20.01 0.35 0.35 3.96 2.96 15.57 14.37
    27.03 24.11 0.91 1.18 7.34 8.34 12.55 13.55
    29.03 31.05 1.03 1.08 11.16 9.16 10.37 8.12]
Data01_mutant = [91.09 81.19
    82.14 74.49
    69.08 68.11
    59.57 64.89
    55.71 56.12
    51.68 60.08]
Data02_mutant = [23.83 20.11
    41.84 34.41
    57.87 51.86
    76.11 65.38
    85.12 77.12
    92.12 81.11]

"Take the avrages of repeted experiments"
function new_data_maker(Data_old)
    size_matrix = size(Data_old)
    println(size_matrix)
    data_new = zeros(size_matrix[1], trunc(Int, size_matrix[2] ./ 2))
    for i = 1:trunc(Int, size_matrix[2] ./ 2)
        global data_new[:, i] = (Data_old[:, 2*i-1] + Data_old[:, 2*i]) / 2
    end
    return data_new
end

# For non avrage data
#index_general = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
#index_mutant = [4, 4]


# For avrage data
index_general = [1, 2, 3, 4]
index_mutant = [4]
Data01_glucose = new_data_maker(Data01_glucose)
Data02_glucose = new_data_maker(Data02_glucose)
Data01_mutant = new_data_maker(Data01_mutant)
Data02_mutant = new_data_maker(Data02_mutant)


timevalues_general = [0.0, 10.0, 20.0, 30.0, 40.0, 60.0, 120.0]
timevalues_mutant = [0.0, 10.0, 27.0, 35.0, 60.0, 120.0]

experiment1 = experiment_results(3.346e8, index_general, Data01_glucose,
    timevalues_general)
experiment2 = experiment_results(6.685e8, index_general, Data02_glucose,
    timevalues_general)

experiment3 = experiment_results(3.346e8, index_mutant, Data01_mutant,
    timevalues_mutant)
experiment4 = experiment_results(3.346e8, index_mutant, Data02_mutant,
    timevalues_mutant)

#If with mutant data
experimental_data = [experiment1, experiment2, experiment3, experiment4]

#If without mutant data
experimental_data = [experiment1, experiment2]

"Constructs the model with 11 parameters
return a problem_object"
function model_initialize()
    @parameters t Extracellular_glucose k_a_Snf3 k_i_Snf3g k_i_Std1
    K_Std1_Rgt1 k_i_Mth1 K_Mth1_Rgt1 k_p_ATP k_i_Snf1 k_i_Mig1 T_mHXT1
       _activation    controller_Rgt1 controller_Mig2
    @variables Snf3(t) Snf3g(t) Std1(t) Mth1(t) Rgt1_active(t) mSNF3(t)
    mSTD1(t) mMTH1(t) mRGT1(t) mHXT1(t) Hxt1(t) mHXT2(t) Hxt2(t) mHXT3(t)
    Hxt3(t) mHXT4(t) Hxt4(t) mSNF1(t) Snf1(t) Cellular_glucose(t) mMIG1(t)
    Mig1(t) mMIG2(t) Mig2(t) Rgt1(t)
    D = Differential(t)

    k_t_Snf3 = 0.010 #From RGT2!!
    k_d_Snf3 = 0.231 #From RGT2!!
    k_t_Std1 = 42.8
    k_d_Std1 = 0.087
    k_t_Mth1 = 6.000
    k_d_Mth1 = 0.025
    k_t_Rgt1 = 19.000
    k_d_Rgt1 = 0.050

    k_t_Hxt1 = 1.480
    k_t_Hxt2 = 4.220
    k_t_Hxt3 = 4.230
    k_t_Hxt4 = 1.530
    k_d_Hxt1 = 0.010
    k_d_Hxt2 = 0.010
    k_d_Hxt3 = 0.010
    k_d_Hxt4 = 0.010
    k_t_Snf1 = 0.160
```

```
k_d_Snf1 = 0.020
k_t_Mig1 = 62.000
k_d_Mig1 = 0.020
k_t_Mig2 = 6.000
k_d_Mig2 = 0.046
V_transport_Hxt1 = 4.14 * 10^20
K_transport_Hxt1 = 5.40 * 10^22
V_transport_Hxt2 = 5.82 * 10^19
K_transport_Hxt2 = 9.00 * 10^20
V_transport_Hxt3 = 2.16 * 10^20
K_transport_Hxt3 = 3.30 * 10^22
V_transport_Hxt4 = 9.60 * 10^19
K_transport_Hxt4 = 5.58 * 10^21


V_mSNF3 = 50
  _Mig1_Snf3  = 0.000  #Not enough decimals
  _Mig2_Snf3  = 0.010
V_mSTD1 = 0.040
   _Rgt1_active_Std1   = 0.050
V_mMTH1 = 0.170
  _Rgt1_active_MTH1  = 0.030
 _Mig1_MTH1  = 0.460
 _Mig2_MTH1  = 0.001
V_mRGT1 = 1.000

V_mHXT1 = 2.56
  _Rgt1_active_HXT1  = 5.00 * 10^-001
V_mHXT2 = 1.430
  _Rgt1_active_HXT2  = 0.450
 _Mig1_HXT2  = 0.110
 _Mig2_HXT2  = 0.010
V_mHXT3 = 2.350
  _Rgt1_active_HXT3  = 0.240
 _Mig1_HXT3  = 0.020
 _Mig2_HXT3  = 0.001
V_mHXT4 = 34.200
  _Rgt1_active_HXT4  = 0.026
 _Mig1_HXT4  = 0.430
 _Mig2_HXT4  = 0.080

V_mMIG1 = 0.020
  _Mig1_MIG1  = 0.020
  _Mig2_MIG1  = 0.000 #Not enough decimals
V_mMIG2 = 0.230
   _Rgt1_active_MIG2   = 0.100
  _Mig1_MIG2  = 0.001
  _Mig2_MIG2  = 0.010
V_mSNF1 = 2.900

k_d_mHXT1 = 0.03
k_d_mHXT2 = 0.03
k_d_mHXT3 = 0.03
k_d_mHXT4 = 0.06
k_d_mSNF3 = 0.02
k_d_mMIG1 = 0.04
k_d_mMIG2 = 0.04
k_d_mMTH1 = 0.04
k_d_mSTD1 = 0.01
k_d_mRGT1 = 0.04
k_d_mSNF1 = 0.04
```

```
173
174
175     equation_system = [D(Snf3) ~ k_t_Snf3 * mSNF3 - k_d_Snf3 * Snf3 -
        k_a_Snf3 * Snf3 * Extracellular_glucose + k_i_Snf3g * Snf3g,
176         D(Snf3g) ~ k_a_Snf3 * Snf3 * Extracellular_glucose - k_i_Snf3g *
        Snf3g,
177         D(Std1) ~ k_t_Std1 * mSTD1 - k_d_Std1 * Std1 - k_i_Std1 * Std1 *
        Snf3g,
178         D(Mth1) ~ k_t_Mth1 * mMTH1 - k_d_Mth1 * Mth1 - k_i_Mth1 * Mth1 *
        Snf3g,
179         D(Rgt1) ~ k_t_Rgt1 * mRGT1 - k_d_Rgt1 * Rgt1,
180         Rgt1_active ~ K_Std1_Rgt1 * Std1 * Rgt1 + K_Mth1_Rgt1 * Mth1 * Rgt1,
181
182         D(Hxt1) ~ k_t_Hxt1 * mHXT1 - k_d_Hxt1 * Hxt1,
183         D(Hxt2) ~ k_t_Hxt2 * mHXT2 - k_d_Hxt2 * Hxt2,
184         D(Hxt3) ~ k_t_Hxt3 * mHXT3 - k_d_Hxt3 * Hxt3,
185         D(Hxt4) ~ k_t_Hxt4 * mHXT4 - k_d_Hxt4 * Hxt4,
186         D(Snf1) ~ k_t_Snf1 * mSNF1 - k_d_Snf1 * Snf1 - k_i_Snf1 * Snf1 *
        Cellular_glucose,
187         D(Mig1) ~ k_t_Mig1 * mMIG1 - k_d_Mig1 * Mig1 - k_i_Mig1 * Mig1 *
        Snf1,
188         D(Mig2) ~ k_t_Mig2 * mMIG2 - k_d_Mig2 * Mig2,
189         D(Cellular_glucose) ~ V_transport_Hxt1 * Extracellular_glucose / (
        K_transport_Hxt1 + Extracellular_glucose) + V_transport_Hxt2 *
        Extracellular_glucose / (K_transport_Hxt2 + Extracellular_glucose) +
        V_transport_Hxt3 * Extracellular_glucose / (K_transport_Hxt3 +
        Extracellular_glucose) + V_transport_Hxt4 * Extracellular_glucose / (
        K_transport_Hxt4 + Extracellular_glucose) - k_p_ATP * Cellular_glucose,
190
191         D(mSNF3) ~ -k_d_mSNF3 * mSNF3 + V_mSNF3 / (1 +  _Mig1_Snf3  * Mig1)
        / (1 +  _Mig2_Snf3  * Mig2),
192         D(mSTD1) ~ -k_d_mSTD1 * mSTD1 + V_mSTD1 / (1 +  _Rgt1_active_Std1  *
         Rgt1_active),
193         D(mMTH1) ~ -k_d_mMTH1 * mMTH1 + V_mMTH1 / (1 +  _Rgt1_active_MTH1  *
         Rgt1_active) / (1 +  _Mig1_MTH1  * Mig1) / (1 +  _Mig2_MTH1  * Mig2),
194         D(mRGT1) ~ controller_Rgt1 * (-k_d_mRGT1 * mRGT1 + V_mRGT1),
195         D(mHXT1) ~ -k_d_mHXT1 * mHXT1 + V_mHXT1 * (T_mHXT1 + ((1 - T_mHXT1)
        *  _activation  * Rgt1) / (1 +  _activation  * Rgt1)) / (1 +
          _Rgt1_active_HXT1  * Rgt1_active),
196         D(mHXT2) ~ -k_d_mHXT2 * mHXT2 + V_mHXT2 / (1 +  _Rgt1_active_HXT2  *
         Rgt1_active) / (1 +  _Mig1_HXT2  * Mig1) / (1 +  _Mig2_HXT2  * Mig2),
197         D(mHXT3) ~ -k_d_mHXT3 * mHXT3 + V_mHXT3 / (1 +  _Rgt1_active_HXT3  *
         Rgt1_active) / (1 +  _Mig1_HXT3  * Mig1) / (1 +  _Mig2_HXT3  * Mig2),
198         D(mHXT4) ~ -k_d_mHXT4 * mHXT4 + V_mHXT4 / (1 +  _Rgt1_active_HXT4  *
         Rgt1_active) / (1 +  _Mig1_HXT4  * Mig1) / (1 +  _Mig2_HXT4  * Mig2), D(
        mMIG1) ~ -k_d_mMIG1 * mMIG1 + V_mMIG1 / (1 +  _Mig1_MIG1  * Mig1) / (1 +
         _Mig2_MIG1  * Mig2),
199         D(mMIG2) ~ controller_Mig2 * (-k_d_mMIG2 * mMIG2 + V_mMIG2 / (1 +
          _Rgt1_active_MIG2  * Rgt1_active) / (1 +  _Mig1_MIG2  * Mig1) / (1 +
         _Mig2_MIG2  * Mig2)),
200         D(mSNF1) ~ -k_d_mSNF1 * mSNF1 + V_mSNF1]
201
202     @named system = ODESystem(equation_system)
203     system = structural_simplify(system)
204
205     c0 = zeros(24)
206      in  = zeros(14)
207
208     u0 = [
209         Snf3 => c0[1],
210         Snf3g => c0[2],
```

```
211           Std1 => c0[3],
212           Mth1 => c0[4],
213           Hxt1 => c0[5],
214           Hxt2 => c0[6],
215           Hxt3 => c0[7],
216           Hxt4 => c0[8],
217           Snf1 => c0[9],
218           Cellular_glucose => c0[10],
219           Mig1 => c0[11],
220           Mig2 => c0[12],
221           Rgt1 => c0[13],
222           mSNF3 => c0[14],
223           mSTD1 => c0[15],
224           mMTH1 => c0[16],
225           mRGT1 => c0[17],
226           mHXT1 => c0[18],
227           mHXT2 => c0[19],
228           mHXT3 => c0[20],
229           mHXT4 => c0[21],
230           mSNF1 => c0[22],
231           mMIG1 => c0[23],
232           mMIG2 => c0[24]]
233
234       p = [k_a_Snf3 =>  in [1],
235           k_i_Snf3g =>  in [2],
236           k_i_Std1 =>  in [3],
237           K_Std1_Rgt1 =>  in [4],
238           k_i_Mth1 =>  in [5],
239           K_Mth1_Rgt1 =>  in [6],
240           k_p_ATP =>  in [7],
241           k_i_Snf1 =>  in [8],
242           k_i_Mig1 =>  in [9],
243           T_mHXT1 =>  in [10],
244              _activation   =>  in [11],
245           Extracellular_glucose =>  in [12],
246           controller_Rgt1 =>  in [13],
247           controller_Mig2 =>  in [14]]
248
249       CSV.write("p_est_results/C_order.csv", DataFrame(index=collect(1:24), C=
          states(system)))
250       CSV.write("p_est_results/param_order.csv", DataFrame(index=collect(1:14)
          , C=parameters(system)))
251
252       tspan = (0.0, 10)
253       problem_object = ODEProblem(system, u0, tspan, p)
254       return problem_object, system
255   end
256
257   "Constructs the model with 13 parameters.
258   return a problem_object"
259   function model_initialize_big()
260       @parameters t Extracellular_glucose k_a_Snf3 k_i_Snf3g k_i_Std1
          K_Std1_Rgt1 k_i_Mth1 K_Mth1_Rgt1 k_p_ATP k_i_Snf1 k_i_Mig1 T_mHXT1
             _activation    controller_Rgt1 controller_Mig2    _Mig1_Snf3    _Mig2_MIG1
261       @variables Snf3(t) Snf3g(t) Std1(t) Mth1(t) Rgt1_active(t) mSNF3(t)
          mSTD1(t) mMTH1(t) mRGT1(t) mHXT1(t) Hxt1(t) mHXT2(t) Hxt2(t) mHXT3(t)
          Hxt3(t) mHXT4(t) Hxt4(t) mSNF1(t) Snf1(t) Cellular_glucose(t) mMIG1(t)
          Mig1(t) mMIG2(t) Mig2(t) Rgt1(t) #Variabler i modellen
262       D = Differential(t)
263
264       k_t_Snf3 = 0.010 #from RGT2!!
```

```
k_d_Snf3 = 0.231 #from RGT2!!
k_t_Std1 = 42.8
k_d_Std1 = 0.087
k_t_Mth1 = 6.000
k_d_Mth1 = 0.025
k_t_Rgt1 = 19.000
k_d_Rgt1 = 0.050

k_t_Hxt1 = 1.480
k_t_Hxt2 = 4.220
k_t_Hxt3 = 4.230
k_t_Hxt4 = 1.530
k_d_Hxt1 = 0.010
k_d_Hxt2 = 0.010
k_d_Hxt3 = 0.010
k_d_Hxt4 = 0.010
k_t_Snf1 = 0.160
k_d_Snf1 = 0.020
k_t_Mig1 = 62.000
k_d_Mig1 = 0.020
k_t_Mig2 = 6.000
k_d_Mig2 = 0.046
V_transport_Hxt1 = 4.14 * 10^20
K_transport_Hxt1 = 5.40 * 10^22
V_transport_Hxt2 = 5.82 * 10^19
K_transport_Hxt2 = 9.00 * 10^20
V_transport_Hxt3 = 2.16 * 10^20
K_transport_Hxt3 = 3.30 * 10^22
V_transport_Hxt4 = 9.60 * 10^19
K_transport_Hxt4 = 5.58 * 10^21


V_mSNF3 = 50
#  _Mig1_Snf3   = 0.000  #Not enough decimals
   _Mig2_Snf3   = 0.010
V_mSTD1 = 0.040
    _Rgt1_active_Std1    = 0.050
V_mMTH1 = 0.170
   _Rgt1_active_MTH1   = 0.030
  _Mig1_MTH1  = 0.460
  _Mig2_MTH1  = 0.001
V_mRGT1 = 1.000

V_mHXT1 = 2.56
   _Rgt1_active_HXT1   = 5.00 * 10^-001
V_mHXT2 = 1.430
   _Rgt1_active_HXT2   = 0.450
  _Mig1_HXT2  = 0.110
  _Mig2_HXT2  = 0.010
V_mHXT3 = 2.350
   _Rgt1_active_HXT3   = 0.240
  _Mig1_HXT3  = 0.020
  _Mig2_HXT3  = 0.001
V_mHXT4 = 34.200
   _Rgt1_active_HXT4   = 0.026
  _Mig1_HXT4  = 0.430
  _Mig2_HXT4  = 0.080


V_mMIG1 = 0.020
  _Mig1_MIG1  = 0.020
```

```
# _Mig2_MIG1  = 0.000 #Not enough decimals
V_mMIG2 = 0.230
  _Rgt1_active_MIG2   = 0.100
 _Mig1_MIG2  = 0.001
 _Mig2_MIG2  = 0.010
V_mSNF1 = 2.900

k_d_mHXT1 = 0.03
k_d_mHXT2 = 0.03
k_d_mHXT3 = 0.03
k_d_mHXT4 = 0.06
k_d_mSNF3 = 0.02
k_d_mMIG1 = 0.04
k_d_mMIG2 = 0.04
k_d_mMTH1 = 0.04
k_d_mSTD1 = 0.01
k_d_mRGT1 = 0.04
k_d_mSNF1 = 0.04


equation_system = [D(Snf3) ~ k_t_Snf3 * mSNF3 - k_d_Snf3 * Snf3 -
k_a_Snf3 * Snf3 * Extracellular_glucose + k_i_Snf3g * Snf3g,
    D(Snf3g) ~ k_a_Snf3 * Snf3 * Extracellular_glucose - k_i_Snf3g *
Snf3g,
    D(Std1) ~ k_t_Std1 * mSTD1 - k_d_Std1 * Std1 - k_i_Std1 * Std1 *
Snf3g,
    D(Mth1) ~ k_t_Mth1 * mMTH1 - k_d_Mth1 * Mth1 - k_i_Mth1 * Mth1 *
Snf3g,
    D(Rgt1) ~ k_t_Rgt1 * mRGT1 - k_d_Rgt1 * Rgt1,
    Rgt1_active ~ K_Std1_Rgt1 * Std1 * Rgt1 + K_Mth1_Rgt1 * Mth1 * Rgt1,


    D(Hxt1) ~ k_t_Hxt1 * mHXT1 - k_d_Hxt1 * Hxt1,
    D(Hxt2) ~ k_t_Hxt2 * mHXT2 - k_d_Hxt2 * Hxt2,
    D(Hxt3) ~ k_t_Hxt3 * mHXT3 - k_d_Hxt3 * Hxt3,
    D(Hxt4) ~ k_t_Hxt4 * mHXT4 - k_d_Hxt4 * Hxt4,
    D(Snf1) ~ k_t_Snf1 * mSNF1 - k_d_Snf1 * Snf1 - k_i_Snf1 * Snf1 *
Cellular_glucose,
    D(Mig1) ~ k_t_Mig1 * mMIG1 - k_d_Mig1 * Mig1 - k_i_Mig1 * Mig1 *
Snf1,
    D(Mig2) ~ k_t_Mig2 * mMIG2 - k_d_Mig2 * Mig2,
    D(Cellular_glucose) ~ V_transport_Hxt1 * Extracellular_glucose / (
K_transport_Hxt1 + Extracellular_glucose) + V_transport_Hxt2 *
Extracellular_glucose / (K_transport_Hxt2 + Extracellular_glucose) +
V_transport_Hxt3 * Extracellular_glucose / (K_transport_Hxt3 +
Extracellular_glucose) + V_transport_Hxt4 * Extracellular_glucose / (
K_transport_Hxt4 + Extracellular_glucose) - k_p_ATP * Cellular_glucose,

    #mRNA
    D(mSNF3) ~ -k_d_mSNF3 * mSNF3 + V_mSNF3 / (1 +  _Mig1_Snf3  * Mig1)
/ (1 +  _Mig2_Snf3  * Mig2),
    D(mSTD1) ~ -k_d_mSTD1 * mSTD1 + V_mSTD1 / (1 +  _Rgt1_active_Std1  *
 Rgt1_active),
    D(mMTH1) ~ -k_d_mMTH1 * mMTH1 + V_mMTH1 / (1 +  _Rgt1_active_MTH1  *
 Rgt1_active) / (1 +  _Mig1_MTH1  * Mig1) / (1 +  _Mig2_MTH1  * Mig2),
    D(mRGT1) ~ controller_Rgt1 * (-k_d_mRGT1 * mRGT1 + V_mRGT1),
    D(mHXT1) ~ -k_d_mHXT1 * mHXT1 + V_mHXT1 * (T_mHXT1 + ((1 - T_mHXT1)
*  _activation  * Rgt1) / (1 +  _activation  * Rgt1)) / (1 +
 _Rgt1_active_HXT1  * Rgt1_active),
    D(mHXT2) ~ -k_d_mHXT2 * mHXT2 + V_mHXT2 / (1 +  _Rgt1_active_HXT2  *
 Rgt1_active) / (1 +  _Mig1_HXT2  * Mig1) / (1 +  _Mig2_HXT2  * Mig2),
```

```
370         D(mHXT3) ~ -k_d_mHXT3 * mHXT3 + V_mHXT3 / (1 + _Rgt1_active_HXT3 *
        Rgt1_active) / (1 + _Mig1_HXT3 * Mig1) / (1 + _Mig2_HXT3 * Mig2),
371         D(mHXT4) ~ -k_d_mHXT4 * mHXT4 + V_mHXT4 / (1 + _Rgt1_active_HXT4 *
        Rgt1_active) / (1 + _Mig1_HXT4 * Mig1) / (1 + _Mig2_HXT4 * Mig2), D(
        mMIG1) ~ -k_d_mMIG1 * mMIG1 + V_mMIG1 / (1 + _Mig1_MIG1 * Mig1) / (1 +
        _Mig2_MIG1 * Mig2),
372         D(mMIG2) ~ controller_Mig2 * (-k_d_mMIG2 * mMIG2 + V_mMIG2 / (1 +
        _Rgt1_active_MIG2 * Rgt1_active) / (1 + _Mig1_MIG2 * Mig1) / (1 +
        _Mig2_MIG2 * Mig2)),
373         D(mSNF1) ~ -k_d_mSNF1 * mSNF1 + V_mSNF1]
374
375     @named system = ODESystem(equation_system)
376     system = structural_simplify(system)
377
378     c0 = zeros(24)
379      in  = zeros(16)
380
381     u0 = [
382         Snf3 => c0[1],
383         Snf3g => c0[2],
384         Std1 => c0[3],
385         Mth1 => c0[4],
386         Hxt1 => c0[5],
387         Hxt2 => c0[6],
388         Hxt3 => c0[7],
389         Hxt4 => c0[8],
390         Snf1 => c0[9],
391         Cellular_glucose => c0[10],
392         Mig1 => c0[11],
393         Mig2 => c0[12],
394         Rgt1 => c0[13],
395         mSNF3 => c0[14],
396         mSTD1 => c0[15],
397         mMTH1 => c0[16],
398         mRGT1 => c0[17],
399         mHXT1 => c0[18],
400         mHXT2 => c0[19],
401         mHXT3 => c0[20],
402         mHXT4 => c0[21],
403         mSNF1 => c0[22],
404         mMIG1 => c0[23],
405         mMIG2 => c0[24]]
406
407     p = [k_a_Snf3 =>  in [1],
408         k_i_Snf3g =>  in [2],
409         k_i_Std1 =>  in [3],
410         K_Std1_Rgt1 =>  in [4],
411         k_i_Mth1 =>  in [5],
412         K_Mth1_Rgt1 =>  in [6],
413         k_p_ATP =>  in [7],
414         k_i_Snf1 =>  in [8],
415         k_i_Mig1 =>  in [9],
416         T_mHXT1 =>  in [10],
417           _activation  =>  in [11],
418         Extracellular_glucose =>  in [12],
419         controller_Rgt1 =>  in [13],
420         controller_Mig2 =>  in [14],
421          _Mig1_Snf3  =>  in [14],
422          _Mig2_MIG1  =>  in [14]]
423
```

```
424     CSV.write("p_est_results/C_order.csv", DataFrame(index=collect(1:24), C=
        states(system)))
425     CSV.write("p_est_results/param_order.csv", DataFrame(index=collect(1:16)
        , C=parameters(system)))
426
427     tspan = (0.0, 10)
428     problem_object = ODEProblem(system, u0, tspan, p)
429     return problem_object, system
430 end
431
432 "Gives a solution over time
433 with points at t_stop_points"
434 function model_solver(_problem_object::ODEProblem,  in ::AbstractVector, c0
        ::AbstractVector, t_stop::Number)
435     problem_object = remake(_problem_object, u0=convert.(eltype( in ), c0),
        tspan=(0.0, t_stop), p= in )
436     sol = solve(problem_object, Rodas5P(), abstol=1e-8, reltol=1e-8; verbose
        =false)
437     return sol
438 end
439
440 "Condition to terminate pre_equilibrium. Happens when gradient is flat
        enough meaning steady-state is reached"
441 function terminate_condition(u, t, integrator)
442     abstol = 1e-8 / 100
443     reltol = 1e-8 / 100
444
445     dudt = DiffEqBase.get_du(integrator)
446     valCheck = sqrt(sum((dudt ./ (reltol * integrator.u .+ abstol)) .^ 2) /
        length(u))
447     return valCheck < 1.0
448 end
449
450 "Terminates pre_equilibrium"
451 function terminate_affect!(integrator)
452     terminate!(integrator)
453 end
454
455 "Calculates steady-state concentrations"
456 function ss_conc_calc(_problem_object::ODEProblem,  in ::AbstractVector, c0
        ::AbstractVector)
457     t_maximum = 10000 #Maximala tid att nå maximum
458     problem_object = remake(_problem_object, u0=convert.(eltype( in ), c0),
        tspan=(0.0, t_maximum), p= in )
459     cb = DiscreteCallback(terminate_condition, terminate_affect!)
460     sol = solve(problem_object, callback=cb, Rodas5P(), abstol=1e-8, reltol
        =1e-8; verbose=false)
461     if sol.retcode     :Success && sol.retcode     :Terminated
462         @warn "Failed solving ss ODE, reason: $(sol.retcode)" maxlog = 10
463         return Inf
464     end
465     if sol.t == t_maximum
466         @warn "Did not reach steady-state in pre_equilibrium in alloted time
        " maxlog = 10
467         return Inf
468     end
469     return sol.u[end]
470 end
471
472 "Linear interpolation"
473 function interpolate(t::Number, f1, f2, t1, t2)
```

```
474     return (f2 - f1) ./ (t2 - t1) .* (t - t1) + f1
475 end
476
477 "For some error checking in ODE:solver"
478 function check_extra_error(e)
479     if e isa BoundsError
480         @warn "Bounds error ODE solve"
481     elseif e isa DomainError
482         @warn "Domain error on ODE solve"
483     elseif e isa SingularException
484         @warn "Singular exeption on ODE solve"
485     else
486         rethrow(e)
487     end
488 end
489
490 "Calculate difference between experiments and model"
491 function cost_function(problem_object, log  , experimental_data::
    AbstractVector,
492     index_first_Hxt=6, index_glucose=3, index_controller_Rgt1=11,
    index_controller_Mig2=14)
493
494        = exp.( log  )
495       _type  = eltype(  )
496
497     zero_typefix = convert.( _type , 0)
498     one_typefix = convert.( _type , 1)
499
500     insert!(  , index_glucose, zero_typefix)
501     insert!(  , index_controller_Rgt1, one_typefix)
502     #insert!(  , index_controller_Mig2, one_typefix)
503     append!(  , one_typefix)
504
505     error = 0
506     c_eq_store = []
507     for (i, experiment) in enumerate(experimental_data)
508         try
509             c_eq = [1]
510             if i == 2
511                 c_eq = c_eq_store
512             else
513                 if i == 3
514                       [index_controller_Rgt1] = zero_typefix
515                       [index_controller_Mig2] = one_typefix
516                 else
517                     i == 4
518                       [index_controller_Rgt1] = one_typefix
519                       [index_controller_Mig2] = zero_typefix
520                 end
521
522                 global c_eq = ss_conc_calc(problem_object,    , zeros(24))
523                 if c_eq == Inf
524                     return Inf
525                 end
526
527                 if i == 1
528                     c_eq_store = c_eq
529                 end
530             end
531
532             if i == 1
```

```
533                 c_eq_store = c_eq
534             end
535 
536               [index_glucose] = convert.( _type , experiment.glucose_conc)
537             sol = model_solver(problem_object,    , c_eq, 120) #All
    experiments have end time 120
538             if sol.retcode     :Success
539                 if sol.retcode     :DtLessThanMin
540                     @warn "Failed solving ODE, reason: $(sol.retcode)"
    maxlog = 10
541                 end
542                 return Inf
543             end
544             for (index_time_data, t) in enumerate(experiment.t)
545                 index_time_model = convert.(Int64, findfirst(isone, sol.t
    .>= t))
546 
547                 if index_time_model == 1
548                     c_t = sol.u[1]
549                 else
550                     c_t = interpolate(t, sol.u[index_time_model-1], sol.u[
    index_time_model], sol.t[index_time_model-1], sol.t[index_time_model])
551                 end
552                 for index_hxt = experiment.hxt_types #Kika
553                     if i == 3 || i == 4
554                         error += sum((c_t[index_first_Hxt-1+4] - experiment.
    c[index_time_data, 1]) .^ 2)
555                     else
556                         #For without mutant
557                         #error += sum((c_t[index_first_Hxt-1+index_hxt] -
    experiment.c[index_time_data, index_hxt]) .^ 2) #Håll koll på så index
    (+5 blir rätt)
558 
559                         #For with mutant
560                         error += sum((c_t[index_first_Hxt-1+index_hxt] -
    experiment.c[index_time_data, ceil(Int, index_hxt / 2)]) .^ 2) #Håll koll
     på så index (+5 blir rätt)
561                     end
562                 end
563             end
564         catch e
565             check_extra_error(e)
566             return Inf
567         end
568     end
569     return error
570 end
571 
572 function timing_tests(problem_object, experimental_data, f)
573     #Solve one time first to fix compliation time
574     model_solver(problem_object, ones(12), zeros(26), 100)
575     cost_function(problem_object, zeros(11), experimental_data)
576     ForwardDiff.gradient(f, ones(11))
577     ForwardDiff.hessian(f, ones(11))
578 
579     time_model_solver = @elapsed model_solver(problem_object, ones(12),
    zeros(26), 100)
580     time_cost_function = @elapsed cost_function(problem_object, zeros(11),
    experimental_data)
581     time_gradient = @elapsed ForwardDiff.gradient(f, ones(11))
582     time_hessian = @elapsed ForwardDiff.hessian(f, ones(11))
```

```
583     data = DataFrame(Function=["model_solver", "cost_function", "gradient",
        "hessian"], time=[time_model_solver, time_cost_function, time_gradient,
        time_hessian])
584     CSV.write("ss_timer.csv", data)
585 end
586
587 "Timing operations for big modell"
588 function timing_tests_big(problem_object, experimental_data, f)
589     #Solve one time first to fix compliation time
590     model_solver(problem_object, ones(16), zeros(26), 100)
591     cost_function(problem_object, zeros(13), experimental_data)
592     ForwardDiff.gradient(f, ones(13))
593     ForwardDiff.hessian(f, ones(13))
594
595     time_model_solver = @elapsed model_solver(problem_object, ones(16),
        zeros(26), 100)
596     time_cost_function = @elapsed cost_function(problem_object, zeros(13),
        experimental_data)
597     time_gradient = @elapsed ForwardDiff.gradient(f, ones(13))
598     time_hessian = @elapsed ForwardDiff.hessian(f, ones(13))
599     data = DataFrame(Function=["model_solver", "cost_function", "gradient",
        "hessian"], time=[time_model_solver, time_cost_function, time_gradient,
        time_hessian])
600     println(time_model_solver)
601     println(time_cost_function)
602     println(time_gradient)
603     println(time_hessian)
604     CSV.write("ss_timer.csv", data)
605 end
606
607 function bounds_generator( _estimation )
608     #bounds = [(1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1
        e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e
        -3, 1e3)]
609     bounds = [(1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4
        ), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e
        -4, 1e4)]
610     newbounds = bounds
611     for i = 1:11
612         newbounds[i] = _estimation [i] .* bounds[i]
613     end
614     return newbounds
615 end
616
617 function bounds_generator_big( _estimation )
618     bounds = [(1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3
        ), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e-3, 1e3), (1e
        -3, 1e3), (1e-3, 1e3), (1e-3, 1e3)]
619     #bounds = [(1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1
        e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e-4, 1e4), (1e
        -4, 1e4), (1e-4, 1e4), (1e-4, 1e4)]
620     newbounds = bounds
621     for i = 1:13
622         newbounds[i] = _estimation [i] .* bounds[i]
623     end
624     return newbounds
625 end
626
627 "Plot ode and compare with data"
628 function plot_kinetic(_ ,experimental_data,
```

```
index_first_Hxt=6, index_glucose=3, index_controller_Rgt1=11,
index_controller_Mig2=14)
   = _
  _type  = eltype(  )

zero_typefix = convert.( _type , 0)
one_typefix = convert.( _type , 1)

insert!(  , index_glucose, zero_typefix)
insert!(  , index_controller_Rgt1, one_typefix)
insert!(  , index_controller_Mig2, one_typefix)


c_eq_store = []
for (i, experiment) in enumerate(experimental_data)
    c_eq = [1]
    if i == 2
        c_eq = c_eq_store
    else
        if i == 3
              [index_controller_Rgt1] = zero_typefix
              [index_controller_Mig2] = one_typefix
        else
            i == 4
              [index_controller_Rgt1] = one_typefix
              [index_controller_Mig2] = zero_typefix
        end

        global c_eq = ss_conc_calc(problem_object,   , zeros(24))
        if c_eq == Inf
            return Inf
        end

        if i == 1
            c_eq_store = c_eq
        end
    end
      [index_glucose] = convert.( _type , experiment.glucose_conc)
    sol = model_solver(problem_object,   , c_eq, 120) #All experiments
have end time 120
    if sol.retcode     :Success
        if sol.retcode     :DtLessThanMin
            @warn "Failed solving ODE, reason: $(sol.retcode)" maxlog =
10
        end
        return Inf
    end

    model_conc = transpose(Matrix(sol))
    if i == 1 || i==2
        plot(sol.t, model_conc[:,18], labels ="Modell mHXT1", linewidth
= 2, c=RGB(0.41, 0.82, 0.91), ylims=(0,22))
        plot!(sol.t, model_conc[:,19], labels ="Modell mHXT2", linewidth
 = 2, c=RGB(0.98, 0.41, 0))
        plot!(sol.t, model_conc[:,20], labels ="Modell mHXT3", linewidth
 = 2, c=RGB(177/255, 58/255, 105/255))
        plot!(sol.t, model_conc[:,21], labels ="Modell mHXT4", linewidth
 = 2, c=RGB(102/255, 188/255, 102/255))

        plot!(experiment.t, experiment.c[:,1], labels = "Data mHXT1",
seriestype=:scatter, c=RGB(0.41, 0.82, 0.91))
```

```
682             plot!(experiment.t, experiment.c[:,2], labels = "Data mHXT2",
    seriestype=:scatter, c=RGB(0.98, 0.41, 0))
683             plot!(experiment.t, experiment.c[:,3], labels = "Data  mHXT3",
    seriestype=:scatter, c=RGB(177/255, 58/255, 105/255))
684             plot!(experiment.t, experiment.c[:,4], labels = "Data  mHXT4",
    seriestype=:scatter, c=RGB(102/255, 188/255, 102/255))
685         else
686             plot(sol.t, model_conc[:,21], labels ="mHXT4", linewidth = 2, c=
    RGB(102/255, 188/255, 102/255), ylims=(0,80))
687             plot!(experiment.t, experiment.c, labels = ["mHXT4" "mHXT4"],
    seriestype=:scatter, c=RGB(102/255, 188/255, 102/255))
688         end
689         plot!(xlabel="Tid (s)", ylabel="Koncentration (molekyler/cell)",
    legend=:topright)
690         savefig("p_est_results/plot_ode_over_t$i")
691     end
692 end
693
694 # Avrages + Mutant + Big modell (-3, 3)*short_optim bounds
695 long_optim = [0.7969990063233143, 68.93424287588466, 6997.780815301634,
    0.01593630757371225, 17.286346860335527, 0.008671320875673462,
    0.00012446646232616555, 304.4640461729089, 111.99828602219864,
    0.00030830983134835766, 235.26237907052467, 0.0001357129836595192,
    23973.609058384074]
696
697
698 # 11 parameter modell
699 #=
700 problem_object, system = model_initialize()
701 bounds = bounds_generator(ones(11))
702 log_bounds = map(x -> (log(x[1]), log(x[2])), bounds)
703 f(x) = cost_function(problem_object, x, experimental_data) # 3 är index för
    glukos
704 timing_tests(problem_object, experimental_data, f) # run the parameter
    estimation
705 time = @elapsed x_min, f_min = p_est(f, log_bounds, 1000, false)
706 println("The optimization took: $time")
707 plot_kinetic(long_optim, experimental_data)
708 =#
709
710
711 # 13 parameter modell
712 problem_object, system = model_initialize_big()
713 bounds = bounds_generator_big(ones(13))
714 log_bounds = map(x -> (log(x[1]), log(x[2])), bounds)
715 f(x) = cost_function(problem_object, x, experimental_data, 18, 3, 12, 16) #
    3 är index för glukos
716 timing_tests_big(problem_object, experimental_data, f)
717 time = @elapsed x_min, f_min = p_est(f, log_bounds, 1000, false) # run the
    parameter estimation
718 println("The optimization took: $time")
719 plot_kinetic(long_optim, experimental_data, 18, 3, 12, 16)
720
721
722
723
724
725
726
727
728 #=
```

```
# Define the initial parameter values
params = x_min

# Perform profile likelihood analysis for each parameter
num_points = 100
threshold = 3.84

# save threshold
CSV.write("profilelikelihood_results/threshold.csv", DataFrame(threshold=
    threshold))

run_profile_likelihood(params, log_bounds, 50, num_points, threshold)
```

G.3 Optimeringsalgoritm

```
using ForwardDiff
using FiniteDifferences
using LinearAlgebra
using Distributions
using Random
using CSV
using DataFrames
using DelimitedFiles
using LinearAlgebra

"Search for the step size in the gradient direction dir used to find the next point for function f at point x"
function line_step_search(f::Function, x, dir; alpha=1.0)
    is_descent_direction::Bool = true

    # divide the start step size (alpha) until a function value less than that of the previous point is found
    for i in 1:50
        x_new = x + alpha * dir
        if f(x_new) == Inf
            alpha /= 2
            continue
        elseif f(x_new) < f(x) && i != 50
            break
        elseif i == 50
            is_descent_direction = false
        end

        alpha /= 2
    end

    return alpha, is_descent_direction
end

#Alternative method to find a point in a descending direction
function steepest_descent(f::Function, grad, x, log_bounds)
    dir = -grad / norm(grad)
    alpha, is_descent_direction = line_step_search(f, x, dir)

    x_old = x

    # calculate next point and ensure point is within bounds, project back if that is the case
    if is_descent_direction
        x += alpha * dir
        for i in eachindex(log_bounds)
            if x[i] < log_bounds[i][1]
                x[i] = log_bounds[i][1]
            elseif x[i] > log_bounds[i][2]
                x[i] = log_bounds[i][2]
            end
        end
    end

    if f(x) > f(x_old)
        is_descent_direction = false
        x = x_old
    end

```

```
57      return x, is_descent_direction
58  end
59
60  "Generate samples according to latin square method with same dimentions as
        bounds"
61  function latin_hypercube(n_samples, log_bounds; seed=123)
62      # fix seed
63      Random.seed!(seed)
64
65      n_vars = length(log_bounds)
66
67      # Initialize the Latin square as an n-by-n array of zeros
68      square = zeros(Int, n_samples, n_samples)
69
70      # Fill the first row with random integers between 1 and n
71      square[1, :] = randperm(n_samples)
72
73      # Fill the remaining rows with shifted copies of the first row
74      for i in 2:n_samples
75          square[i, :] = circshift(square[i-1, :], 1)
76      end
77
78      # create random values to be added to sample values
79      random_matrix = rand(n_samples, n_vars)
80
81      # create a matrix where samples will be inserted to
82      x_samples_log = zeros(n_samples, n_vars)
83
84      # generate samples with random position within varible intervals
85      for i in 1:n_samples
86          for j in 1:n_vars
87              x_samples_log[i, j] = (square[i, j] - 1) / ((n_samples - 1) * (
        n_samples / (n_samples - 1))) + random_matrix[i, j] / n_samples
88          end
89      end
90
91      # scale samples to bounds
92      for i in 1:n_samples
93          for j in 1:n_vars
94              x_samples_log[i, j] = (log_bounds[j][2] - log_bounds[j][1]) *
        x_samples_log[i, j] + log_bounds[j][1]
95          end
96      end
97
98      return x_samples_log
99  end
100
101 function sample_correction(x_samples_log, n_samples)
102     # Failed samples
103     fail_samples = []
104
105     # Successful samples
106     success_samples = []
107
108     while length(success_samples) < n_samples
109         for sample in eachrow(x_samples_log)
110             if f(sample) == Inf && length(success_samples) < n_samples
111                 push!(fail_samples, sample)
112             elseif length(success_samples) < n_samples
113                 push!(success_samples, sample)
114             end
```

```
115         end
116         if n_samples - length(success_samples) < length(log_bounds)
117             x_samples_log = latin_hypercube(12, log_bounds)
118         else
119             x_samples_log = latin_hypercube(n_samples - length(
    success_samples), log_bounds)
120         end
121     end
122 
123     x_samples_log = success_samples
124 
125     return x_samples_log
126 end
127 
128 "Remove elements in a vector equal to zeros"
129 function remove_zeros(v::AbstractVector)
130     return filter(x -> x != 0, v)
131 end
132 
133 "Check the quality of a gradient of a function f at x with ForwardDiff in
    comparison to FiniteDifferences"
134 function check_gradient(f::Function, x)
135     # gradient of first sample using ForwardDiff
136     grad_forwarddiff = ForwardDiff.gradient(f, x)
137 
138     # gradient of first sample using FiniteDifferences
139     grad_finitdiff = grad(central_fdm(10, 1), f, x)[1]
140 
141     # if gradient differs more than a tolerance the gradient is not good
    enough and the code stops
142     if any(abs.(grad_forwarddiff - grad_finitdiff) / min(abs(norm(
    grad_forwarddiff)), abs(norm(grad_finitdiff))) .> 1e-3)
143         println("Gradient too unstable")
144         return Inf
145     end
146 end
147 
148 # struct for collecting data for logging
149 struct log_results
150     sample_num_list::Vector{Int64}
151     x_current_sample_list::Vector{Union{Float64,AbstractArray}}
152     x_current_iter::Vector{Union{Float64,AbstractArray}}
153     function_values::Vector{Float64}
154     term_criteria::Vector{Union{Float64,AbstractArray,String}}
155     term_reason::Vector{Union{Float64,String}}
156     time_log::Vector{Float64}
157 end
158 
159 "Minimizes a function f a point x with a combination of steepest descent and
     newtons method"
160 function opt(f::Function, x, sample_num, iter, log_bounds; max_iter=1000)
161     # initiate lists for logging results
162     sample_num_list::Vector{Int64} = zeros(max_iter + 1)
163     x_current_sample_list::Vector{Union{Float64,AbstractArray}} = zeros(
    max_iter + 1)
164     x_current_iter::Vector{Union{Float64,AbstractArray}} = zeros(max_iter +
    1)
165     function_values::Vector{Float64} = zeros(max_iter + 1)
166     term_criteria::Vector{Union{Float64,AbstractArray,String}} = zeros(
    max_iter + 1)
167     term_reason::Vector{Union{Float64,String}} = zeros(max_iter + 1)
```

```
168     time_log::Vector{Float64} = zeros(1)
169
170     x_current_samplepoint = x
171
172     # calculate gradient for first point
173     grad = ForwardDiff.gradient(f, x)
174     func_val = f(x)
175
176     # logging for first x
177     sample_num_list[1] = sample_num
178     x_current_sample_list[1] = exp.(x_current_samplepoint)
179     x_current_iter[1] = exp.(x)
180     function_values[1] = func_val
181     term_criteria[1] = "start point, no termination criteria"
182     term_reason[1] = "start point, no reason for temination"
183
184     min_iter = 0
185
186     time = @elapsed for i in 1:max_iter
187         # increment interation number used for printing current iteration
    number
188         iter += 1
189
190         # increment iteration number for current sample, resets for each
    sample
191         min_iter += 1
192
193         # print sample number and interation number
194         println("Iteration number: ", iter, ", Sample number: ", sample_num)
195
196         # Evaluate the function and its gradient and Hessian at the current
    point
197         grad = ForwardDiff.gradient(f, x)
198
199         # To compare with the current x in termination criteria
200         x_prev = x
201
202         is_descent_direction::Bool = false
203
204         # calculate the next point via steepest descent method
205         x, is_descent_direction = steepest_descent(f, grad, x, log_bounds)
206
207         #= if a descent direction could not be found, the optmimization of
    the current sample is terminated
208         and continue with the next =#
209         if !is_descent_direction
210             println("Descent direction not found!")
211             break
212         end
213
214         # Finite termination criteria
215         eps = 1e-3
216
217         # calculate function value used in termination criteria
218         function_value = f(x)
219
220         current_term_criteria = []
221         # termination criteria 1
222         if norm(grad) <= eps * (1 + abs(function_value))
223             push!(current_term_criteria, "1")
224         end
```

```
225         # termination criteria 2
226         if f(x_prev) - function_value <= eps * (1 + abs(function_value))
227             push!(current_term_criteria, "2")
228         end
229         # termination criteria 3
230         if norm(x_prev - x) <= eps * (1 + norm(x))
231             push!(current_term_criteria, "3")
232         end
233 
234         # logging
235         sample_num_list[i+1] = sample_num
236         x_current_sample_list[i+1] = exp.(x_current_samplepoint)
237         x_current_iter[i+1] = exp.(x)
238         function_values[i+1] = f(x)
239         term_criteria[i+1] = current_term_criteria
240 
241         # Checks if two or more of the termination criteria are met
242         if length(current_term_criteria) >= 2
243             term_reason[i+1] = "Two or more termination criteria was met"
244             break
245         else
246             term_reason[i+1] = " "
247         end
248     end
249 
250     # log time for each sample
251     time_log[1] = time
252 
253     # create a log_results object for current logging data
254     res = log_results(remove_zeros(sample_num_list),
255         remove_zeros(x_current_sample_list),
256         remove_zeros(x_current_iter),
257         remove_zeros(function_values),
258         remove_zeros(term_criteria),
259         remove_zeros(term_reason),
260         time_log)
261 
262     return res, iter, min_iter, x, res.x_current_iter[end], res.
    function_values[end]
263 end
264 
265 "Runs an optimization on function f in the region of bounds with n_samples
    number of samples.
266 If running p_est through profile likelihood pl_mode should be true"
267 function p_est(f::Function, log_bounds, n_samples, pl_mode; x_samples_log=0,
     run_latin_hypercube=true)
268     if pl_mode == false
269         # create a directory for parameter estimation
270         if isdir("p_est_results") == false
271             mkdir("p_est_results")
272         end
273         # Check if the data.csv exists and truncate it if it does
274         if isfile("p_est_results/data.csv")
275             data_file = open("p_est_results/data.csv", "w")
276             truncate(data_file, 0)
277             close(data_file)
278             data = DataFrame(Iterationnumber=[],
279                 Samplepoint=[],
280                 Currentsample=[],
281                 x_current=[],
282                 Functionvalues=[],
```

```
283                 Terminationcriteria=[],
284                 Descentmethod=[],
285                 Terminationreason=[])
286             CSV.write("p_est_results/data.csv", data, header=[:Iteration, :
    Samplepoint, :Currentsample, :x_current, :Functionvalues, :
    Terminationcriteria, :Descentmethod, :Terminationreason])
287         end
288
289         if isfile("p_est_results/sample_data.csv")
290             waterfall_file = open("p_est_results/sample_data.csv", "w")
291             truncate(waterfall_file, 0)
292             close(waterfall_file)
293         end
294
295         # Check if the time_log.csv exists and truncate it if it does
296         if isfile("p_est_results/time_log.csv")
297             timelog_file = open("p_est_results/time_log.csv", "w")
298             truncate(timelog_file, 0)
299             close(timelog_file)
300         end
301
302         # previous bounds
303         if isfile("p_est_results/bounds.csv")
304             read_previous_bounds = CSV.File("p_est_results/bounds.csv") |>
    DataFrame
305             previous_bounds = [(x, y) for (x, y) in zip(read_previous_bounds
    [:, 1], read_previous_bounds[:, 2])]
306
307             if log_bounds == previous_bounds && length(readdlm("
    p_est_results/latin_hypercube.csv", Float64)[:, 1]) == n_samples
308                 x_samples_log = readdlm("p_est_results/latin_hypercube.csv",
     Float64)
309             end
310         else
311             # Generate Latin hypercube samples in the search space
312             x_samples_log = latin_hypercube(n_samples, log_bounds)
313
314             x_samples_log = sample_correction(x_samples_log, n_samples)
315
316             # save generated samples in file
317             open("p_est_results/latin_hypercube.csv", "w") do io
318                 writedlm(io, x_samples_log)
319             end
320
321             # log used bounds
322             CSV.write("p_est_results/bounds.csv", DataFrame(log_bounds))
323         end
324     end
325
326     # start values, set for first sample
327     x_min = x_samples_log[1, :]
328     f_min = f(x_min)
329
330     # initiate varable that holds the iteration that has given the lowest
    function value
331     iter_min = 1
332
333     # initiate varible sample and iteration number
334     sample_num = 0
335     iter = 0
336
```

```
337     if !pl_mode
338         iter_res::Vector{Int64} = zeros(n_samples)
339         x_sample_list::Vector{Union{Float64,AbstractArray}} = zeros(
        n_samples)
340         x_iter_min_list::Vector{Union{Float64,AbstractArray}} = zeros(
        n_samples)
341         f_min_list::Vector{Float64} = zeros(n_samples)
342     end
343 
344     # iterate over the samples, each sample is optimized
345     for x in eachrow(x_samples_log)
346         sample_num += 1
347 
348         # minimizes the cost function for the current start-guess
349         res, iter, min_iter, x, x_current_min, f_current_min = opt(f::
        Function, x, sample_num, iter, log_bounds)
350 
351         # only necessary if Profile likelihood is not currently used
352         if pl_mode == false
353             data = DataFrame(Iterationnumber=collect(1:length(res.
            sample_num_list)),
354                 Samplepoint=res.sample_num_list,
355                 Currentsample=res.x_current_sample_list,
356                 x_current=res.x_current_iter,
357                 Functionvalues=res.function_values,
358                 Terminationcriteria=res.term_criteria,
359                 Terminationreason=res.term_reason)
360 
361             # modifying the content of data.csv using write method
362             CSV.write("p_est_results/data.csv", data; append=true)
363 
364             # log time for each sample point
365             CSV.write("p_est_results/time_log.csv", DataFrame(time=res.
            time_log); append=true)
366             iter_res[sample_num] = min_iter
367             x_sample_list[sample_num] = exp.(x)
368             x_iter_min_list[sample_num] = x_current_min
369             f_min_list[sample_num] = f_current_min
370         end
371 
372         # Update the minimum point and value
373         f_val = f(x)
374         if f_val < f_min
375             x_min = exp.(x)
376             f_min = f_val
377             iter_min = iter
378         end
379     end
380 
381     if !pl_mode
382         CSV.write("p_est_results/sample_data.csv", DataFrame(sample_num=
        collect(1:length(x_sample_list)),
383             min_iter=iter_res,
384             x_sample_list=x_sample_list,
385             x_iter_min_list=x_iter_min_list,
386             f_min_list=f_min_list))
387 
388         CSV.write("p_est_results/opt_point.csv", DataFrame(x_min=[x_min],
        f_min=f_min))
389     end
390 
```

```
391     # Print the results
392     println("Minimum point: ", x_min)
393     println("Minimum value: ", f_min)
394     println("Iteration resposible for minimum: ", iter_min)
395 
396     return x_min, f_min
397 end
```

G.4 Profile likelihood

```
using LaTeXStrings
include("newton_minimize.jl")

"Determines the next point in the profiling of the cost function"
function new_point(log_param_last, param_index, log_bounds, sign, threshold; q=1e-1)
    stop_flag = false

    # initiate step size vector
    step_size = zeros(length(log_param_last))

    # initiate step size of parameter of interest
    step_size[param_index] = 1e-3 * log_param_last[param_index]

    #= since step_size[param_index] can be negative, this expression ensures that
    step_size[param_index] is always positive =#
    if step_size[param_index] < 0
        step_size[param_index] = -step_size[param_index]
    end
    cond_val = abs(f(log_param_last + sign * step_size) - f(log_param_last))

    if f(log_param_last + sign * step_size) == Inf || cond_val > q * threshold
        while f(log_param_last + sign * step_size) == Inf || cond_val > q * threshold
            step_size[param_index] /= 2
            if step_size[param_index] < 1e-6
                step_size[param_index] = 1e-6
                stop_flag = true
                break
            end
            cond_val = abs(f(log_param_last + sign * step_size) - f(log_param_last))
        end
    elseif cond_val <= q * threshold
        i = 0
        while cond_val <= q * threshold
            i += 1
            step_size[param_index] *= 2
            cond_val = abs(f(log_param_last + sign * step_size) - f(log_param_last))
            if cond_val > q * threshold
                step_size[param_index] /= 2
                break
            elseif i > 20
                break
            end
        end
    end
    new_point = log_param_last + sign * step_size

    # point can not be outside of bounds
    if sign == -1
        if new_point[param_index] < log_bounds[param_index][1]
            new_point[param_index] = log_bounds[param_index][1]
            stop_flag = true
        end
    elseif sign == 1
```

```
54          if new_point[param_index] > log_bounds[param_index][2]
55              new_point[param_index] = log_bounds[param_index][2]
56              stop_flag = true
57          end
58      end
59
60      return new_point, stop_flag
61  end
62
63  "Returns a cost function only dependent on indexes gives in index_x_small,
        other variables are held constant"
64  function intermediate_cost_function(x_small, index_x_small, x_big)
65      x_big_ = convert.(eltype(x_small), x_big)
66      x_big_[index_x_small] .= x_small
67
68      return cost_function(problem_object, x_big_, experimental_data)
69  end
70
71  # struct for logging results from profile likelihood
72  struct log_pl_results
73      fix_param_index::Vector{Int64}
74      fix_param_list::Vector{Float64}
75      x_list::Vector{Union{Float64,AbstractArray}}
76      costfunc_value_list::Vector{Float64}
77  end
78
79  "Perform profile likelihood analysis for one parameter"
80  function profile_likelihood(params, param_index, log_bounds, num_points,
        threshold)
81      # list of indexes to be optimized
82      index_list = [i for i in 1:length(log_bounds) if i != param_index]
83
84      # log-scale parameters
85      log_params = log.(params)
86
87      # new bounds
88      bounds_ = copy(log_bounds)
89      current_bounds = deleteat!(bounds_, param_index)
90
91      # new start values
92      log_x_samples = readdlm("profilelikelihood_results/pl_latin_hypercube",
        Float64)
93
94      # remove the bound at the index of the parameter that is profiled
95      new_log_x_samples = hcat(log_x_samples[:, 1:param_index-1],
        log_x_samples[:, param_index+1:end])
96
97      stop_flag = false
98
99      # begin profiling in negative direction
100     sign = -1
101
102     # initiate lists for logging
103     fix_param_index::Vector{Int64} = zeros(2 * num_points + 1)
104     fix_param_list::Vector{Union{Float64,AbstractArray}} = zeros(2 *
        num_points + 1)
105     x_list::Vector{Union{Float64,AbstractArray}} = zeros(2 * num_points + 1)
106     costfunc_value_list::Vector{Float64} = zeros(2 * num_points + 1)
107
108     # log optimized parameters (start values)
109     fix_param_index[Int(num_points)+1] = param_index
```

```
110     fix_param_list[Int(num_points)+1] = x_min[param_index]
111     x_list[Int(num_points)+1] = [x_min[i] for i in eachindex(x_min) if i !=
        param_index]
112     costfunc_value_list[Int(num_points)+1] = f_min
113
114     i = 0
115     log_params_current = log_params
116
117     while i < num_points
118         i += 1
119
120         #= continue to profile in the current direction if stop_flag is not
        true and maximum steps (num_points)
121         in that direction is not taken, else change direction. If profiling
        has been done in both directions
122         break the code =#
123         if stop_flag == false && i != num_points
124             # calculate next point
125             log_params_current, stop_flag = new_point(log_params_current,
        param_index, log_bounds, sign, threshold)
126
127         elseif (i == num_points && sign == -1 && !(stop_flag == true)) ||
        (!(i == num_points) && sign == -1 && stop_flag == true)
128             sign = 1
129             i = 1
130             log_params_current = log_params
131             log_params_current, stop_flag = new_point(log_params_current,
        param_index, log_bounds, sign, threshold)
132         else
133             break
134         end
135
136         # Redefine the cost function
137         cost_function_profilelikelihood = (x) -> intermediate_cost_function(
        x, index_list, log_params_current)
138
139         # Find the maximum likelihood estimate for the parameter of interest
140         x_min, f_min = p_est(cost_function_profilelikelihood, current_bounds
        , 1000, true; x_samples_log=new_log_x_samples)
141
142         # update logging lists with results
143         if sign == -1
144             fix_param_index[Int(num_points)+1-i] = param_index
145             fix_param_list[Int(num_points)+1-i] = exp.(log_params_current[
        param_index])
146             x_list[Int(num_points)+1-i] = x_min
147             costfunc_value_list[Int(num_points)+1-i] = f_min
148         else
149             fix_param_index[Int(num_points)+1+i] = param_index
150             fix_param_list[Int(num_points)+1+i] = exp.(log_params_current[
        param_index])
151             x_list[Int(num_points)+1+i] = x_min
152             costfunc_value_list[Int(num_points)+1+i] = f_min
153         end
154
155         # function value should not exceed a specific percentage of the
        starting point of profile profile_likelihood
156         if f_min - f(log_params) > 1.2 * threshold
157             stop_flag = true
158         end
159     end
```

```
160
161     # puts data in struct and removes zeros
162     pl_res = log_pl_results(remove_zeros(fix_param_index),
163         remove_zeros(fix_param_list),
164         remove_zeros(x_list),
165         remove_zeros(costfunc_value_list))
166
167     return pl_res
168 end
169
170 "Run profile likelihood with the optimized parameters params, specifying how
        many steps can
171 maximally be made in each direction"
172 function run_profile_likelihood(params, log_bounds, n_samples_pl, num_points
    , threshold)
173     # create a directory for profile likelihood
174     if isdir("profilelikelihood_results") == false
175         mkdir("profilelikelihood_results")
176     end
177
178     # save threshold
179     CSV.write("profilelikelihood_results/threshold.csv", DataFrame(threshold
    =threshold))
180
181     # Check if the profile_likelihood.csv exists and truncate it if it does
182     if isfile("profilelikelihood_results/profile_likelihood.csv")
183         pl_file = open("profilelikelihood_results/profile_likelihood.csv", "
    w")
184         truncate(pl_file, 0)
185         close(pl_file)
186     end
187
188     x_samples_log = latin_hypercube(n_samples_pl, log_bounds)
189     x_samples_log = sample_correction(x_samples_log, n_samples_pl)
190
191     # save generated samples in file
192     open("profilelikelihood_results/pl_latin_hypercube", "w") do io
193         writedlm(io, x_samples_log)
194     end
195
196     # iterate over all parameters
197     for i in 1:length(log_bounds)
198         # run profile likelihood for the current parameter
199         pl_res = profile_likelihood(params, i, log_bounds, num_points,
    threshold)
200
201         # create a DataFrame for the data to be logged
202         data = DataFrame(Fixed_parameter_index=pl_res.fix_param_index,
203             Fixed_parameter=pl_res.fix_param_list,
204             Parameters=pl_res.x_list,
205             CostfunctionValues=pl_res.costfunc_value_list)
206
207         # modifying the content of profile_likelihood.csv using write method
208         CSV.write("profilelikelihood_results/profile_likelihood.csv", data;
    append=true)
209     end
210 end
211
212 "Make contour plot for 2 variables, only possible for a model with two
    unknown paramters"
213 function contourplot_2p()
```

```
    x = collect(range(0.1, 6, length=100))
    y = collect(range(0.1, 6, length=100))
    points = Vector{Vector{Float64}}(undef, length(x) * length(y))

    k = 1
    for i in x
        for j in y
            points[k] = [i, j]
            k += 1
        end
    end

    A = zeros(100, 100)
    index = 1 # initialize index for vector
    for i in 1:100
        for j in 1:100
            # Generate the element you want to insert
            element = f(log.(points[index]))

            # Insert the element into the matrix at the current index
            A[i, j] = element

            index += 1 # increment index for next element in vector
        end
    end
    contour(x, y, A, xaxis=L"\theta_1", yaxis=L"\theta_2", colorbar_title="
    Kostnadsfunktion", labelfontsize=15, colorbar_titlefont=font(15))
    savefig("plot_2d.png")
end
```

G.5 Plottning av resultat från parameterestimation

```
using Plots
using CSV
using DataFrames
using LaTeXStrings

function plot_pl()
    data = CSV.read("profilelikelihood_results/profile_likelihood.csv", DataFrame;
        header=[:fixed_parameter_index, :fixed_parameter, :parameters, :cost_function_values])

    grouped = groupby(data, :fixed_parameter_index)
    # Iterate over groups and create a plot for each one
    for (sample_number, group) in pairs(grouped)
        sample_number = sample_number.fixed_parameter_index

        # plot profile likelihood
        plot(group[!, :fixed_parameter], group[!, :cost_function_values], xaxis=L"$\theta_%$sample_number$", yaxis="Kostnadsfunktion", legend=false, lc=:black, lw=2, labelfontsize=15)

        # plot threshold
        x = collect(Float64, range(group[!, :fixed_parameter][1], group[!, :fixed_parameter][end], length=2))
        y = (CSV.read("p_est_results/opt_point.csv", DataFrame)[1, 2] + CSV.read("profilelikelihood_results/threshold.csv", DataFrame)[1, 1]) * ones(length(x))
        plot!(x, y, lc=:black, linestyle=:dash, lw=2)

        #save plot
        savefig("profilelikelihood_results/parameter$sample_number.png")
    end
end

function plot_waterfall()
    data = CSV.read("p_est_results/sample_data.csv", DataFrame)
    y = data[:, 5]
    sort!(y)
    x = collect(1:length(y))

    plot(x, y, xaxis="Startvärden", yaxis="Kostnadsfunktion", lw=2, labelfontsize=15, legend=false)
    savefig("p_est_results/waterfall_plot")

    n_convergent_samples = 0
    for i in 1:length(x)
        if abs(y[i] - y[1]) < 0.1
            n_convergent_samples += 1
        else
            break
        end
    end
    convergence_ratio = 100 * (n_convergent_samples / length(x))
    result = "$convergence_ratio% of the start points converged to the same minimum point,
with an accepted difference of 0.1 in function value from the smallest function value"
    CSV.write("p_est_results/waterfall_results.csv", DataFrame(waterfall_result=result))
```

```
49 end
50
51 # plot profile likelihood
52 plot_pl()
53
54 # plot waterfall plot
55 plot_waterfall()
```