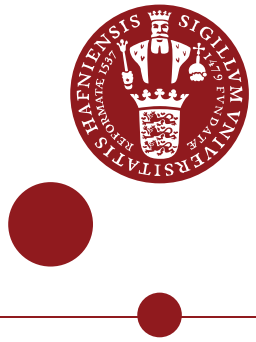
KØBENHAVNS UNIVERSITET
DET SAMFUNDSVIDENSKABELIGE FAKULTET
ØKONOMISK INSTITUT



# Kandidatspeciale

Andreas Jakob Vodstrup & Mads Chrøis

# Kognitive hierarkimodeller i avancerede spil

## En empirisk analyse ved brug af data fra online gaming

Vejleder: Anders Munk-Nielsen

Antal ECTS: 2 x 30

Afleveret den: 19. december 2020

Antal typeenheder: 169.377

Økonomisk Institut, Københavns Universitet.

Kandidatspecialet er skrevet i 12 pt. med halvanden linjeafstand og 2,5 cm margener.
Opsætning, typografi, tabeller og figurlayout er lavet i Overleaf.
Databearbejdning og statistisk analyse er foretaget i Python 3.7.1, Microsoft Excel (Office 365) og Gambit.

# Forord

Inden vi påbegynder kandidatspecialet, vil vi gerne takke personer, institutter og virksomheder, der har hjulpet til udarbejdelsen af afhandlingen.

Først og fremmest et kæmpe tak til Anders Munk-Nielsen for at være en yderst kompetent vejleder igennem hele processen, der har været turbulent grundet den globale Covid-19-pandemi. Dernæst skal virksomheden Vicious Syndicate have et stort tak for brug af deres data. Slutteligt skal Økonomisk Institut på Københavns Universitet takkes for de mange fremragende studieår, såvel som at stille specialepladser til rådighed under skriveprocessen.

Afsnit 1, 9.4 og 11 er skrevet af både Andreas Jakob Vodstrup og Mads Chrøis, og vi er begge ansvarlige for disse dele.
Andreas Jakob Vodstrup er primærforfatter og hovedansvarlig for afsnit 3.1, 3.3, 5, 6.2, 7.2, 9.1, 9.2, 9.3, 9.5.1 og 10 side 77-79.
Mads Chrøis er primærforfatter og hovedansvarlig for afsnit 2, 3.2, 4, 6.1, 7.1, 7.3, 8, 9.5.2 og 10 side 74-77.
Det skal dog understreges, at specialet er udarbejdet i fællesskab.

A. Jakob Vodstrup Mads Chrøis

# Abstract

Behavioral game theory has in the past decade received an increased amount of attention in economic literature. At this point in time John Nash's theorem, and the equilibrium strategy that follows, is disputed whether it properly models agents decision-making in real settings. This causes a disparity in economic modelling - if agents are not rational, then how do we model their behavior?

Camerer et al. [2004] suggests that a Poisson Cognitive Hierarchy Model (PCHM) fits empirical data better than the Nash equilibrium. PCHM describes a framework where each player assumes that her strategy is the most sophisticated, and the distribution of opponents of lower levels follows a Poisson distribution. However, PCHM has shortcomings. The Poisson distribution is limited by a single parameter, $\tau$, which limits its ability to model cases where there is a surplus of strong strategic thinkers as well as agents who act at random.

In this paper, we provide the theory behind the Nash equilibrium, the cognitive hierarchy model and their limitations. Using data from the online card game "Hearthstone" we model the decision-making process of individuals in an attempt to explain human behaviour in advanced games. Building on the basis of the PCHM by Camerer et al. [2004], we present a new cognitive hierarchy model founded on the assumption, that the expected distribution of individuals follows a beta distribution. We find that this model explains behaviour marginally better than the PCHM, although it still struggles to explain large parts of the strategic behaviour observed in data. Specifically, we find that approximately 86 pct. of players act random in their strategic assessments, whilst between 1.8 and 5.9 pct. of agents perform more than *one* step of iterated strategic thinking. However, we find that players who play the game in a more competitive setting displays a significantly increased number of strategic iterations. The results are consistent over time and highlight a common problem faced in behavioral game theory: Decisions are usually *not* simple and the current selection of models lack the tools to properly analyse complicated strategic considerations. We thus encourage others to further investigate the potential of cognitive hierarchy models to model human behavior.

# Indholdsfortegnelse

# Figurliste



# Tabelliste

# 1 Indledning

Adfærdsøkonomi er de senere år steget i fokus inden for økonomisk forskning, hvor Colin F. Camerer er en af de førende forskere. I samarbejde med sine kollegaer prøver de at forklare menneskelig adfærd i simple spil. Til dette har de opfundet en såkaldt kognitiv hierarkimodel. Modellen tester i laboratorieforsøg, hvor stor en andel af agenter der agerer mere eller mindre rationelt i simple 3x3 spil. Dette kandidatspeciale tager udgangspunkt i de samme modeller, men har til formål at teste dem i avancerede spil ved brug af data fra onlinespillet "Hearthstone". Ved numerisk optimering løser vi mixed Nash-ligevægten og finder, at denne forklarer den observerede adfærd i data dårligt. De kognitive hierarkimodeller forklarer adfærden markant bedre, men vi finder, at de fortsat har svært ved at forklare strategivalgene i den avancerede setting, og en mindre andel af populationen klassificeres som kognitivt tænkende sammenlignet med Camerer et al. [2004]. Imidlertid analyserer vi delmængder af data fra hhv. højt og lavt kompetetivt niveau og finder, at high ability agenter udviser større omtanke i deres strategivalg end low ability. Dette indikerer, at der foretages højere grader af strategisk tænkning i højt kompetetive miljøer. Med disse erfaringer opstiller vi mulige udvidelser til modellerne, således at de potentielt kunne modellere agenters adfærd i avancerede spil bedre.

---

Siden Adam Smith fremlagde sin teori om den usynlige hånd i bogen "The Theory of Moral Sentiments" (Smith [1759]), har økonomi som videnskab prøvet at forklare både mikro- og makroøkonomiske tendenser og opstillet teorier, der skulle håndtere både individuel ageren såvel som aggregeret. Økonomi har historisk set antaget fuldkommen rationalitet, men har imidlertid afgrenet sig til forskellige felter, herunder adfærdsøkonomi. Adfærdsøkonomien inddrager andre videnskaber såsom psykologi for på bedst mulig vis at kunne forklare menneskelig adfærd. Ved at fremføre en litteraturoversigt gennemgår afsnit 2 motivationen såvel som byggestenene, der ligger til grund for afhandlingen. Dizikes [2020] viser, at artikler inden for psykologi i nyere tid citerer økonomiske artikler lige så meget som sociologiske - en tendens der ikke var gældende for blot 10 år siden. Det samme sker inden for andre felter, fordi flere artikler inden for økonomi i højere grad divergerer fra rationalitetsbetingede modeller til at fokusere på forklaring af faktisk menneskelig adfærd.

Allerede i 1936 udtaler Keynes [1936]: *"It is not the case of choosing those which, to*

*the best of one's judgment, are really the prettiest, nor even those which average opinion genuinely thinks the prettiest. We have reached the third degree, where we devote our intelligences to anticipating what average opinion expects the average opinion to be. And there are some, I believe, who practice the fourth, fifth, and higher degrees"* og implicerer da, at ikke alle individer er lige rationelle. Citatet refererer til en avisartikel, hvor læserne skulle gætte det ansigt, som de mente, at andre individer fandt smukkest. Idéen, der lå til grund for konkurrencen i denne avisartikel, er senere blevet døbt *Keynesian Beauty Contest*-spil, og den findes stadig jævnligt i avisartikler i dag, f.eks i Politiken (Schou [2005]) eller The New York Times (Leonhardt og Quealy [2015]). Her skal spillerne vælge et nummer mellem 0 og 100, og den spiller der vælger nummeret tættest på en vilkårlig andel (eksempelvis $\frac{2}{3}$) af gennemsnittet vinder konkurrencen. Fuldkomment rationelle spillere og antagelse af samme hos modspillere vil resultere i Nash-ligevægten, hvor alle gætter på 0. I Beauty Contest spillet, hvor spillerne skal gætte $\frac{2}{3}$ af gennemsnittet, vil ingen strategiske spillere gætte over 67,[1] og en sådanne spiller er, hvad Keynes beskriver som 1. grads rationel. En 2. grads rationel vil tage skridtet længere og antage, at ingen vil spille over 45,[2] 3. grad på 30 osv. Eksperimenter med dette setup giver ofte et gennemsnitligt gæt på mellem 20 og 35 (Crawford et al. [2013]), hvorfor tankeprocessen bag individernes beslutninger er interessant, da dette resultat indikerer, at individer ikke spiller Nash-ligevægten. Spillet lagde grundlag for *k-rationalitetsmodeller*, der forklarer den kognitive fordeling af populationen på baggrund af det antal strategiske iterationer, de udfører. Modellerne forklarer ofte data bedre end Nash-ligevægten (Fudenberg og Liang [2019]).

Camerer et al. [2004] udvider k-rationalitetsmodellerne med en såkaldt *kognitiv hierarkimodel*, hvor agenterne best-responder til en fleksibel fordeling af modstandere. De gør dette ved laboratorieforsøg, som Cassar og Friedman [2004] argumenterer for, ofte giver bedre indsigt i menneskelig adfærd, end feltstudier gør. Forsøgstypen er dog under kritik for ikke altid at være retvisende for virkeligheden.[3] Feltstudier giver dog et indblik i, hvordan tusinder af agenter rent faktisk agerer, hvorfor der er behov for, at begge forsøgstyper

[1] 2/3 af 100.
[2] 2/3 af 67.
[3] Se f.eks. Levitt og List [2007]: "What Do Laboratory Experiments Measuring Social Preferences Reveal About the Real World?" og Lane [2016]: "Discrimination in the laboratory: A meta-analysis of economics experiments".

bliver anerkendt. Denne afhandling benytter data fra onlinespillet "Hearthstone", der kan opstilles som et statisk to-personers spil, og det vil derfor kategoriseres som et feltstudie. Datasættet inkluderer en stor mængde data fra spilindustrien, der er let tilgængeligt samt billigt at anvende til spilteoretiske modeller, hvorfor dette data er oplagt at bruge til analyse af menneskelig adfærd.

Dette tillader os at fremføre forskningspørgsmålet, som denne afhandling vil forsøge at besvare:

*Hvordan agerer individer i avancerede spil, og kan kognitive hierarkimodeller anvendes til at forklare denne adfærd?*

Hvor Camerer et al. [2004] finder et relativt konsistent resultat for, at under halvdelen af populationen i laboratorieforsøg er ustrategiske spillere, finder vi, at samme model prædikerer en andel på ca. 86 pct. af ustrategiske spillere i det avancerede setup. Vi undersøger, hvorvidt modellen er begrænset af dens parametriske fordelingsantagelse, og vi anvender alternative modeller, der tager udgangspunkt i andre fordelinger, end den Camerer bruger. Vi finder, at disse nye modeller forklarer data marginalt bedre, og vi udforsker hernæst, om der er forskel på strategivalget mellem high- og low ability spillere. Vi finder i afsnit 9 at andelen af ustrategiske spillere falder når vi betragter high ability spillere. Der tegner sig da et billede af, at individer agerer efter et kognitivt hierarki, hvor high ability spillere tænker markant mere strategisk end low ability.

På trods af at resultaterne afviger markant fra Camerer et al. [2004], finder vi belæg for, at kognitive hierarkimodeller kan give indblik i menneskelig adfærd selv for avancerede spil, og de kan med de rigtige udvidelser have potentialet til at forklare den endnu bedre.

Som nævnt vil der i afsnit 2 være en kortfattet litteraturgennemgang af udviklingen i spilteori indtil nu med fokus på den gren af adfærdsøkonomi, som er relevant for vores emne. Afsnit 3 gennemgår nogle grundbegreber inden for spilteori, mens afsnit 4 beskriver, hvorfor vi kan opstille de strategiske overvejelser i Hearthstone som et spilteoretisk problem. Herefter klargører afsnit 5 dataindsamlingsprocessen og potentielle fejlkilder for-

bundet ved denne. Afsnit 6 går mere teoretisk i dybden med de specifikke modeller, vi tester i afhandlingen, og afsnit 7 beskriver de statistiske værktøjer, der anvendes i analysen. Afsnit 8 gennemgår specifikationen og løsningmetoden for vores modeller, hvor afsnit 9 beskriver og sammenligner resultaterne fra de forskellige modeller. Afsnit 10 diskuterer, hvordan nogle af de antagelser, vi tager i løbet af afhandlingen, potentielt kan påvirke resultaterne, samt giver et indblik i, hvilke udvidelser der ville være interessante i fremtidig forskning. Slutteligt afrundes der i afsnit 11.

# 2 Litteraturreview

I dette afsnit vil vi gennemgå relevant litteratur, således at der gives et kronologisk indblik i den generelle økonomiske historie frem til Nash-ligevægten, hvorefter fokus ligges på adfærdsmæssig spilteori. Vi runder af med den nuværende situation i adfærdsmæssig spilteori, som kognitive hierarkimodeller stammer fra. Dette for at give indblik i fordele og begrænsninger ved modellerne der anvendes inden for dette felt.

Selvom breve mellem matematikere viser, at samtaler omhandlende løsningen af to-personers spil har været diskuteret siden start 1700-tallet, som f.eks. Waldegrave Problemet,[4] blev spilteori først anerkendt som et unikt felt indenfor matematik i 1928 da John von Neumann udgav bogen "On the Theory of Games of Strategy"(von Neumann [1928])[5]. Efterfølgende udgav han i 1944, i samarbejde med Oskar Morgenstern, bogen "Theory of Games and Economic Behavior"(von Neumann og Morgenstern [1944]). Her opstilles en teori, hvori agenterne mikser mellem strategier i stedet for at spille én strategi med sikkerhed. Teorien blev udvidet i 1951 af John Nash, som beviste, at der for alle to-personers spil med et begrænset antal strategier vil eksistere mindst én Nash-ligevægt - et udtryk netop opkaldt efter John Nash. Nash's teorem lagde fundamentet for den moderne spilteori, og sidenhen er spilteori blevet benyttet inden for mange videnskabelige felter, herunder beskrivelsen af menneskelig adfærd, såvel som grundlag for politiske handlinger. I de senere år har man dog fundet, at Nash-ligevægtens prædiktioner kan være misledende, og teorien derfor nogle gange ikke formår at prædikere den menneskelige adfærd korrekt. I afsnit

[4]Fra bogen "Essay d'analyse sur les jeux de hazard"af pierre Rémond de Montmort og brevkorrespondancer hertil.

[5]Selvom forskere som Antoine-Augustin Cournot og Francis Edgeworth begge havde nogle af de samme idéer.

3 vil en mere grundlæggende teoretisk gennemgang af den klassiske spilteori blive fremvist.

Som nævnt i afsnit 1 er andelen af citerede økonomiske artikler steget markant i fag som sociologi og psykologi, som dels skyldes fremgangen i adfærdsøkonomi, der fortsat tilegner sig momentum. I Allais-paradokset (Allais [1953]) fremgår det, at der er en inkonsistens mellem agenters adfærd og den adfærd, som Expected Utility Theory (EUT) forudsiger. Agenter handler altså *ikke* altid rationelt og i overensstemmelse med traditionel teori og *Homo økonomikus*. Kahneman og Tversky [1979] fremlægger deres "Prospect Theory", der opstiller en model som alternativ til EUT. Modellen forklarer, at agenter handler forskelligt, når de står overfor valg, der involverer risiko, da den observerede adfærd er inkonsistent, når udfaldet gavner dem,[6] og når udfaldet skader dem.[7] I 1992 reviderer de to forfattere teorien, mens koncepter som overconfidence og illusion of control bliver en del af den samlede teori - koncepter der prøver at forklare agenternes adfærdsmæssige afvigelser fra antagelsen om fuldkommen rationalitet. F.eks. beskriver overconfidence, hvordan agenter har for høje forventninger til deres egne evner og konsekvent overvurderer, hvor gode deres egne beslutninger er, relativt til andres. Glaser og Weber [2007] viser i et forsøg med 3.000 investorer, at et overtal af individer overvurderer deres egne evner, hvilket korroborerer teorien om overconfidence.

Camerer et al. [2003] finder, at Nash-ligevægten er mere relevant, hvis agenterne har mulighed for at opnå erfaring med det specifikke spil. Dette kan enten opnås ved gentagne runder i et eksperiment eller gentagne gange at deltage i samme eksperiment. I disse spil, hvor det er muligt at tilegne sig erfaring, har spillernes strategier en tendens til at konvergere mod Nash-ligevægten. Der findes dog spilformater, hvor spillerne ikke har mulighed for at tilegne sig erfaring. Hvis spillernes strategier i disse spilformater konvergerer mod en ligevægt, kræver det et vist niveau af strategisk tænkning (Crawford et al. [2013]). Nagel [1995], Ho et al. [1998] og Bosch-Domenech et al. [2002] viser, at spillerens initiale strategi kan afvige systematisk fra ligevægten, selv i tilfælde hvor ligevægten kan opnås ved elimination af dominerede strategier. Stahl og Wilson [1995], Costa-Gomes et al. [2001] og Costa-Gomes og Crawford [2006] korroborerer resultaterne. De finder, at over halvdelen af deltagernes beslutninger i symmetriske to-personers matrixspil forklares

[6] Gain Domain.
[7] Loss domain.

bedre af en teori om kognitive hierarkier end ligevægtsteori.

Disse kognitive hierarkier bygger på, at agenten, der skal træffe en strategisk beslutning, tilskriver kognitive niveauer til andre spillere, som agenten optimerer ud fra. Teorien om kognitive hierarkier ses i litteraturen som en vigtig del af modeller, der beskriver menneskelig adfærd. Woodruff og Premack [1979] demonstrerede ved eksperimenter med chimpanser, at det også for andre primater end mennesket var muligt i en vis grad at lave flere kognitive iterationer. Teorien har endvidere vist sig særlig relevant til strategiske sociale anretninger, såsom spil. Shultz og Cloghesy [1981] undersøgte i et eksperiment blandt kortspillere, hvorvidt spillerne kunne foretage kognitive beregninger på baggrund af spillere af lavere niveau.[8] De konkluderer, at denne iterative kognitive proces kun finder sted hos relativt udviklede individer. Barenboim [1978] finder evidens for iterativ strategisk tænkning blandt unge individer. På baggrund af disse resultater er det usikkert at konkludere, hvor udbredt iterativ strategisk tænkning er på tværs af en hel befolkning, men fundene påpeger, at individer besidder strategiske kompetencer af forskellige niveauer, hvilket stemmer overens med teorien om kognitive hierarkier.

Selten [1998] forklarer, at det grundlæggende element i ligevægtsteori i spil ofte bygges på cirkulære antagelser, men at trinvis rationel strategisk tænkning[9] undgår disse antagelser. Rationel strategisk tænkning tillader dermed at opstille et problem med en reel løsning. Adfærdsmæssig spilteori fokuserer primært på sådanne modeller, hvor spilleren antager en vis form for rationalitet hos sine modspillere og best-responder til deres strategivalg. En type af disse modeller er, som nævnt i afsnit 1, level-k og kognitive hierarkimodeller. Nagel [1995] og Stahl og Wilson [1995] var nogle af forløberne, der anvendte en simpel model til at forklare valg af tal i deres $p$-beauty contest-spil.[10] De opdelte populationen, der deltog i eksperimentet, i tre forskellige kognitive iterationer; level-*0*, level-*1* og level-*2*, hvor Stahl og Wilson [1995] inkluderer en 'worldly' type.[11]

Den første reelle modellering af den level-$k$ model, der anvendes i denne analyse, blev

[8] I setuppet fra Shultz og Cloghesy [1981] spiller voksne mod børn, hvor børnene påtager sig rollen som spillere af lavere niveauer.
[9] Hvor agenten er rationel, men ikke nødvendigvis antager, at andre er rationelle af samme grad.
[10] En udvidelse til det eksperiment Keynes opstillede i 1936.
[11] En wordly type antager, at fordelingen af individer, som hun spiller imod, består af disse tre kognitive niveauer og individer, der spiller Nash-ligevægten.

introduceret af Costa-Gomes et al. [2001]. Modellen inddeler individer i forskellige typer, hvor den enkelte type navngives level-$k$, som refererer til spillerens kognitive niveau. Konceptet bag level-k er, at et individ af $k$. niveau best-responder til spillere af level-$(k-1)$'s handlinger. Denne type modeller er senere blevet anvendt til eksperimenter med simple spil af Costa-Gomes og Crawford [2006], Crawford og Iriberri [2007] og Costa-Gomes et al. [2009]. Endvidere introducerer Crawford [2003] en level-$k$ model af ensidet førspilskommunikation i et to-persons spil. Generelt finder de, at størstedelen af populationen består af spillere, der angives som level 1 eller level 2.

Camerer et al. [2004] opstiller en *poisson cognitive hierarchy model* (PCHM), der bygger videre på level-k konceptet. PCHM antager, at spillere af niveau $k$ spiller mod en kombination af strategiske individer $< k$. De modellerer spillernes forventninger til fordelingen af modstanderne med en poissonfordeling, der afhænger af parameteren $\tau$. Det er altså muligt for spillerne i denne model at antage, at de spiller mod et miks af levels under dem selv og best-responder dermed til en fordeling af disse individer. På trods af at PCHM er en konceptuel model, har den vist sig at være særdeles velforklarende i en bred vifte af spil, herunder p-beauty contest games, 3×3 matrixspil og market entry-spil (Camerer et al. [2004]) samt i commitment og coordination games (hhv. Carvalho et al. [2014] og Costa-Gomes et al. [2009]). Dog fejler modellen i spil som Prisoner's dilemma, da alle spillere over level 0 spiller optimalt og aldrig spiller dominerede strategier (Crawford et al. [2013]). Chong et al. [2016] opstiller en alternativ model kaldet *Generalized Cognitive Hierarchy* (GCH), som laver en nested variant af den tidligere CH-model, som integrerer stereotypiske biases. De finder, at modellen forklarer adfærd marginalt bedre end PCHM.

Denne analyse tager udgangspunkt i CH-modellen fra Camerer et al. [2004], hvor vi udvider med en mere fleksibel fordeling, navnlig betafordelingen. I næste afsnit beskrives grundlæggende spilteori, som danner fundamentet for level-k og CH-modellerne beskrevet i afsnit 6.

# 3 Introduktion til spilteori

Inden for spilteori findes forskellige spiltyper, og løsningerne hertil varierer afhængigt af, hvilken type vi betragter. Dette afsnit giver et overblik over nogle af de mest normale spiltyper samt et indblik i forskellige løsningsmetoder.

Når vi taler om spil, refererer vi oftest til situationer, hvor en eller flere agenter står over for en række beslutninger, der har en effekt på andre spillere. Antallet af spillere kan principielt variere fra én person til uendeligt mange. I denne afhandling fokuserer vi udelukkende på spil med to agenter. De forskellige typer af spil nedenfor antager derfor, at vi taler om to-personers spil.

## 3.1 Statisk eller dynamisk

Når agenter vælger deres optimale strategi, afhænger dette valg af, om modstanderen skal foretage sit strategivalg samtidigt, eller om valgene foretages forskudt. Et spil, hvor begge parter vælger samtidigt, altså uden at vide hvad den anden agent vælger, kaldes et statisk spil. Omvendt kalder vi det et dynamisk spil, hvis de skiftes til at vælge en strategi.

For statiske spil vil begge spillere stå over for et antal af mulige træk. Dette antal kan være mellem 0 og $k < \infty$, men hvis det for en af spillerne er 0, har denne spiller ingen effekt på spillets udfald. Vi observerer oftest kvadratiske spil, hvori begge spillere har det samme antal strategier, men det er ikke en nødvendighed. I disse spil vil enhver unik sammensætning af strategier resultere i et såkaldt payoff til begge parter. I generelle eksempler vil vi illustrere spil i form af en m $\times$ n matrix, hvor m er antallet af mulige træk for spiller R (række), og n er antallet af mulige træk for spiller K (kolonne). Payoff for hvert af disse træk afbildes i matricen.

Eksempelvis ville et 2x2 spil mellem spiller R og K, hvor spiller R har de to mulige træk "T" og "B", imens spiller K har "V" og "H", udtrykkes ved:

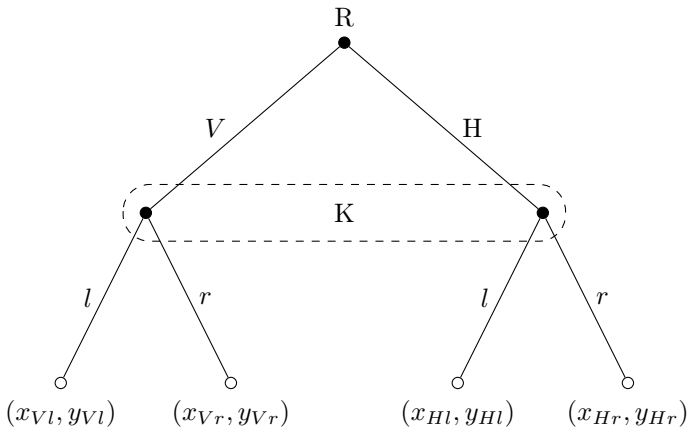
**Tabel 3.1:** Eksempel på 2x2 payoff-matrix

| | | Spiller $K$ | |
|---|---|---|---|
| | | $V$ | $H$ |
| Spiller $R$ | $T$ | $x_{TV}, y_{TV}$ | $x_{TH}, y_{TH}$ |
| | $B$ | $x_{BV}, y_{BV}$ | $x_{BH}, y_{BH}$ |

Hvor spillernes payoffs er hhv. $x$ for R og $y$ for K.

Hvis vi i stedet betragter et dynamisk spil med fuld information, skiftes spillerne til at tage et træk. F.eks. starter spiller R med at skulle vælge mellem "V" eller "H", hvorefter spiller K vælger mellem "l" og "r". Spillet er altså meget lig det ovenfor, dog hvor spiller K kan observere, hvilket træk spiller R laver, og agere derefter. Et sådant spil opstiller vi med et "spiltræ", der visualiserer hver spillers mulige træk i nogle grene. Eksemplet beskrevet kan ses i figur 3.1 nedenfor.

**Figur 3.1:** Eksempel på spiltræ




I både statiske og dynamiske spil kan man have at gøre med komplet/ikke-komplet information. De ovenfor beskrevne eksempler er spil, hvor begge spillere kender til alle aspekter af spillet, og det er derfor et spil med komplet information. Hvis én eller flere af spillerne derimod ikke kender alle aspekter af spillet, taler vi om et spil med ukomplet information. Et eksempel på dette kunne være, at spiller K i ovenstående eksempel ikke ser spiller R's træk og derved ikke ved hvilken side hun er i. I afsnit 3.2 vil der blive gennemgået metoder til, hvordan spilteori giver løsningsforslag til spil lig dem beskrevet ovenfor. Da

afhandlingen omhandler statiske spil,[12] vil der kun blive gennemgået løsninger til disse.[13]

## 3.2 Løsning af statiske spil

### Iterated Elimination of Strictly Dominated Strategies

Før vi gennemgår en generel metode til at finde løsninger i statiske spil, er det værd at nævne Iterated Elimination of Strictly Dominated Strategies (IESDS). IESDS kan benyttes i spil, hvori én eller flere af strategierne er stærkt domineret af en anden strategi. Vi lader $S_i$ være et sæt af strategier for spiller $i$, og hendes nyttefunktion er hertil $u_i(s_1, s_2, ..., s_n)$, hvor $s_1$ angiver strategien for spiller R. Hendes nytte afhænger således også af strategivalget fra modspillerne. Hvis $s_1^*$ og $s_1^{**}$ er strategier for spiller 1, vil $s_1^*$ dominere $s_1^{**}$, hvis $u_1(s_1^*, s_2, s_3, ..., s_n) > u_1(s_1^{**}, s_2, s_3, ..., s_n) \ \forall \ s_2, s_3, ..., s_n$.

Eksempelvis hvis vi betragter 2x3 payoff-matricen nedenfor, vil strategi "M" være en domineret strategi for spiller K, da det både for "T" og "B" bedre vil kunne betale sig at spille "V".

| | | Spiller $K$ | | |
|---|---|---|---|---|
| | | $V$ | $M$ | $H$ |
| Spiller $R$ | $T$ | $3,2$ | $8,1$ | $3,5$ |
| | $B$ | $4,4$ | $9,1$ | $4,0$ |

Spiller R vil da antage, at "M" ikke bliver spillet, og vi kan reducere spillet til en 2x2 matrix. Spiller K kan bruge samme metode til at eliminere spiller R's "T", da "B" er strengt bedre. Dette betyder, at spiller K altså vælger mellem at spille "V" og få 4 eller "H" og få 0, hvorfor han vælger "V", og spiller R vælger "B".

| | | Spiller $K$ | |
|---|---|---|---|
| | | $V$ | $H$ |
| Spiller $R$ | $T$ | $3,2$ | $3,5$ |
| | $B$ | $4,4$ | $4,0$ |

Ikke alle spil kan løses på denne måde, hvorfor vi i næste delafsnit vil gennemgå to mulige løsningsmetoder til statiske spil, nemlig ved at finde rene- og mixed Nash-ligevægte. Vi

[12] Selvom k-rationalitets og CH-modeller prøver at omdanne tankestrømmen bag løsningen af statiske spil til det lignende dynamiske.
[13] Nogle af metoderne gælder dog for begge, f.eks. IESDS.

definerer Nash-ligevægten for et spil med n strategier, hvor $(s_1^*, s_2^*, s_3^*, ...s_n^*)$ er en Nash-ligevægt, hvis:

$$u_i(s_1^*, s_2^*, ..., s_i^*, ..., s_n^*) \geq u_i(s_1^*, s_2^*, ..., s_i, ..., s_n^*) \tag{1}$$

for alle spillere $i$ og for alle $s_i \in S_i$. Nash-ligevægten implicerer, at ingen spiller kan forbedre sit payoff ved at ændre sin strategi.

### Pure Strategy Nash Equilibrium

Ved Pure Strategy Nash Equilibrium (PSNE) skal forstås, det strategisæt en spiller vælger at benytte, hvis hendes optimale strategi udelukkende består af ét træk. I eksemplet fra forrige afsnit svarer det til, at spiller R altid vil spille "B", og spiller K vil spille "V". Betragter vi imidlertid en 3×3 matrix som nedenfor, er det ikke muligt at benytte IESDS til at finde resultatet, da der ikke er nogen stærkt dominerede strategier.

| | | Spiller $K$ | | |
|---|---|---|---|---|
| | | $V$ | $M$ | $H$ |
| Spiller $R$ | $T$ | $1,0$ | $1,2$ | $3,1$ |
| | $C$ | $2,0$ | $0,1$ | $4,5$ |
| | $B$ | $0,3$ | $2,1$ | $2,0$ |

Hver spiller finder sit best-response til alle modspillerens strategier. I dette tilfælde vil spiller R's best-response til spiller K's "V" være "C". Ligeledes vil det være "B", hvis spiller K spiller "M". På samme måde vil spiller K spille "M" til "T" osv. Vi illustrerer dette ved at understrege de givne payoffs:

| | | Spiller $K$ | | |
|---|---|---|---|---|
| | | $V$ | $M$ | $H$ |
| Spiller $R$ | $T$ | $1,0$ | $1,\underline{2}$ | $3,1$ |
| | $C$ | $\underline{2},0$ | $0,1$ | $\underline{4},\underline{5}$ |
| | $B$ | $0,\underline{3}$ | $\underline{2},1$ | $2,0$ |

Hvis der eksisterer et PSNE, findes det ved strategierne til de payoffs, hvor R og K best-responder på hinanden. Vi skriver, at PSNE er "$\{(C,H)\}$". Der findes også spil, hvor der er flere ligevægte. Et eksempel på dette ses i spillet "Battle of the sexes":

| | | Spiller $K$ | |
|---|---|---|---|
| | | $V$ | $H$ |
| Spiller $R$ | $T$ | $\underline{2}, \underline{1}$ | $0, 0$ |
| | $B$ | $0, 0$ | $\underline{1}, \underline{2}$ |

Med PSNE $= \{(T, V), (B, H)\}$. Vi har ligeledes spil hvor der ikke findes PSNE:

| | | Spiller $K$ | |
|---|---|---|---|
| | | $V$ | $H$ |
| Spiller $R$ | $T$ | $2, \underline{2}$ | $\underline{2}, 0$ |
| | $B$ | $\underline{3}, 0$ | $0, \underline{9}$ |

Til disse spil skal vi altså introducere en ny metode for at kunne finde en løsning.

#### Mixed Strategy Nash Equilibrium

Ved Mixed Strategy Nash Equilibrium (MSNE) skal forstås, det strategisæt en spiller vælger at benytte, hvis hun benytter sig af flere strategier med en vis sandsynlighed på hver. I eksemplet fra afsnit 3.1, hvor der ikke fandtes et PSNE, svarer det til, at spiller R laver et miks af "T" og "B". Man kan tænke på det som, at hvis 100 spillerpar spillede dette spil, og spiller R benytter en mixed strategy, hvor hun spiller "T" med sandsynligheden $p_T = 0,82$ og "B" med $p_B = 0,18$, ville det svare til, at 82 af spillerne vælger strategi "T", og 18 af dem vælger strategi "B". Et spil kan godt have både ét eller flere PSNE'er samt MSNE'er. Afsnit 3.2.1 nedenfor gennemgår, hvordan man løser MSNE for simple spil.

### 3.2.1 Generel løsning

Vi betragter et spil med to spillere: R og K, der i et statisk spil hver har tre mulige handlinger. Dette giver en 3x3 matrix med ni mulige udfald. $x_{ij}$ angiver payoff til spiller R og $y_{ij}$ til spiller K, hvor $i$ angiver spiller R's strategi, og $j$ angiver spiller K's strategi med sandsynlighederne $P$ og $Q$ for $i, j \in \{1, 2, 3\}$:

$$P = \mathbf{p} = (p_1, p_2, p_3)^T, \ p_i \geq 0, \text{for } i = 1, 2, 3 \text{ og } \sum_1^3 p_i = 1$$
$$Q = \mathbf{q} = (q_1, q_2, q_3)^T, \ q_j \geq 0, \text{for } j = 1, 2, 3 \text{ og } \sum_1^3 q_j = 1$$

Hvilket illustreres i en payoff-matrix, hvor spiller R vælger række, mens spiller K vælger kolonne:

**Tabel 3.2:** 3x3 Payoff-Matrix

| | | K | | |
|---|---|---|---|---|
| | | $j=1$ | $j=2$ | $j=3$ |
| R | $i=1$ | $(x_{11}, y_{11})$ | $(x_{12}, y_{12})$ | $(x_{13}, y_{13})$ |
| | $i=2$ | $(x_{21}, y_{21})$ | $(x_{22}, y_{22})$ | $(x_{23}, y_{23})$ |
| | $i=3$ | $(x_{31}, y_{31})$ | $(x_{32}, y_{32})$ | $(x_{33}, y_{33})$ |

Hvis $p_1 = 1 \vee p_2 = 1 \vee p_3 = 1$, vil der være tale om en pure strategy for spiller R, mens enhver kombination af strategier, hvor $0 < p_1 < 1 \vee 0 < p_2 < 1 \vee 0 < p_3 < 1$, vil resultere i en mixed strategy. Ligeledes gælder for spiller K, at $q_1 = 1 \vee q_2 = 1 \vee q_3 = 1$ resulterer i en pure strategy, mens $0 < q_1 < 1 \vee 0 < q_2 < 1 \vee 0 < q_3 < 1$ resulterer i en mixed.

Hver spiller vil da vælge den kombination af hhv. **p** for R og **q** for K, der gør, at modspilleren *ikke* kan udnytte dem. Spiller R undgår at blive udnyttet af spiller K, hvis hun benytter en strategi, hvor hendes forventede payoff er det samme for alle spiller K's strategier.

Spiller R prøver altså at maksimere sit payoff, givet at spiller K spiller **q**, mens spiller K maksimerer sit payoff, givet at spiller R spiller **p**. Vi opstiller da nedenstående matrix:

**Tabel 3.3:** 3x3 Payoff-Matrix Med Sandsynligheder

| | | K | | |
|---|---|---|---|---|
| | | $q_1$ | $q_2$ | $q_3$ |
| R | $p_1$ | $(x_{11}, y_{11})$ | $(x_{12}, y_{12})$ | $(x_{13}, y_{13})$ |
| | $p_2$ | $(x_{21}, y_{21})$ | $(x_{22}, y_{22})$ | $(x_{23}, y_{23})$ |
| | $p_3$ | $(x_{31}, y_{31})$ | $(x_{32}, y_{32})$ | $(x_{33}, y_{33})$ |

Spiller R's forventede payoff ved at spille hhv. $i = 1$, $i = 2$ og $i = 3$ er givet ved:

$$E_R[1] = q_1 \cdot x_{11} + q_2 \cdot x_{12} + (1 - q_1 - q_2) \cdot x_{13}$$
$$E_R[2] = q_1 \cdot x_{21} + q_2 \cdot x_{22} + (1 - q_1 - q_2) \cdot x_{23}$$
$$E_R[3] = q_1 \cdot x_{31} + q_2 \cdot x_{32} + (1 - q_1 - q_2) \cdot x_{33}$$

Hvor $q_1$ og $q_2$ angiver sandsynlighederne for, at spiller K spiller hhv. strategi 1 og 2. Sandsynligheden for at spille strategi 3 defineres ved $q_3 = 1 - q_1 - q_2$.

Hvis spiller R ikke skal kunne blive udnyttet af spiller K, skal hendes forventede payoff af disse tre strategier være lig hinanden: $E_R[1] = E_R[2] = E_R[3]$, hvilket da kan løses som et ligningsystem. På samme måde løser spiller K ligningsystemet $E_K[1] = E_K[2] = E_K[3]$, hvor:

$$E_K[1] = p_1 \cdot y_{11} + p_2 \cdot y_{21} + (1 - p_1 - p_2) \cdot y_{31}$$
$$E_K[2] = p_1 \cdot y_{12} + p_2 \cdot y_{22} + (1 - p_1 - p_2) \cdot y_{32}$$
$$E_K[3] = p_1 \cdot y_{13} + p_2 \cdot y_{23} + (1 - p_1 - p_2) \cdot y_{33}$$

Hvor $p_1$ og $p_2$ angiver sandsynlighederne for, at spiller R spiller hhv. strategi 1 og 2. Sandsynligheden for at spille strategi 3 defineres ved $p_3 = 1 - p_2 - p_1$. Dette resulterer i de optimale mixed strategies: $\mathbf{p}^* = (p_1^*, p_2^*, p_3^*)$ og $\mathbf{q}^* = (q_1^*, q_2^*, q_3^*)$.

Den samme metode kan udvides til en m $\times$ n-matrix, hvor der er $n$ ligninger med $n - 1$ ubekendte. Bemærk, at antagelsen om at ens modstander optimerer på samme vis som en selv, og begge er fuldt rationelle, er essentiel for Nash-ligevægten. I det kommende eksempel vil vi betragte en $3 \times 3$ matrix og gennemgå løsningsprocessen for en mixed Nash-ligevægt.

### 3.2.2 Eksempel

Vi vil nu løse et $3 \times 3$ matrixspil for den optimale mixed Nash-strategi. Betragt payoff-matricen i tabel 3.4,[14] hvor der ikke eksisterer et PSNE.

**Tabel 3.4:** 3x3 Payoff-Matrix Eksempel

| | | K | | |
|---|---|---|---|---|
| | | $q_1$ | $q_2$ | $q_3$ |
| R | $p_1$ | (0.50, 0.50) | (0.46, 0.54) | (0.56, 0.44) |
| | $p_2$ | (0.54, 0.46) | (0.50, 0.50) | (0.40, 0.60) |
| | $p_3$ | (0.44, 0.56) | (0.60, 0.40) | (0.50, 0.50) |

[14] Uddrag af aktuel data.

Her er spiller R's forventede payoffs af at spille hhv. 1, 2 eller 3:

$$E_R[1] = q_1 \cdot 0.50 + q_2 \cdot 0.46 + (1 - q_1 - q_2) \cdot 0.56$$
$$E_R[2] = q_1 \cdot 0.54 + q_2 \cdot 0.50 + (1 - q_1 - q_2) \cdot 0.40$$
$$E_R[3] = q_1 \cdot 0.44 + q_2 \cdot 0.60 + (1 - q_1 - q_2) \cdot 0.50$$

Hvoraf det følger:

$$E_R[1] = 0.56 - 0.06q_1 - 0.1q_2$$
$$E_R[2] = 0.40 + 0.14q_1 + 0.1q_2$$
$$E_R[3] = 0.50 - 0.06q_1 + 0.1q_2$$

Når $E_R[1] = E_R[2] = E_R[3]$ findes værdierne $\mathbf{q}^* = (q_1^*, q_2^*, q_3^*) = (0.5, 0.3, 0.2)$, hvilket da er spiller K's optimale mixed strategy, således at han er indifferent over for spiller R's strategivalg. Da vi betragter et symmetrisk zero-sum game, vil spiller K have præcis samme vægte: $\mathbf{q}^* = (q_1^*, q_2^*, q_3^*) = (0.5, 0.3, 0.2)$.[15]

I det kommende afsnit vil vi gennemgå løsningen af spil ved minimax-metoden, der anvender en algoritme til at løse større spil efficient.

## 3.3 Løsning med minimax strategier

John von Neumann var den første, der fremviste minimax-teoremet, med hans bog *Theory of Parlor Games* i 1928. Netop dette teorem, som omhandler optimale løsninger til to-personers nulsumsspil, blev brugt som grundlag til George Dantzig's *Simplex Algorithm* fremført i 1947. Her beskrev han en metode til effektivt at løse linære optimeringsproblemer, som kan anvendes til at løse MSNE i disse spil.

Eftersom minimax-metoden er et specialtilfælde af John Nash's teorem, samt at der er en velfunderet matematisk tilgang til at løse optimiseringsproblemer af denne type, har vi lagt fokus på at benytte denne til løsning af MSNE i vores afhandling. Endvidere er det beregningstungt og inefficient at løse Nash-ligevægte for større spil (Daskalakis [2013]),

[15] Kan løses på samme måde som spiller R.

hvorfor det er oplagt at omdanne dem til linære programmer og løse ved brug af simplex algoritmen. Dantzig [1963] beskriver minimax-teoremet formelt ved:

Lad $\mathbf{X} \subset \mathbb{R}^n$ og $\mathbf{Y} \subset \mathbb{R}^m$ være kompakte og konvekse sæt af mixed-strategier. Hvis f: $X \times Y \to \mathbb{R}$ er en kontinuert funktion, der er konkav-konveks, altså:

$$f(\cdot, y) : \mathbf{X} \to \mathbb{R} \text{ er konkav for et givent y}$$

og

$$f(x, \cdot) : \mathbf{Y} \to \mathbb{R} \text{ er konveks for et givent x}$$

Så har vi, at

$$\max_{x \in X} \min_{y \in Y} \mathbf{X}^T \mathbf{A} \mathbf{Y} = \min_{y \in Y} \max_{x \in X} \mathbf{X}^T \mathbf{A} \mathbf{Y} = v$$

Hvor $\mathbf{A}$ er payoff-matricen, og $v$ er værdien af spillet. I et to-personers nulsumsspil er dette ækvivalent til Nash-ligevægten. Med udgangspunkt i Ferguson [2000] opstiller vi følgende eksempel, hvor vi løser en $3 \times 3$ matrix ved brug af minimax-teoremet. Metoden kan dog nemt udvides til $m \times n$ matricer.

Vi betragter to spillere: R(række) og K(kolonne), der har strategirum på hhv. $P(p_1, p_2, p_3)$ og $Q(q_1, q_2, q_3)$. Spillernes strategier skal være større eller lig nul samt summe til 1:

$$P = \mathbf{p} = (p_1, p_2, p_3)^T, \ p_i \geq 0, \text{for } i = 1, 2, 3 \text{ og } \sum_1^3 p_i = 1$$
$$Q = \mathbf{q} = (q_1, q_2, q_3)^T, \ q_j \geq 0, \text{for } j = 1, 2, 3 \text{ og } \sum_1^3 q_j = 1$$

Payoff-matricen for hver spiller er givet ved:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Hvis vi antager, at spiller K spiller tilfældigt med $q \in Q$, så vil spiller R's payoff ved at spille strategi $i$ være:

$$\sum_{j=1}^{3} a_{ij} q_j = (\mathbf{Aq})_i \tag{2}$$

og ligeledes hvis spiller R spiller $p \in P$, og spiller K spiller et vilkårligt $q_j$, vil spiller K's payoff være:

$$\sum_{i=1}^{3} p_i a_{ij} = (\mathbf{p}^T \mathbf{A})_j \tag{3}$$

Hvis begge spiller en vilkårlig strategi, er det gennemsnitlige payoff for spiller R:

$$u_R(p_i, q_j) = \sum_{i=1}^{3} \sum_{j=1}^{3} p_i a_{ij} q_j = \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{4}$$

Hvis vi antager, at spiller K spiller en strategi $q \in Q$, så vil spiller R vælge den række $i$, der maksimerer hendes forventede payoff:

$$\max_i \sum_{j=1}^{3} a_{ij} q_j = \max_{p \in P} \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{5}$$

hvor det $p \in P$ sikrer, at maksimum er en (ren) best-response strategi mod $q$.

Ligeledes hvis spiller K ved, at spiller R vil spille en specifik strategi $p$, så vil spiller K vælge den kolonne $j$, der maksimerer hans forventede payoff. Da vi beskæftiger os med nulsumsspil, vil maksimering af ens eget payoff være ækvivalent til at minimere modstanderens. Spiller K vil dermed spille sin optimale strategi, der minimerer modstanderens payoff:

$$\min_j \sum_{i=1}^{3} p_i a_{ij} = \min_{q \in Q} \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{6}$$

hvor det $q \in Q$ sikrer, at minimum er en (ren) best-response strategi mod $p$.

Hvis vi antager, at spiller K spiller en mixed strategi $\mathbf{q} \in Q$, vil spiller R spille best-response til dette, hvorfor at K vil spille det $\mathbf{q}$, der minimerer spiller R's payoff:

$$\overline{V} = \min_{\mathbf{q} \in Q} \max_i \sum_{j=1}^{3} a_{ij} q_j = \min_{\mathbf{q} \in Q} \max_{\mathbf{p} \in P} \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{7}$$

hvor $\overline{V}$ kaldes den øvre værdi af spillet og angiver det mindste gennemsnitlige tab, som spiller K kan sikre, uanset hvad spiller R gør. Ethvert $\mathbf{q} \in Q$, der minimerer $\overline{V}$, kaldes en minimax-strategi for spiller K. Hvis spiller K spiller en minimax-strategi, minimerer han sit maksimale tab og derved sit gennemsnitlige tab.

En lignende strategi finder sted for spiller R, som vil maksimere sit payoff givet spiller K's strategi:

$$\underline{V} = \max_{\mathbf{p} \in P} \min_{j} \sum_{i=1}^{3} p_i a_{ij} = \max_{\mathbf{p} \in P} \min_{\mathbf{q} \in Q} \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{8}$$

hvor $\underline{V}$ kaldes den nedre værdi af spillet og angiver den maksimale gennemsnitlige gevinst, som spiller R kan sikre, uanset hvad spiller K gør. Ethvert $\mathbf{p} \in P$, der maksimerer $\underline{V}$, kaldes en minimax-strategi for spiller R. Det er det maksimale payoff, som spiller R kan garantere, givet spiller K's strategi.

I et symmetrisk spil kan hver spiller se sig selv som enten R eller K, hvorfor den omvendte terminologi gælder for begge. Von Neumann's minimax-teorem siger, at for endelige spil vil begge spillere have minimax-strategier, hvor:

$$\underline{V} = \overline{V}$$

Altså hvor begge spillere er i ligevægt og opnår samme payoff, og dermed er strategierne ækvivalente:

$$\max_{\mathbf{p} \in P} \min_{\mathbf{q} \in Q} \mathbf{p}^T \mathbf{A} \mathbf{q} = \min_{\mathbf{q} \in Q} \max_{\mathbf{p} \in P} \mathbf{p}^T \mathbf{A} \mathbf{q} \tag{9}$$

Dantzig [1963] viser, at de optimale strategier i (9) kan løses ved at opstille ligningen som et lineært programmeringsproblem. Vi har derfor optimeringsproblemet for spiller R:

$$\underbrace{\max_{\mathbf{p} \in P}}_{\text{R's strategi}} \underbrace{\min_{j} \overbrace{\sum_{i=1}^{3} p_i a_{ij}}^{\text{R's payoff}}}_{\text{K's strategi}} \tag{10}$$

betinget på

$$p_1 + p_2 + p_3 = 1$$

og

$$1 \geq p_i \geq 0 \; \forall \; i = 1, 2, 3$$

Dette kommer af, at spiller R ved, at spiller K vælger det $j$, der minimerer spiller R's payoff. Netop dette payoff forsøger R da at maksimere ved at vælge det optimale $\mathbf{p} \in P$. Da kriteriefunktionen i (10) ikke er lineær af natur, omskrives denne til en lineær funktion, hvor vi indsætter $v \leq \min_j \sum_{i=1}^{3} p_i a_{ij}$ som en betingelse og substituerer:

$$\max v \tag{11}$$

betinget på

$$\begin{gathered} v \leq \textstyle\sum_{i=1}^{3} p_i a_{i1}, \\ v \leq \textstyle\sum_{i=1}^{3} p_i a_{i2}, \\ v \leq \textstyle\sum_{i=1}^{3} p_i a_{i3}, \\ p_1 + p_2 + p_3 = 1 \text{ og} \\ 1 \geq p_i \geq 0 \; \forall \; i = 1, 2, 3 \end{gathered}$$

hvor de nye bibetingelser sikrer, at spiller R's valg af $\mathbf{p}$ ikke kan få $v$ til at blive større end de mulige udfald for alle K's mulige valg af $j$. Et identisk problem kan ligeledes opstilles for spiller K.

Williams [1966] gennemgår en pivoteringsmetode, som kan løse disse lineære programmeringsproblemer. Denne metode udgør de bagvedliggende beregninger for flere numeriske optimisatorer, der findes online, hvor vi benytter SciPy optimize.linprog. Til forståelse af optimeringsprocessen gennemgås da denne pivoteringsalgoritme i algoritme 1 nedenfor.

**Algoritme 1:** Beskrivelse af algoritme til løsning af $M \times N$ spil

Vi betragter matricen **A**:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & .... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mn} \end{pmatrix}$$

**Trin 1**: Tilføj eventuelt en vilkårlig værdi til alle elementerne i matricen, således at værdien af spillet er $\geq 0$.

**Trin 2**: Lav en tabel ved at tilføje en række med -1, en kolonne med +1 og 0 i det nedre højre hjørne. Spiller R's strategier navngives $x_1, x_2, ..., x_m$ og spiller K's strategier som $y_1, y_2, ..., y_n$ som vist i illustrationen:

| | $y_1$ | $y_2$ | $y_3$ | ... | $y_n$ | |
|---|---|---|---|---|---|---|
| $x_1$ | $a_{11}$ | $a_{12}$ | $a_{13}$ | ... | $a_{1n}$ | 1 |
| $x_2$ | $a_{21}$ | $a_{22}$ | $a_{32}$ | ... | $a_{2n}$ | 1 |
| ... | ... | ... | ... | ... | ... | 1 |
| $x_m$ | $a_{m1}$ | $a_{m2}$ | $a_{m3}$ | ... | $a_{mn}$ | 1 |
| | -1 | -1 | -1 | -1 | -1 | 0 |

**while** *Mindst ét element i række* $m+1$ *er negativt* **do**

**Trin 3**: Vælg et vilkårligt element, der opfylder tre kriterier (række = r, kolonne = k):

**3.a**: Selve pivoten, $a_{rk}$, skal være positiv.

**3.b**: $a_{(m+1)k}$ skal være negativ.

**3.c**: Elementet skal have den laveste ratio til kolonne n+1 i k, altså: $min(\frac{a_{r(n+1)}}{a_{rk}})$ for alle positive elementer i kolonnen.

**Trin 4**: Pivoter som følger, med $a_{11}$ som pivoten:

**4.a**: Erstat alle elementer $a_{ij}$, der ikke er på samme række eller kolonne som pivoten med $a_{ij} - \frac{a_{rj}a_{ik}}{a_{rk}}$, hvor $i$ og $j$ angiver hhv. rækken og kolonnen for elementet, der erstattes.

**4.b**: Erstat alle elementer $a_{ij}$, der er på samme række som pivoten med $\frac{a_{ik}}{a_{rk}}$

**4.c**: Erstat alle elementer $a_{ij}$, der er på samme kolonne som pivoten med $-\frac{a_{rj}}{a_{rk}}$

**4.d**: Erstat pivoten med $\frac{1}{a_{rk}}$

**Trin 5**: $y_k$ og $x_r$ bytter position. Efter én iteration ses den transformerede tabel:

| | $x_1$ | $y_2$ | $y_3$ | ... | $y_n$ | |
|---|---|---|---|---|---|---|
| $y_1$ | $\frac{1}{a_{11}}$ | $\frac{a_{12}}{a_{11}}$ | $\frac{a_{13}}{a_{11}}$ | ... | $\frac{a_{1n}}{a_{11}}$ | $\frac{1}{a_{11}}$ |
| $x_2$ | $-\frac{a_{21}}{a_{11}}$ | $a_{22} - \frac{a_{12}a_{21}}{a_{11}}$ | $a_{23} - \frac{a_{13}a_{21}}{a_{11}}$ | ... | $a_{2n} - \frac{a_{1n}a_{21}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{21}}{a_{11}}$ |
| $x_3$ | $-\frac{a_{31}}{a_{11}}$ | $a_{32} - \frac{a_{12}a_{31}}{a_{11}}$ | $a_{33} - \frac{a_{13}a_{31}}{a_{11}}$ | ... | $a_{3n} - \frac{a_{1n}a_{31}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{31}}{a_{11}}$ |
| ... | ... | ... | ... | ... | ... | $1 - \frac{a_{1(n+1)}a_{k1}}{a_{11}}$ |
| $x_m$ | $-\frac{a_{m1}}{a_{11}}$ | $a_{m2} - \frac{a_{12}a_{m1}}{a_{11}}$ | $a_{m3} - \frac{a_{13}a_{m1}}{a_{11}}$ | ... | $a_{mn} - \frac{a_{1n}a_{m1}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{m1}}{a_{11}}$ |
| | $\frac{-1}{a_{11}}$ | $-1 - \frac{a_{12}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{13}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{1r}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{1n}a_{(m+1)1}}{a_{11}}$ | $0 - \frac{a_{1(n+1)}a_{(m+1)1}}{a_{11}}$ |

Note: Bemærk at en større og mere letlæselig version af tabellen kan ses i Appendix A.1.

**end**

Spiller R's strategi er som følger: Alle $x$, der er på toppen af tabellen, er spiller R's vægte. Hvis spiller R har variable på venstresiden af tabellen, har de en vægt på 0.

Spiller K's strategi er som følger: Alle $y$, der er på venstre side af tabellen, er spiller K's vægte. Hvis spiller K har variable på toppen af tabellen, har de en vægt på 0.

**Eksempel**

I nedenstående eksempel betragter vi matrix 3.4 fra afsnit 3.2.1, som er et udsnit fra vores datasæt. Vi opstiller programmeringsproblemet og løser til sidst med algoritme 1. Vi skriver minimax-strategierne som et lineært programmeringsproblem:

$$\max v$$

betinget på

$$v \leq 0.5p_1 + 0.46p_2 + 0.56p_3$$
$$v \leq 0.54p_1 + 0.5p_2 + 0.4p_3$$
$$v \leq 0.445p_1 + 0.6p_2 + 0.5p_3$$
$$p_1 + p_2 + p_3 = 1$$
$$1 \geq p_i \geq 0 \ for \ i = 1, 2, 3$$

Da alle elementerne i matricen er positive, er **trin 1** opfyldt. Vi opsætter herefter tabellen beskrevet ved **trin 2** og vælger 0.54 i **trin 3**:

| | $y_1$ | $y_2$ | $y_3$ | |
|---|---|---|---|---|
| $x_1$ | 0.50 | 0.46 | 0.56 | 1.00 |
| $x_2$ | (0.54) | 0.50 | 0.40 | 1.00 |
| $x_3$ | 0.44 | 0.60 | 0.50 | 1.00 |
| | -1 | -1 | -1 | 0 |

Vi pivoterer som i **trin 4** og skifter placeringen af $y_1$ og $x_2$:

| | $x_2$ | $y_2$ | $y_3$ | |
|---|---|---|---|---|
| $x_1$ | -0.93 | -0.003 | 0.190 | 0.074 |
| $y_1$ | 1.852 | 0.926 | 0.741 | 1.852 |
| $x_3$ | -0.815 | (0.193) | 0.174 | 0.185 |
| | 1.852 | -0.074 | -0.259 | 1.852 |

Da der fortsat er to negative værdier i den nederste række gentager vi pivoteringen ved at vælge en ny pivot (**trin 3**) og pivoterer som beskrevet i **trin 4**:

| | $x_2$ | $x_3$ | $y_3$ | |
|---|---|---|---|---|
| $x_1$ | -0.938 | 0.015 | 0.192 | 0.077 |
| $y_1$ | 5.769 | -4.808 | -0.096 | 0.962 |
| $x_3$ | -4.231 | 5.192 | 0.904 | 0.962 |
| | 1.538 | 0.385 | -0.192 | 1.923 |

Der er fortsat en negativ værdi i den nederste række, og vi gentager derfor pivoteringen. Vi opnår resultatet i nedenstående tabel:

| | $x_2$ | $x_3$ | $x_1$ | |
|---|---|---|---|---|
| $y_3$ | -4.88 | 0.08 | 5.2 | 0.4 |
| $y_1$ | 5.3 | -4.8 | 0.5 | 1 |
| $y_2$ | 0.18 | 5.12 | -4.7 | 0.6 |
| | 0.6 | 0.4 | 1 | 2 |

Ved løsning af problemet finder vi, at vægtene for spiller R er:

$$(x_1, x_2, x_3) = (\tfrac{1}{1+0.6+0.4}, \tfrac{0.6}{1+0.6+0.4}, \tfrac{0.4}{1+0.6+0.4}) = (0.5, 0.3, 0.2)$$

og ligeledes for spiller K:

$$(y_1, y_2, y_3) = (\tfrac{1}{1+0.6+0.4}, \tfrac{0.6}{1+0.6+0.4}, \tfrac{0.4}{1+0.6+0.4}) = (0.5, 0.3, 0.2)$$

Som er de samme vægte, vi fandt i eksemplet fra afsnit 3.2. Imidlertid er algoritme 1 betydeligt mere efficient for større spil, hvor der kan vælges mange strategier. Afsnit 4 nedenfor beskriver spillet Hearthstone, og hvorfor vi kan opstille det som en $20 \times 20$ payoff-matrix.

# 4 Hearthstone

Activision Blizzard er en af verdens største spilgiganter. De står for udviklingen af kendte spiltitler som World of Warcraft, Diablo og Call of Duty, men ejer også mange mobilspil, blandt andre Candy Crush Saga. Da spilindustrien har udviklet sig fra en omsætning på 64,5 milliarder dollars i 2009 til 108,9 milliarder dollars i 2017,[16] og er estimeret til at overstige 200 milliarder dollars i 2023,[17] kommer industrien i større grad i fokus hos medier, virksomheder og statslige enheder.

[16] I 2012-priser: https://vgsales.fandom.com/wiki/Video_game_industry

[17] https://www.reuters.com/article/esports-business-gaming-revenues-idUSFLM8jkJMl, https://www.statista.com/statistics/292056/video-game-market-value-worldwide/

I 2014 udgav Activision Blizzard spillet "Hearthstone", der sidenhen er blevet et af verdens mest spillede computerspil[18] med 100 millioner brugere i 2018.[19] Spillet er et turbaseret kortspil, hvor man spiller en-mod-en. Hver spiller sammensætter et "deck" bestående af 30 kort. Da der findes flere hundrede kort, er der utallige kombinationer af deck, men grundet opsætningen af spillet er kombinationen af visse kort bedre end andre. Dette medfører, at antallet af deck, der bliver spillet, bliver begrænset. I denne analyse anvendes "deck" og "strategi" i flæng, da de to er ens.

Hvis man vinder et spil i Hearthstone, bliver man belønnet med en stjerne, hvorimod man bliver fratrukket en stjerne, hvis man taber. Bliver kampen uafgjort, hverken får eller taber man en stjerne - der er altså tale om et nulsumsspil.

Hearthstone er sæsonbaseret, således at der tre gange om året udkommer nye kort, hvorfor de decktyper, der bliver spillet, ændres ved sæsonskiftet. I starten af hver sæson vil der være en uvished om, hvor godt nogle decktyper klarer sig mod andre. I slutningen af en sæson vil der dog være klarhed omkring, hvor gode de forskellige deck er mod hinanden. Et deck kan nemlig være godt mod én slags decktype men ringe mod en anden. Hvor mange forskellige slags decks, der bliver spillet, varierer fra sæson til sæson, men ligger oftest mellem 14 og 22 forskellige decks. I vores datasæt benyttes 20.

Hver spiller vælger sin strategi, *inden* hun oplyses om modstanderens. Da hvert deck er godt mod nogle decks og dårligt mod andre, kan en portefølje på 20 decks hurtigt blive uoverskueligt, når man skal vælge, hvilken strategi man vil spille. Grundet den enorme spillerbase findes der flere hjemmesider på nettet, der indsamler data over, hvor meget alle slags deck bliver spillet, og hvor ofte de vinder mod andre. Denne sejrsprocent vil vi i opgaven betegne som en "winrate".[20] Det er klart, at to identiske decks, der spiller mod hinanden, har en winrate på 50 pct., mens winraten mod de resterende 19 deck er tilgængelige på websider som ViciousSyndicate [2020] og HSReplay [2020]. Med dette data

[18] Spillet kan dog også spilles via. smartphone eller tablet.

[19] https://www.statista.com/statistics/323239/number-gamers-hearthstone-heroes-warcraft-worldwide/ - Dog er det muligt, at alle af disse ikke er aktive.

[20] Winraten angiver da, hvor ofte et givent deck vinder over et andet. En winrate på 62 pct. vinder da 62 ud af 100 spil.

kan man altså opstille en $20 \times 20$ winrate/payoff-matrix.

Når en spiller vil vælge, hvilket deck hun vil spille, vil det være logisk at vælge det deck, der har den højeste gennemsnitlige winrate, og dermed kan man opnå det højeste gennemsnitlige payoff. Da modspilleren muligvis også har disse tanker, vil det kunne betale sig at divergere fra at spille det deck med den højeste winrate, kald det $s_i^*$, til at spille det deck med den højeste winrate mod $s_i^*$, og vi befinder os da i et statisk spil, hvor hver spiller har 20 mulige strategier.

Hvis alle 100 millioner spillere var fuldt rationelle, ville de jf. afsnit 3.2 spille Nash-ligevægten. Afsnit 9.2 finder dette MSNE, tester hvor godt det passer på, hvordan populationen rent faktisk spiller, og finder at ikke alle tænker sådan. Afsnit 6.1 og 6.2 opstiller da alternative modeller, der prøver at modellere spillernes adfærd, hvor resultaterne heraf ses i afsnit 9.4. I afsnit 5 nedenfor gennemgår vi dataindsamlingsprocessen og potentielle fejlkilder hertil.

# 5 Data

Grundet den store brugerbase, som omtalt i afsnit 4, findes der mange hjemmesider, som samler og analyserer brugernes adfærd samt lagrer det. Vores data stammer primært fra ViciousSyndicate [2020] og HSReplay [2020]. Disse hjemmesider indsamler data gennem en applikation, som Hearthstonespillere har kørende i baggrunden, imens de spiller. Denne applikation registrerer, hvilket deck de spiller, hvilket deck de spiller mod, og om de vinder eller taber. Applikationen er designet sådan, at den hjælper spillere med at holde styr på visse ting inde i spillet - dette for at spilleren har et incitament for at downloade appen. Da nok spillere benytter disse applikationer, er det muligt at danne et præcist overblik over, hvilke deck der bliver spillet, og hvor meget de bliver spillet. Endvidere gør de det muligt at differentiere imellem spillernes beslutninger over tid og afhængigt af niveau. En spiller har nemlig en såkaldt "Rank", som spænder mellem seks ligaer, hvor 1 er bedst.[21]

Hearthstone lancerer en ny udgivelse ca. tre gange om året, hvor de deck, der bliver

[21] Bemærk, at der som nævnt, skal en aktiv handling til for at have applikationen, hvorfor der findes markant flere observationer for bedre ligaer.

spillet, ændres markant. En udvidelse varer da ca. 4 måneder. I starten af en udvidelse vil der forekomme mange eksperimentelle deck, da brugerbasen endnu ikke har fundet ud af, hvilke kortkombinationer der passer godt sammen, og hvordan man skal spille de forskellige deck. Antallet af deck, der anvendes, vil dog indskrænkes over tid, hvor den winrate, vi observerer i data, vil konvergere mod den sande winrate imellem de forskellige deck.[22] Vi benytter derfor i hoveddelen af vores analyse data fra den sidste uge i en given udvidelse, da der på dette tidspunkt er størst sandsynlighed for, at antallet af deck, winrate osv. er konvergeret.

Vi anvender data fra udvidelsen ("Ashes of Outland"), der var aktiv fra 7. april til 6. august 2020. I denne sæson eksisterer der 20 forskellige strategier, der har spillet mere end 50 kampe mod hinanden. Dette resulterer i, at vi observerer en $20 \times 20$ winrate-matrix, såvel som en matrix der angiver antallet af kampe, som hvert deck har spillet. Winrate-matricen fremgår af tabel 8.1 i afsnit 8. De aktuelle observationer, der angiver antallet af kampe for hver strategi, kan ses af tabel 8.2 i afsnit 8.

Indsamlingsprocessen og vores antagelser, benyttet til at opstille data som et spilteoretisk problem, kan potentielt medføre en række problemer for analysen. I det kommende underafsnit 5.1 vil vi påpege nogle af disse. Afsnit 10 vil yderligere diskutere, hvilke implikationer dette kan have for vores resultater.

## 5.1 Potentielle fejlkilder

### 5.1.1 Bortfaldsproblematikker

Dataindsamlingen foregår gennem en tredjepartsapplikation, der downloades eksternt til Hearthstone. Dette giver anledning til nogle bortfaldsproblematikker, som skaber skævhed i datamaterialet, og derved påvirker repræsentativiteten. Eksempelvis er det ikke usandsynligt, at der er en stærk korrelation mellem individers erfaring med Hearthstone og kendskab til disse tredjepartsprogrammer, hvorfor der kunne være en overrepræsentation af erfarne spillere. Dette leder til, at nye spillere, uden erfaring og kendskab til disse tredjepartsprogrammer, potentielt underrepræsenteres i data. Dette behøver imidlertid ikke nødvendigvis at være en dårlig egenskab, da det kan styrke vores antagelse om, at man

[22] Store tals lov.

"spiller for at vinde". Individer, der downloader disse tredjepartsprogrammer, har udvist en lyst til at forbedre sit spil ved at tilegne sig en overskuelig metode til at holde styr på personlige statistikker mm.

### 5.1.2 Budgetrestriktioner

I afhandlingen antager vi, at alle spillere har adgang til alle decks, der er angivet i winrate-matricen i tabel 8.1 i afsnit 8. Dette er imidlertid *ikke* en selvfølge, da man ikke per automatik har adgang til deckene. For at kunne benytte et deck skal spilleren først købe det. Dette kan gøres enten via. in-game valuta eller for virkelige penge. In-game valuta kan opnås ved kontinuert spil, hvorfor det potentielt kan tage lang tid at tilegne sig alle decks, og der opstår herved to effekter: Den første er, at spilleren potentielt vil tænke mere over sit strategivalg, da dette i et vist omfang begrænser hende i at skifte strategi. Det andet er, at hvis man først beslutter at købe et deck, vil der være store omkostninger ved at ændre strategi.[23] Som nævnt ovenfor er det muligt, at bortfaldsproblematikken medfører, at denne budgetbegrænsning ikke er lige så markant i det benyttede data, som den ville være for hele populationen, da vi potentielt observerer dedikerede spillere. Cleghorn og Griffiths [2015] argumenterer endvidere for, at der en korrelation mellem tid investeret i et spil og ens villighed til at foretage køb i spillet for rigtige penge. Disse to effekter styrker antagelsen om, at spillerne ikke er budgetbegrænsede.

### 5.1.3 Spilmekanikken udvikler sig

En Hearthstone-sæson varer omtrent fire måneder. I denne periode forekommer der ikke indskud af nye kort, der kan ændre winrate-matricen. Det hænder dog, at enkelte kort vurderes af spildesignerne til at være for stærke eller svage, hvorfor de kan blive justeret i en mindre opdatering. En sådan ændring vil ændre winrate-matricen for de decks, der påvirkes. Endvidere medfører det, at spillerne igen skal lære, hvad der er optimalt at spille, da der kan opleves fluktuationer i observationerne omkring ændringen. Disse ændringer forekommer oftest i starten af en sæson, men da vores analyse benytter data fra den sidste uge i sæsonen, minimerer vi risikoen for, at dette problem har en reel effekt.

[23] Bemærk at det er muligt for langt størstedelen af spillere at erhverve sig flere deck.

### 5.1.4 Sensation-seeking

Individer har en generel tilbøjelighed til at påtage sig risici i søgen på en bedre oplevelse af en begivenhed (Grinblatt og Keloharju [2009]). Dette kan overføres til Hearthstone. Spillere kan være mere tilbøjelige til at spille ikke-optimale decks i jagten på en anden form for gevinst. Man kan f.eks. bevidst spille dårlige decks, da tanken om at vinde over optimale deck med et dårligt deck vil bringe spilleren mere glæde.

Det er vigtigt at tage forbehold for disse potentielle fejlkilder, når resultaterne fortolkes. En mere detaljeret diskussion af konsekvenserne som følge af disse fremgår af afsnit 10. I det følgende afsnit vil teorien bag level-k og de kognitive hierarkimodeller, vi benytter i analysen, blive beskrevet.

# 6 Teori

I dette afsnit gennemgås relevant teori for de hierarkiske modeller, der anvendes i denne afhandling. Vi tager udgangspunkt i en level-$k$ model og udvider denne til en kognitiv hierarkimodel, der er det primære fokus i denne afhandling. Endvidere kommer vi ind på fordelingerne, der anvendes i de kognitive hierarkimodeller, samt antagelser om disse.

Crawford et al. [2013] fremlægger en litteraturoversigt, der illustrerer det stigende fokus på modeller, der forsøger at forudsige spil, hvor antagelser om ligevægt fejler. I størstedelen af litteraturen anvendes der en kognitiv hierarkimodel, som med udgangspunkt i en ustrategisk spiller beskriver spillere af højere kognitive niveauer, der påtager sig en eller flere strategiske iterationer. Camerer et al. [2004] finder, at agenter *oftest* foretager strategiske beslutninger, men selve graden af strategisk omtanke begrænses af ens kognitive egenskaber.

Et vigtigt koncept i modelleringen af disse typer af rationalitetsbegrænsede valg under usikkerhed er beslutningstagerens vurdering af sandsynlighederne for de mulige udfald, som hun ikke kan kontrollere. Disse sandsynligheder vil agenten da prøve at approksimere bedst muligt. Eksempelvis kan en agent, der prøver at forstå sandsynlighederne ved terningkast, opstille en model af sandsynlighederne for hvert udfald og måske konkludere, at

der må være en sandsynlighed på $\frac{1}{6}$ for hvert udfald. Agenten vil i dette tilfælde, i hvert fald for en fair terning, have dannet sig et korrekt billede af sandsynlighedsfordelingen. Hvis vi overfører dette til et problem med flere agenter, vil agenterne på samme vis danne sig en forestilling om fordelingen af andre agenter samt deres strategivalg.

Afsnit 6.1 og 6.2 opstiller disse modeller, hvori spillerne former forventninger til andre spilleres adfærd og best-responder givet denne forventning.

## 6.1 Level-K

Dette afsnit beskriver det teoretiske framework for vores Level-K model. Level-K modellen består af beslutningskriterier for spillere, der foretager k strategiske iterationer i en beslutningsproces. Modellen beskriver et simpelt framework, hvori spilleren best-responder til sin egen tro om modstanderne.

### 6.1.1 Modellen

Vores Level-K model har følgende to typer:

Level 0: Spillere foretager beslutninger uniformt på tværs af mulige strategier og kan dermed klassificeres som ustrategiske spillere. Det er muligt at udvide modellen, således at spillere af 0. orden spiller med en anden strategi, men der er en række gode egenskaber som følge af, at 0. ordens spillere agerer uniformt. Eksempelvis sikrer denne antagelse, at der er en positiv sandsynlighed på alle strategier i data.

Level k: Spillere foretager valg baseret på deres tro omkring andre individer, der deltager i beslutningsprocessen. Vi antager i denne model ligesom Costa-Gomes et al. [2001], at spillerne antager, at alle modstandere er præcis ét level under dem selv, hvorfor de vælger at best-responde netop til dette level. Eksempelvis best-responder en level 1 på den strategi en level 0 spiller, og en level 2 spiller vil best-responde på en level 1'er. Dette fortsætter i en (teoretisk) evig følge op til level k.

Det antages således, at spillere af k. level ikke indser, at modstanderen kan være af højere level end dem selv.[24] Dette bygger på antagelsen om overconfidence, som Huang og Kisgen [2013] definerer som en adfærdsmæssig egenskab, hvor agenter oftest overvurderer

[24] Eller bare på samme level.

deres egne egenskaber relativt til andre. I tidligere arbejde fra Camerer og Lovallo [1999] viser de, at individer udstråler konsistent overvurdering af egne evner på mange områder. Dette er konsistent med vores antagelse om Level-K modellen.

At spillere af niveau k best-responder til spillere af niveau $h = k - 1$ gør, at der er en række uheldige egenskaber ved modellen. Level-K modellen er et snapshot af en Cournot-model, da alle spillere antager, at andre spillere altid vil gentage deres valg. Dermed er det muligt for spillere at best-responde i en cyklisk følge, der gentager sig selv. Chong et al. [2005] fremhæver, at hvis spillere øger det antal trin, som de itererer over, så vil spillernes forestilling om de andre spillere divergere fra rationelle forventninger.

Beslutningskriterierne for en level-k model beskrives ved, at vi angiver en spiller $i$'s strategi $j$ som $s_i^j$ og antager, at spiller $i$ har $m_i$ strategier. Det antages, at spillere af niveau 0 spiller ustrategisk og dermed spiller uniformt tilfældigt med sandsynlighederne $p_0(s_i^j) = 1/m_i \ \forall \ j$ . Spillere af niveau $k \geq 1$ best-responder til niveau $h = k - 1$ med den strategi, der medfører det højeste forventede payoff.

Vi betragter modspilleren -$i$ og hans strategi, $s_{-i}^j$. Spiller $i$'s payoff afhænger af strategierne $s_i^j$ og $s_{-i}^j$ og betegnes dermed $u_i(s_i^j, s_{-i}^j)$. Da spiller $i$ forventer, at spiller -$i$ er af niveau $h = k - 1$, vil det forventede payoff for spiller $i$ af level $k$ for strategien $\bar{j}$ være:

$$E_k[u_i(s_i^{\bar{j}}, s_{-i}^j)] = \sum_{j=1}^{m_{-i}} u_i(s_i^{\bar{j}}, s_{-i}^j) \cdot p_h(s_{-i}^j) \tag{12}$$

Vi antager, at spiller $i$ best-responder i rene strategier, med sandsynligheden $p_k(s_i^{\bar{j}}) = 1$, hvis:

$$s_i^{\bar{j}} = \max_{s_i^j} E_k(u_i(s_i^j, s_{-i}^j)) \tag{13}$$

og $p_k(s_i^j) = 0$ for resterende strategier. Hvis der er flere ækvivalente payoffs, så mikser spiller $i$ (af level k $> 0$) uniformt blandt strategierne. For at illustrere den iterative proces bag Level-K modellen gennemgår vi et eksempel, der beskriver de optimale beslutninger, som spiller $i$ foretager.

**Eksempel**

Antag et 3x3 spil, hvor spiller $i$'s payoff-matrix er givet ved:

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 5 \\ 1 & 3 & 2 \\ 5 & 2 & 4 \end{pmatrix}$$

Med udgangspunkt i den symmetriske matrix $\mathbf{A}$ har vi at:
Level 0 spiller uniformt med sandsynlighed $p_0(s_i^j) = \frac{1}{3} \; \forall \; j \in \{1, 2, 3\}$.
Level 1 maksimerer sit payoff, givet strategien for level 0, hvor $u_i(s_i^1) = 3,33$, $u_i(s_i^2) = 2$ og $u_i(s_i^1) = 3,67$, hvorfor spiller $i$ vælger $p_1(s_i^3) = 1$, samt $p_1(s_i^2) = p_1(s_i^1) = 0$ da $s_i^3$ har det højeste forventede payoff.
Level 2 spiller $p_2(s_i^1) = 1$, da $u_i(s_i^1) = 5 > u_i(s_i^2) = 4 > u_i(s_i^2) = 2$ og da gives det højeste forventede payoff, når en spiller af level 1 spiller $p_1(s_i^3) = 1$.
Level 3 spiller dermed $p_3(s_i^3) = 1$, hvilket altså er den samme strategi som Level 1. Det ses da, at som $k$ stiger, har vi en cyklisk adfærd i strategierne for spiller $i$, da hvert andet level vil spille $s_i^1$, og resten spiller $s_i^3$, med undtagelse af spillere af level 0. Dette er et generelt resultat, som Chong et al. [2005] påpeger.

## 6.2 Kognitivt hierarki

I dette afsnit beskrives vores kognitive hierarkimodel (CH-model[25]). I en CH-model beskrives spillere ved et kognitivt niveau $k$ samt en frekvensfordeling af samtlige spillere under niveau $k$. Den grundlæggende model er baseret på arbejde af Camerer et al. [2004].

### 6.2.1 Modellen

CH-modellen består af beslutningsregler for spillere, der foretager k strategiske iterationer i en beslutningsproces. Modellen beskriver et mere komplekst framework end Level-K modellen. Beslutningsprocessen for spillerne består af en trinvis iteration af optimale beslutninger pba. spillerens forestilling om fordelingen af levels under dem selv og deres strategivalg. Ligesom ved Level-K modellen har vi to typer spillere:

Level 0: Spillere foretager beslutninger uniformt på tværs af mulige handlinger og kan

[25] CH = Cognitive Hierarchy

dermed anføres som ustrategiske spillere. Disse er identiske med level 0 i Level-K modellen beskrevet i afsnit 6.1 ovenfor.

Level k: Spillere af level $k \geq 1$ best-responder til en kombination af alle niveauer **h** under sig selv ($\mathbf{h} = (0, 1, 2, ..., h)$ hvor $h < k$). Dette på baggrund af en vurdering af fordelingen af andre spillere. Eksempelvis foretager en level 1 (L1) spiller beslutninger på baggrund af, hvilken strategi en level 0 (L0) benytter. En level 2 beslutter sin strategi på baggrund af en kombination af L0' og L1's strategier. Level 2 spilleren opstiller altså en hypotese om levelfordelingen under sig selv og best-responder på denne. Eksempelvis ville en forestilling om, at der var 100 pct. L0'ere og 0 pct. L1'ere gøre, at L2'eren benyttede præcis den samme strategi som L1'eren. Modsat ville en forestilling om 0 pct. L0'ere og 100 pct. L1'ere resultere i, at L2'eren udelukkende best-responder på levelet under sig selv, og modellen bliver da identisk med Level-K modellen. Antager hun dog en fordeling med 50 pct. L0'ere og 50 pct. L1'ere, kan hendes best-response være en helt tredje strategi. Dette fortsætter i en (teoretisk) evig følge for alle k levels.

Spillere af level k antager, at deres modstandere er fordelt jævnført en normaliseret fordeling fra niveau 0 til niveau *k-1*. Herunder ligger en antagelse om, at spillere af niveau $k$ præcist forudsiger fordelingen af de kognitive niveauer under dem selv,[26] men ikke tilegner sandsynlighed til, at andre spillere kan være af højere eller samme niveau som dem selv ($h > k$). Modellen beskrives formelt således:

Vi angiver jf. afsnit 6.1 spiller $i$'s strategi $j$ som $s_i^j$ og antager, at spiller $i$ har $m_i$ strategier. Det antages, at spillere af niveau 0 spiller ustrategisk, og dermed spiller uniformt tilfældigt med sandsynlighederne $p_0(s_i^j) = 1/m_i$ for alle j. Modspilleren -$i$'s strategi angives $s_{-i}^j$. Spiller $i$'s payoff er dermed $u_i(s_i^j, s_{-i}^j)$. Det bemærkes, som beskrevet i afsnit 3.2, at fordelingen af kognitive niveauer hos modspillerne angiver en sandsynlighed for, at modspilleren tilhører det pågældende kognitive niveau. Dette er ækvivalent til at antage, at spiller -$i$ har en vis sandsynlighed for at tilhøre et vilkårligt kognitivt niveau.

Formelt beskrives CH-modellen ved, at vi angiver spiller $i$'s forventede fordeling af andre

[26] Vi anvender både en poissonfordeling, som der anvendes i lignende litteratur, og dernæst udforsker vi en betafordeling.

spillere ved $g_k(h) \geq 0$. Da spiller $i$ forventer, at spiller -$i$ er af niveau $\mathbf{h} = (0, 1, 2, ..., h)$ med sandsynlighederne $g_k(h) = (p_0, p_1, p_2, ..., p_h)$, vil det forventede payoff for spiller $i$ af level k ved at spille strategien $\bar{j}$ være:

$$E_k(u_i(s_i^{\bar{j}}, s_{-i}^j)) = \sum_{j=1}^{m_{-i}} u_i(s_i^{\bar{j}}, s_{-i}^j)[\sum_{h=0}^{k-1} g_k(h) p_h(s_{-i}^j)] \tag{14}$$

Vi antager, at spiller $i$ best-responder i rene strategier, med sandsynligheden $p_k(s_i^{\bar{j}}) = 1$, hvis:

$$s_i^j = \max_{s_i^j} E_k(u_i(s_i^j, s_{-i}^j)) \tag{15}$$

og $p_k(s_i^j) = 0$ for resterende strategier. Hvis der er flere ækvivalente payoffs, så mikser spiller $i$ (af level k $> 0$) uniformt blandt strategierne for de maksimale payoffs.[27]

CH-modellen er principielt ens med Level-K modellen for $k \leq 1$, men kan afvige ved højere $k$. Fordelingen af kognitive niveauer betragtes ved $g_k(h)$. Det medfører heraf, at vores CH- og K-model er ens for alle k, *hvis* at level k's antagelse om fordelingen af populationen er, at der kun findes spillere af level k-1. Til at modellere spillernes forestilling om levelfordelingen $g_k(h)$ under sig selv benytter vi to forskellige fordelinger i denne afhandling - en poissonfordeling og en betafordeling, som vil blive gennemgået i hhv. afsnit 6.2.2 og 6.2.3 nedenfor.

### 6.2.2 Poissonfordeling

Med afsæt i Camerer et al. [2004] anvender vi en poissonfordeling til estimering af CH-modellens levelfordeling. Da samtlige kognitive niveauer er heltal er det oplagt at anvende en diskret fordeling. Endvidere er det nærliggende at antage, at som $k$ stiger, så øges kravene til agentens kognitive kapacitet, da hvert niveau $k$ skal iterere over alle lavere niveauer *0,1,2...,k-1*. Dette betyder, at agenten vil være tilbøjelig til at vedlægge en relativt simpel fordeling henover porteføljen af de lavere niveauer. Fragiadakis og Troyan [2017] finder, på trods af simpliciteten ved en *naiv* uniform fordeling, at poissonfordelingen er bedre til at beskrive den faktiske fordeling, som agenterne opstiller, og har en generel tendens

[27] Vi opstiller ligeledes en model, hvor at der mikses mellem strategier, hvis payoff er tæt på hinanden. Disse fremvises i afsnit 8.

til at matche virkelighedens data relativt godt. Da en poissonfordeling beskrives med en enkelt parameter $\tau$, der angiver middelværdien og variansen, er denne oplagt at anvende, da poissonfordelingen begrænser agentens mentale omkostninger.

Poissonfordelingen beskrives ved:

$$f(k) = \frac{e^{-\tau}\tau^k}{k!} \tag{16}$$

Hvor vi benytter en normaliseret fordeling:

$$g_k(h) = \frac{f(h)}{\sum_{l=0}^{k-1} f(l)}, \forall\ h\ <\ k \tag{17}$$

Fordelingen normaliseres for at sikre, at en spiller af level k betragter samtlige underliggende niveauer som hele populationen. Poissonfordelingen ligger høj vægt på niveauer, der er tæt på $k$, hvis $\tau$ er høj, og fordelingen forudsiger, at der er mange ustrategiske spillere, hvis $\tau$ er lav. Denne egenskab medfører, at hvis en spiller af niveau $h$ spiller en strategi, der medfører et PSNE, vil alle andre spillere af niveau $k > h$ spille den samme strategi, da best-response på denne PSNE-strategi må være den samme PSNE-strategi.[28] Dermed vil CH-modellen konvergere til enhver Nash-ligevægt, der kan opnås ved IESDS (Camerer et al. [2004]). Trods disse fordele ved poissonfordelingen er den dog afgrænset af den enkelte parameter, der påvirker formen.

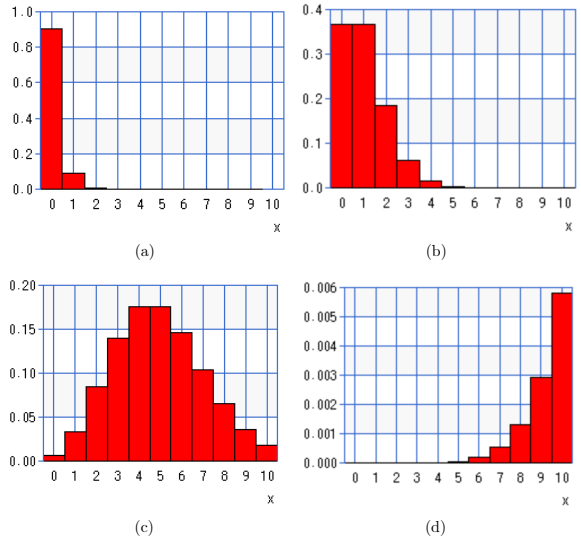
Som nævnt vil der være mange ustrategiske individer i fordelingen, hvis $\tau$ er lav, og hvis $\tau$ er høj, vil der være en overvejende vægt på kognitive niveauer lige under spillerens eget niveau. Dette medfører en begrænsning på fordelingen, da den koncentreres omkring $\tau$-værdien. Det er således ikke muligt for poissonfordelingen at estimere en højere andel L3'ere end L2'ere, hvis der samtidigt er flere L1'ere end L2'ere. I mere komplekse spil kan det måske være en fordel at kunne afbillede en mere fleksibel fordeling, der har mulighed for at skrue på flere parametre over de kognitive niveauer, hvorfor vi udvider CH-modellen med en betafordeling, som beskrives i afsnit 6.2.3.

I figur 6.1 ses poissonfordelingens sandsynlighedsmassefunktion for fire forskellige vær-

[28] Dette er relevant for mere simple spil, hvor der eksisterer et PSNE.

dier af $\tau$. Det ses, at højere værdier skubber fordelingskurven til højre. Dette medvirker, at hvis vi i vores model itererer over få levels, men benytter en høj $\tau$-værdi, vil fordelingen blive normaliseret på en ufordelagtig måde. F.eks. med $\tau = 5$ for en L5'er ville der blive lagt høj vægt på hhv. level 3 og 4, og de høje sandsynligheder på hhv. 5 og 6 ville blive negliceret. I et sådan tilfælde bør modellen normalisere hele fordelingen og skubbe den til venstre, således at der i stedet bliver lagt relativt mest vægt på de levels i midten af ens kognitive overvejelser.[29] Hvis man ikke gjorde dette, ville man potentielt kunne få antagelser af et ekstremt højt $\tau$, der på grund af den manglende sandsynlighedsmasse ville tillægge uniforme sandsynligheder på alle levels, man itererer over. Da estimatet for $\tau$ kan tolkes som en proxy for, hvor sofistikeret en population er, vil det kun give mening, hvis $\tau$ ikke er for høj relativt til det antal levels, agenten itererer over. Dette bør ikke være et problem, eftersom $\tau$ oftest er under 1,5 i litteraturen (Camerer et al. [2004]), hvilket vi ligeledes finder i afsnit 9.

**Figur 6.1:** PDF for poissonfordelingen med $k = 10$: (a) hvor $\tau = 0.1$; (b) hvor $\tau = 1$; (c) hvor $\tau = 5$; og (d) hvor $\tau = 20$.




### 6.2.3 Betafordeling

Betafordelingen er en kontinuer sandsynlighedsfordeling, der spænder fra 0 til 1 og beskrives ved de to parametre: $\alpha$ og $\beta$, som definerer formen på fordelingen. Betafordelingen kan blive interessant at anvende til vores data, da der er mulighed for at påvirke fordelingen i begge ender af de kognitive niveauer. Poissonfordelingen er begrænset af, at den kun har

[29] Dette ville effektivt være det samme som at vælge en lavere $\tau$-værdi.

én parameter $\tau$, som angiver vægten af niveauerne. Betafordelingen tillader os at antage, at der er en overvægt af ustrategiske spillere, samt spillere der gennemgår mange kognitive iterationer for at spille optimalt,[30] mens den ligger mindre vægt på spillere derimellem. Den kumulative betafordeling er givet ved:

$$F_\beta(t) = \frac{\int_0^x t^{\alpha-1}(1-t)^{\beta-1}dt}{B(\alpha,\beta)}, \qquad 0 \leq t \leq 1;\ \alpha,\beta > 0 \tag{18}$$

hvor $B(\alpha,\beta)$ er den inkomplette betafunktion,[31] t er observationerne, mens $\alpha$ og $\beta$ er parametre, der styrer formen på fordelingen. Da betafordelingen spænder over intervallet [0,1], hvilket ikke kan beskrive et diskret antal levels $\geq 1$, normaliseres ligning (18) til en diskret fordeling af individer op til level $h$:[32]

$$g_k(h) = \frac{F_\beta(\frac{h+1}{k}) - F_\beta(\frac{h}{k})}{\sum_{l=0}^{k-1} F_\beta(\frac{l+1}{k}) - F_\beta(\frac{l}{k})}, \qquad \forall h\ <\ k \tag{19}$$

Ligning (19) beskriver en betafordeling, der er normaliseret, således at fordelingen af levels splittes op i intervallerne $\frac{1}{k}$, hvor $k$ er antallet af levels. Vi har således, at sandsynligheden for at være en spiller af level $h$, er den kumulative betafordeling i det interval, som spilleren tilhører.

I figur 6.2 ses PDF'en for betafordelingen, og hvordan hhv. en Level 2, Level 3, Level 4 og en Level 5 danner sin overbevisning omkring andelen af levels under hende. Der er taget udgangspunkt i et eksempel hvor $\alpha = \beta = 0.5$, men intuitivt tages integralet under funktionen og opdeles ligeligt for alle levels under en selv.

En begrænsning ved betafordelingen er, at den er relativt kompleks og ikke kan estimeres 'on the fly' af et individ, der er i gang med at udvælge en strategi i et spil. Spillerne kan dog have en fornemmelse af fordelingen af levels under dem selv, uafhængigt om de eksplicit tager udgangspunkt i en statistisk fordeling, og betafordelingen kan måske modellere denne bedre end poissonfordelingen. Vi inkluderer dermed betafordelingen, da den

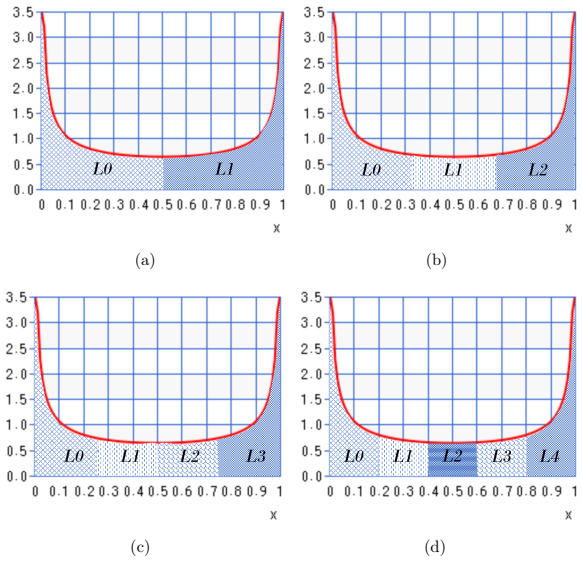
[30] Hvis $\alpha, \beta < 1$ er fordelingen u-formet og ligger højere vægt på fordelingens ender.

[31] Den inkomplette betafunktion er givet ved $B(z;\alpha,\beta) = \int_0^z t^{\alpha-1}(1-t)^{\beta-1}dt$. Den inkomplette betafunktion er lig med den normale betafunktion, når $z = 1$.

[32] En intuitiv måde at se denne normaliserede fordeling på er, at i et eksempel med 5 levels omfatter level 0 $\frac{0+1}{5} - \frac{0}{5} = 0.2 - 0 = 20$ pct. af fordelingen. Spillere af level 1 omfatter de næste $\frac{1+1}{5} - \frac{1}{5} = 0.4 - 0.2 = 20$ pct. af fordelingen.

har potentiale til at estimere den sande fordeling, som agenterne implicit danner, mere retvisende.

**Figur 6.2:** PDF for forskellige levels i betafordelingen visualiseret for $\alpha$ og $\beta = 0.5$: (a) viser hvordan en Level 2 anser fordelingen af levels under hende; (b) hvordan en Level 3 gør; (c) hvordan en Level 4 gør; og (d) hvordan en Level 5 gør.




Efter at have opstillet teorien bag vores modeller vil afsnit 7 nedenfor redegøre for de statistiske værktøjer, der anvendes i analysen, samt deres rolle i estimationen af de kognitive niveauers fordeling.

# 7 Statistisk Metode

I dette afsnit vil vi gennemgå de statistiske værktøjer, der anvendes til at estimere fordelingen af populationens kognitive niveauer, samt de redskaber der anvendes til at vurdere, hvor godt modellerne passer på datasættet.

## 7.1 Minimum Distance Estimator

Til estimering af vores CH-model anvendes en minimum distance estimator (MDE). Vi vil estimere de poisson- og betafordelingsparametre, altså hhv. $\tau$ samt $\alpha$ og $\beta$, der sikrer, at fordelingerne forklarer data bedst muligt. Som navnet angiver, estimerer MDE de parametre, der sikrer os den mindste afstand mellem to vektorer.

Minimum distance estimatoren blev fremlagt af Kiefer og Wolfowitz [1952] og Wolfowitz

[1954], hvor de fremlagde, at denne type estimator viste stærk konsistens og robusthed for diverse parametriske modeller. Minimum distance estimatoren er et generelt tilfælde af Least Square og minimum $\chi^2$-estimatorerne (Nocedal [1999]). Drossos og Philippou [1980] definerer Minimum distance estimatoren ved:

Lad $X_1, X_2, X_3, ..., X_n$ være $i.i.d$[33] samples fra en fordelingsfunktion $F(x;\theta)$, hvor $\theta$ er en ukendt parameter tilhørende $\Theta \subseteq \mathbb{R}_{\geq 1}$.
Lad $F_n(x)$ være den empiriske fordeling baseret på vores observationer $X_1, X_2, X_3, ..., X_n$ og lad $\hat{\theta}$ være en estimator for $\theta$, så er $F(x;\hat{\theta})$ en estimator for $F(x;\theta)$. Hvis der eksisterer et estimat $\hat{\theta}$ i samplet ($\hat{\theta} = \hat{\theta}(X_1, X_2, X_3, ..., X_n)$) således at

$$d[F(x;\hat{\theta}), F(x)] = \min\{d[F(x;\theta), F_n(x)]; \theta \subseteq \Theta\} \tag{20}$$

kaldes $\hat{\theta}$ for minimum distance estimatet for $\theta$. $d[F(x;\hat{\theta}), F(x)]$ angiver kriteriefunktionen for afstanden ($\geq 0$) mellem værdierne af funktionerne $F(\cdot)$. Minimum distance estimatoren vil dermed sikre, at vi finder den minimale distance ved estimatet, hvis det findes.

Til vores estimation af parametrene i minimum distance estimatoren antager vi, at vi har et konsistent estimat for spillernes strategier, $\hat{\pi}_h = \hat{P}_h(s_i^j)\ \forall\ h = 0, 1, 2, ..., k$. I vores tilfælde angiver $h$ da det kognitive level, beskrevet i afsnit 6.1 og 6.2.

Vi angiver kriteriefunktionen $d[F(x;\hat{\theta}), F(x)]$ som $Q(\theta)$:

$$Q(\theta) = d[F(x;\hat{\theta}), F(x)] = (\hat{\pi} - \pi(\theta))^T \cdot \mathbf{W}_N \cdot (\hat{\pi} - \pi(\theta)) \tag{21}$$

hvor $\hat{\pi}$ er vores observationer, og $\pi(\theta)$ er vores modeloutput. Den mest efficiente minimum distance estimator sætter $\mathbf{W}_N = \hat{V}[\hat{\pi}^{-1}]$ (Ferguson [1958]), som sikrer asymptotisk efficiens. I vores analyse anvender vi dog en ligeligt vægtet minimum distance estimator ($\mathbf{W}_N = \mathbf{1}$), som er mindre efficient end $\hat{V}[\hat{\pi}^{-1}]$, men tillader os at opretholde minimal bias i estimaterne givet et mindre antal observationer (Trivedi [2005]). Vores kriteriefunktion

[33] **I**ndependent and **I**dentically **D**istributed

angives dermed som:

$$Q(\theta) = (\hat{\pi} - \pi(\theta))^T(\hat{\pi} - \pi(\theta)) \tag{22}$$

Hvor $\hat{\theta}$ angiver parametrene, således at afstandsmålet $Q(\theta)$ minimeres netop ved $Q(\hat{\theta})$ og findes, hvis $Q(\hat{\theta})$ eksisterer. $Q(\theta)$ angives også som $l^2$-normen, som beskrives videre i afsnit 7.2. Dermed minimerer vi afstanden mellem den fordeling, der spilles i virkeligheden[34] og vores modelprædiktioner. Hermed opnår vi et estimat af parametrene for fordelingen[35] for hvert af spillernes kognitive niveauer.

MDE gør det således muligt at estimere den parameter der sikrer minimum i afstandsmålet (ligning (22)). Imidlertid vil der ikke fremgå en statistisk usikkerhed, f.eks. standardfejl, ved denne metode, da varians i vores observationstype er svær at identificere. Wolfowitz [1954] påpeger som følge af dette, at statistiske test af MDE kan være svage. Som alternativ kunne man benytte bootstrapping som i Banerjee et al. [2015] til at approksimere en varians i data. Kort fortalt benytter bootstrapping tilfældig udtrækning med tilbagelægning, hvilket da gør at man får nogle fiktive samples. Man ville så se på variansen mellem disse samples, for at få en proxy for variansen er i det datasæt de er dannet fra. Som vi diskuterer i afsnit 10 er denne metode dog svær at benytte med den type data vi har tilgængelig.

### 7.1.1 Sequential Quadratic Programming

Til estimering af minimum distance parametrene anvendes Sequential Quadratic Programming (SQP) algoritmen, der er en af de mest effektive metoder til at optimere ikke-lineære bibetingede problemer gennem iteration af kvadratiske beregninger (Nocedal [1999]). SQP minimerer en funktion af en eller flere variable og tillader bibetingelser. SQP vurderes til at være ideél, hvis funktionen er kontinuert dobbeltdifferentiabel (Kraft [1988]).

Nocedal [1999] beskriver, at et ikke-lineært programmeringsproblem opbygget på bag-

[34] Altså observationer $\hat{\pi}$.
[35] Dvs. $\tau$ samt $\alpha$ og $\beta$ for hhv. poisson og beta-CH modellerne.

grund af ligning (22) kan udtrykkes ved:

$$\min_{\theta \in \mathbb{R}^n} Q(\theta) \tag{23}$$

betinget på

$$c_i(\theta) = \pi_i(\theta) \geq 0, \qquad \forall\ i = 1, 2, ..., n \tag{24}$$

Hvor $Q(\theta)$ er kriteriefunktionen, $c_i(\theta)$ angiver bibetingelserne, og $\pi_i$ angiver spillernes sandsynligheder for at spille et deck. Der er 20 decks i datasættet, hvorfor der i vores tilfælde gælder, at vi optimerer en funktion, der afhænger af en vektor bestående af 20 andele, hvor fordelingen af andele bestemmes af $\theta$. CH-modellen løses iterativt ved progression gennem levels fra $0 \rightarrow k$, hvorefter vi beregner de gennemsnitlige deckvægte for hvert level. Derefter minimeres afstanden mellem prædiktioner og data på baggrund af $\theta$. SQP er en iterativ metode, der modellerer ovenstående problem for en iteration af $\theta_t$ gennem en løsningsmetode og løser for $\theta_{t+1}$. $\theta_{t+\epsilon}$ konvergerer til et lokalt minimum, når $\epsilon \rightarrow \infty$.

Nedenfor gennemgås et eksempel på en løsningsproces med én bibetingelse ($c_i(\theta) = c(\theta)$) for at belyse den iterative process. Metoden kan udvides til at indeholde flere restriktioner. SQP-algoritmen er bredt anvendt inden for numerisk analyse, og den er udviklet og optimeret gennem Python-biblioteket SciPy [2020], som vi anvender til vores optimering. Vi har følgende antagelser ved løsning af ikke-lineære problemer med ulighedsbetingelser (Boggs og Tolle [2000]):

**A1:** Førsteordensbetingelserne holder, således at der eksisterer optimale lagrange-multiplikatorer $\lambda^* \geq 0$, således $\nabla_\theta \mathcal{L}(\theta^*, \lambda^*) = 0$.
**A2:** Hesse-matricen af $\mathcal{L}(\theta, \lambda)$ er positiv definit og dermed invertibel.

Det kræves derfor, at vores kriteriefunktion er differentiabel, samt at et optimum eksisterer og kan løses. Denne antagelse opretholdes af vores valg af $l^2$-normen som kriteriefunktion, der sikrer, at $\mathcal{L}(\theta, \lambda)$ er dobbeltdifferentiabel som følge af det kvadrerede afstandsmål. **A2** kræver, at funktionen er dobbeltdifferentiabel, hvilket ligeledes sikres af vores valg af af-

standsmål. Hesse-matricen er endvidere invertibel, da ustrategiske individer (L0) antages at spille uniformt tilfældigt over samtlige strategier. Således tillægges der altid en positiv sandsynlighed til de 20 strategier i datasættet,[36] der da sikrer, at matricen er positiv definit, hvorfor **A2** er opfyldt. Antagelsen brydes dog, hvis SQP-algoritmen estimerer en fordeling af individer, hvor der tillægges en vægt på 0 til spillere af level 0. Studier (Crawford et al. [2013]) viser dog, at en overvejende andel af individer, på tværs af forskellige typer af spil, agerer ustrategisk, hvorfor vi ikke forventer, at **A2** er et problem for estimationen af parametrene til modellen. Vi finder i resultatafsnit 9, at det ikke er et problem.

Lagrangefunktionen for problemet er $\mathcal{L}(\theta, \lambda) = Q(\theta) - \lambda(c(\theta))$, hvor $\lambda$ er lagrange-multiplikatoren. $A(\theta)^T = [\nabla_\theta c_1(\theta), \nabla_\theta c_2(\theta), ..., \nabla_\theta c_n(\theta)]$ er de transponerede gradienter af lagrange-multiplikatoren.

Vi noterer de nødvendige KKT-førsteordensbetingelser[37] for lagrangefunktionen i et lineært system (Kuhn og Tucker [1951]):

$$F(\theta, \lambda) = \begin{bmatrix} \nabla_\theta Q(\theta) - A(\theta)^T \lambda \\ c(\theta) \end{bmatrix} = 0, \quad \text{for et vilkårligt } \lambda^* \geq 0$$

Jacobian-matricen i forhold til $\theta$ og $\lambda$ er

$$F'(\theta, \lambda) = \begin{bmatrix} \nabla^2_{\theta\theta} \mathcal{L}(\theta, \lambda) & -A(\theta)^T \\ A(\theta) & 0 \end{bmatrix}$$

Næste trin angives ved

$$\begin{bmatrix} \theta_{t+1} \\ \lambda_{t+1} \end{bmatrix} = \begin{bmatrix} \theta_t \\ \lambda_t \end{bmatrix} + \begin{bmatrix} p_t \\ p_\lambda \end{bmatrix}$$

hvor $p_t$ og $p_\lambda$ løser ligningssystemet:

$$\begin{bmatrix} \nabla^2_{\theta\theta} \mathcal{L}(\theta, \lambda) & -A_t(\theta)^T \\ A_t(\theta) & 0 \end{bmatrix} \begin{bmatrix} p_t \\ p_\lambda \end{bmatrix} = \begin{bmatrix} \nabla Q_t + A_t(\theta)^T \lambda_t \\ -c_t(\theta) \end{bmatrix}$$

[36] For enhver andel af L0 $> 0$.

[37] Karush-Kuhn-Tucker (KKT) betingelserne er et sæt betingelser, der tillader os at arbejde med ulighedsbetingelser i ikke-lineær programmering. De specifikke krav til betingelserne fremgår af Kuhn og Tucker [1951].

Iterationen klassificeres som en løsning til det ikke-lineære problem, hvis en konvergenstest bestås (Boggs og Tolle [2000]). Ellers fortsættes de iterative trin, indtil konvergens er opnået ($\lambda_t \to \lambda^*$, $\theta_t \to \theta^*$).

Den iterative proces for SQP-algoritmen fremgår simplificeret i algoritme 2 nedenfor:

**Algoritme 2:** Optimering ved SQP-algoritme

Vælg to initiale værdier $(\theta_0, \lambda_0)$, sæt $t = 0$;

**while** $\lambda_t \neq \lambda^*$, $\theta_t \neq \theta^*$ **do**

Evaluer

$$Q_t(\theta),\ \nabla Q_t(\theta),\ \nabla^2_{\theta\theta}\mathcal{L}_t(\theta, \lambda) \text{ og } A_t(\theta)$$

Løs problemet,

$$\begin{bmatrix} \nabla^2_{\theta\theta}\mathcal{L}_t(\theta, \lambda) & -A_t(\theta)^T \\ A_t(\theta) & 0 \end{bmatrix} \begin{bmatrix} p_t \\ p_\lambda \end{bmatrix} = \begin{bmatrix} \nabla Q_t(\theta) + A_t(\theta)^T \lambda_t \\ -c_t(\theta) \end{bmatrix},$$

således at $p_t$ og $p_\lambda$ kan løse,

$$\begin{bmatrix} \theta_{t+1} \\ \lambda_{t+1} \end{bmatrix} = \begin{bmatrix} \theta_t \\ \lambda_t \end{bmatrix} + \begin{bmatrix} p_t \\ p_\lambda \end{bmatrix}$$

**next**

**end**

## 7.2 $l^2$-Norm

Minimum distance estimatoren beskrevet i afsnit 7.1 ovenfor finder den parameter, der sikrer et lokalt minimum for afstandsmålet mellem modellens prædiktioner og observeret data. Essentielt finder ligning (21) netop dette, hvor $Q(\theta)$ er afstanden mellem vores observationer og estimater. Som nævnt benytter vi vægten $\mathbf{W}_N = \mathbf{1}$, som sætter ligelig vægt på alle observationer. Inden SQP-algoritmen anvendes til at estimere parametrene, beregnes summen af den kvadrerede afstand mellem de to vektorer for hver værdi af $\theta$, som ses i ligning (23). Essentielt svarer dette til et specialtilfælde af MDE kaldet $l^2$-normen. Hvis denne afstand er lig 0, betyder det, at modelprædiktionerne ligger oveni det observerede data, og modellen er derved komplet.[38] Desto større afstand, desto dårligere et fit har modellen på data. Det er klart, givet vægtningen $\mathbf{W}_N = \mathbf{1}$, at enkelte outliers fra modellens

[38] I hvert fald for den givne observation.

prædiktioner vil trække denne afstand relativt mere op, da afstanden kvadreres. Dette betyder endvidere også, at MDE i høj grad finder den parameterestimation, der giver den mindste afstand mellem data og outliers, hvorfor disse får en abnormal høj betydning for estimatorens parameterestimation. En måde at tage højde for dette ville være at antage en anden vægtning af $\mathbf{W}_N$.

$l^2$-normen angiver derfor et afstandsmål, som vi kan benytte til at sammenligne de modeller, der indgår i analysen. I denne afhandling vil en lavere $l^2$-norm betyde, at modellen passer bedre på data. Dette mål gør derfor, at vi kan evaluere, hvor godt vores modeller beskriver data. Imidlertid kan denne afstand være svær at kvantificere, hvorfor vi udover denne norm også vil sammenligne $R^2$-værdier, som vi kommer videre ind på i afsnit 7.3.

## 7.3 Determinationskoefficienten $R^2$

$R^2$-koefficienten angiver et mål for forklaringsgraden af vores modelestimater. Hvor det i $l^2$-normen kan være svært at forholde sig til, hvor meget bedre en værdi på 66 er end 114, kan vi benytte $R^2$-værdier til nemmere at danne os et overblik over, hvilken model der passer bedst på data, da $R^2$-værdien er et standardiseret mål for modellens afvigelse.

$R^2$ er et statistisk mål, der angiver, hvor stor en andel af variansen af en afhængig variabel der kan forklares af uafhængige variable. Til at beregne $R^2$-koefficienten benyttes *total sum of squares*, der er proportionel med den samlede varians i data:

$$SS_{tot} = \sum_{i}^{n} (\hat{\pi}_i - \bar{\pi}_i)^2 \tag{25}$$

hvor n er antal deck, $\hat{\pi}_i$ er observationerne, og $\bar{\pi}_i$ er middelværdien for observationerne.[39] Ligning (25) beskriver da den totale kvadrerede afvigelse fra middelværdien. Hernæst findes *sum of squares of residuals*:

$$SS_{res} = \sum_{i}^{n} (\hat{\pi}_i - \pi_i)^2 \tag{26}$$

[39]Middelværdien vil være 1/antal mulige deck i vores modeller.

hvor $\pi_i$ er vores modelprædiktioner. Ligning (26) beskriver da den totale kvadrerede afvigelse imellem modelprædiktionerne og vores observerede data. $R^2$ beskriver således den relative forskel mellem $SS_{tot}$ og $SS_{res}$ og er defineret ved:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{27}$$

hvor $SS_{res} = 0$ medfører $R^2 = 1$, hvilket svarer til, at modelprædiktionen passer perfekt på data, da $\hat{\pi}_i = \pi_i$. En model, hvis prædiktioner er lig middelværdien, $\hat{\pi}_i = \bar{\pi}_i$, resulterer i $R^2 = 0$. Dette svarer til en model udelukkende af Level 0'ere jf. afsnit 6.1.

Det bemærkes, at enkelte modelprædiktioner,[40] der ligger langt fra data, vil forstørre $SS_{res}$ abnormalt meget, og eventuelle outliers fra modellerne vil da forårsage en *ekstremt* lav $R^2$-værdi. Et negativt $R^2$ er da muligt, hvilket teoretisk ville svare til, at modellen passer dårligere på data end en model, der udelukkende består af tilfældige (L0) individer.

Afsnit 8 nedenfor vil gennemgå opsætningen af vores kognitive hierarkimodeller samt fremlægge en generel løsning af best-responses for individer af et vilkårligt level.

# 8 Setup

I dette afsnit beskrives vores specifikke setup af modellerne, som er beskrevet i afsnit 6. Vi gennemgår løsningsprocessen for modellerne, samt hvilke antagelser og udgangspunkter der tages ved løsning af modellerne.

Afsnit 4 beskriver, at vi kan betragte Hearthstone som en spilteoretisk version af et avanceret slags sten-saks-papir med 20 mulige strategier, der hver især har en given winrate mod hinanden. Tabel 8.1 nedenfor opstiller netop disse winrates, hvor vi beregner Nash-ligevægten jf. minimax-metoden, som blev forelagt i afsnit 3.3.[41]

[40] f.eks. $\pi_1$.
[41] I appendix C kan ses pythonkoden 'solvemixednash' der løser dette.

**Tabel 8.1:** Winrate Matrix (Baseline)

| | Aggro DH | Spell Druid | Dragon hunter | Face hunter | Highlander Hunter | Highlander mage | Spell mage | Murloc Paladin | Pure Paladin | Galakrond Priest | Highlander priest | Galakrond rogue | Highlander rogue | Galakrond shaman | Highlander shaman | Totem shaman | Quest warlock | Zoo warlock | Control warrior | Enrage Warrior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aggro Demon Hunter | **50.00** | 57.54 | 54.59 | 37.72 | 52.90 | 40.30 | 46.23 | 52.97 | 42.37 | 46.19 | 46.31 | 45.43 | 53.1 | 63.2 | 49.65 | 69.47 | 43.60 | 56.43 | 41.95 | 34.98 |
| Spell Druid | 42.46 | **50.00** | 31.35 | 50.03 | 47.21 | 50.69 | 49.48 | 26.99 | 38.42 | 50.53 | 52.42 | 55.61 | 56.70 | 42.16 | 61.14 | 30.15 | 50.99 | 41.80 | 49.21 | 58.52 |
| Dragon hunter | 45.41 | 68.65 | **50.00** | 45.83 | 51.74 | 61.07 | 70.86 | 25.43 | 30.42 | 56.98 | 54.09 | 50.78 | 49.57 | 63.84 | 53.65 | 46.97 | 63.38 | 45.67 | 50.55 | 48.48 |
| Face hunter | 62.28 | 49.97 | 54.17 | **50.00** | 55.00 | 63.01 | 73.64 | 33.36 | 26.36 | 38.86 | 39.58 | 52.31 | 60.7 | 67.74 | 58.59 | 35.62 | 60.03 | 53.05 | 53.79 | 51.76 |
| Highlander Hunter | 47.1 | 52.79 | 48.26 | 45.00 | **50.00** | 61.35 | 60.3 | 47.06 | 48.16 | 63.40 | 62.79 | 53.18 | 58.59 | 61.85 | 67.21 | 55.24 | 52.77 | 48.62 | 54.49 | 50.58 |
| Highlander mage | 59.7 | 49.31 | 38.93 | 36.99 | 38.65 | **50.00** | 49.40 | 58.67 | 61.39 | 54.67 | 56.84 | 54.68 | 57.02 | 52.14 | 65.72 | 50.08 | 51.52 | 60.77 | 46.84 | 60.12 |
| Spell mage | 53.77 | 50.52 | 29.14 | 26.36 | 39.70 | 50.6 | **50.00** | 41.72 | 54.25 | 49.64 | 49.14 | 53.19 | 60.64 | 45.80 | 63.74 | 41.16 | 52.48 | 57.79 | 44.36 | 53.37 |
| Murloc Paladin | 47.03 | 73.01 | 74.57 | 66.64 | 52.94 | 41.33 | 58.28 | **50.00** | 50.72 | 55.17 | 57.32 | 50.77 | 53.33 | 67.84 | 53.66 | 57.16 | 40.48 | 56.52 | 41.97 | 36.31 |
| Pure Paladin | 57.63 | 61.58 | 69.58 | 73.64 | 51.84 | 38.61 | 45.75 | 49.28 | **50.00** | 54.58 | 45.78 | 36.73 | 51.19 | 55.25 | 61.31 | 50.63 | 49.07 | 46.72 | 44.85 | 61.74 |
| Galakrond Priest | 53.81 | 49.47 | 43.02 | 61.14 | 36.6 | 45.33 | 50.36 | 44.83 | 45.42 | **50.00** | 56.03 | 48.13 | 49.03 | 43.67 | 57.29 | 34.28 | 38.47 | 46.47 | 63.04 | 71.17 |
| Highlander priest | 53.69 | 47.58 | 45.91 | 60.42 | 37.21 | 43.16 | 50.86 | 42.68 | 54.22 | 43.97 | **50.00** | 41.24 | 45.34 | 45.45 | 55.79 | 37.78 | 37.36 | 44.39 | 64.89 | 73.16 |
| Galakrond rogue | 54.57 | 44.39 | 49.22 | 47.69 | 46.82 | 45.32 | 46.81 | 49.23 | 63.27 | 51.87 | 58.76 | **50.00** | 56.11 | 57.70 | 60.85 | 52.75 | 45.69 | 49.32 | 44.42 | 46.95 |
| Highlander rogue | 46.90 | 43.30 | 50.43 | 39.30 | 41.41 | 42.98 | 39.36 | 46.67 | 48.81 | 50.97 | 54.66 | 43.89 | **50.00** | 43.75 | 54.75 | 52.79 | 43.11 | 44.79 | 44.51 | 42.39 |
| Galakrond shaman | 36.8 | 57.84 | 36.16 | 32.26 | 38.15 | 47.86 | 54.2 | 32.16 | 44.75 | 56.33 | 54.55 | 42.30 | 56.25 | **50.00** | 55.66 | 36.18 | 41.41 | 40.22 | 65.59 | 58.69 |
| Highlander shaman | 50.35 | 38.86 | 46.35 | 41.41 | 32.79 | 34.28 | 36.26 | 46.34 | 38.69 | 42.71 | 44.21 | 39.15 | 45.25 | 44.34 | **50.00** | 37.86 | 34.22 | 44.34 | 35.94 | 51.55 |
| Totem shaman | 30.53 | 69.85 | 53.03 | 64.38 | 44.76 | 49.92 | 58.84 | 42.84 | 49.37 | 65.72 | 62.22 | 47.25 | 47.21 | 63.82 | 62.14 | **50.00** | 59.18 | 44.17 | 43.57 | 35.48 |
| Quest warlock | 56.40 | 49.01 | 36.62 | 39.97 | 47.23 | 48.48 | 47.52 | 59.52 | 50.93 | 61.53 | 62.64 | 54.31 | 56.89 | 58.59 | 65.78 | 40.82 | **50.00** | 55.02 | 27.89 | 53.62 |
| Zoo warlock | 43.57 | 58.20 | 54.33 | 46.95 | 51.38 | 39.23 | 42.21 | 43.48 | 53.28 | 53.53 | 55.61 | 50.68 | 55.21 | 59.78 | 55.66 | 55.83 | 44.98 | **50.00** | 39.06 | 45.12 |
| Control warrior | 58.05 | 50.79 | 49.45 | 46.21 | 45.51 | 53.16 | 55.64 | 58.03 | 55.15 | 36.96 | 35.11 | 55.58 | 55.49 | 34.41 | 64.06 | 56.43 | 72.11 | 60.94 | **50.00** | 35.73 |
| Enrage Warrior | 65.02 | 41.48 | 51.52 | 48.24 | 49.42 | 39.88 | 46.63 | 63.69 | 38.26 | 28.83 | 26.84 | 53.05 | 57.61 | 41.31 | 48.45 | 64.52 | 46.38 | 54.88 | 64.27 | **50.00** |

Note: Bemærk at en større og mere letlæselig version af tabellen kan ses i Appendix B.1.

Foruden disse winrates, der benyttes til at beregne optimale strategier, har vi behov for at observere, hvordan spillerne rent faktisk spiller. De observerede antal kampe, som hver kombination af deck har lavet mod hinanden, kan ses i tabel 8.2. Bemærk, at man for at finde andelen af de totale kampe, som ét deck står for, ikke blot kan summere den pågældende række. Dette da kampene, hvori man spiller mod det samme deck som en selv, ikke giver udtryk for, at denne strategi er blevet benyttet af begge parter. For at finde antallet af totale kampe for et deck skal man da tælle de antal kampe hvor man møder det samme deck, som man selv benytter dobbelt. Eksempelvis findes antallet af kampe for "Aggro Demon Hunter" ved:

$$\begin{aligned}\sum 2 * 608 + 527 + 381 + 184 + 1376 + 554 + 453 + 503 + 319 + 719 + 336 + 618 + 260 \\ +402 + 158 + 258 + 389 + 245 + 262 + 246 \\ = 9.406\end{aligned}$$

Dette gøres for alle deck, hvor det totale antal kampe findes til 152.454, og Aggro Demon Hunter udgør da 6,2 pct. af disse.

**Tabel 8.2:** Frekvens Matrix (Baseline)

| | Aggro DH | Spell Druid | Dragon hunter | Face hunter | Highlander Hunter | Highlander mage | Spell mage | Murloc Paladin | Pure Paladin | Galakrond Priest | Highlander priest | Galakrond rogue | Highlander rogue | Galakrond shaman | Highlander shaman | Totem shaman | Quest warlock | Zoo warlock | Control warrior | Enrage Warrior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aggro DH | **608** | 527 | 381 | 184 | 1376 | 554 | 453 | 503 | 319 | 719 | 336 | 618 | 260 | 402 | 158 | 258 | 389 | 245 | 262 | 246 |
| Spell Druid | 527 | **560** | 403 | 213 | 1113 | 622 | 482 | 467 | 356 | 785 | 329 | 572 | 263 | 425 | 208 | 245 | 400 | 165 | 214 | 229 |
| Dragon hunter | 381 | 403 | **304** | 154 | 870 | 406 | 362 | 374 | 187 | 482 | 194 | 402 | 203 | 317 | 105 | 198 | 252 | 157 | 170 | 170 |
| Face hunter | 184 | 213 | 154 | **92** | 413 | 233 | 202 | 200 | 114 | 245 | 132 | 234 | 89 | 155 | 56 | 92 | 128 | 81 | 79 | 64 |
| Highlander Hunter | 1376 | 1113 | 870 | 413 | **3652** | 1353 | 1084 | 1282 | 791 | 1796 | 756 | 1536 | 686 | 879 | 409 | 553 | 1056 | 513 | 644 | 567 |
| Highlander mage | 554 | 622 | 406 | 233 | 1353 | **774** | 580 | 548 | 417 | 749 | 575 | 585 | 395 | 424 | 261 | 268 | 435 | 268 | 250 | 231 |
| Spell mage | 453 | 482 | 362 | 202 | 1084 | 580 | **486** | 450 | 313 | 623 | 432 | 497 | 251 | 352 | 206 | 243 | 329 | 197 | 173 | 162 |
| Murloc Paladin | 503 | 467 | 374 | 200 | 1282 | 548 | 450 | **598** | 331 | 696 | 298 | 593 | 228 | 412 | 171 | 238 | 343 | 236 | 214 | 168 |
| Pure Paladin | 319 | 356 | 187 | 114 | 791 | 417 | 313 | 331 | **214** | 427 | 251 | 336 | 168 | 234 | 140 | 165 | 244 | 130 | 153 | 116 |
| Galakrond Priest | 719 | 785 | 482 | 245 | 1796 | 749 | 623 | 696 | 427 | **756** | 426 | 711 | 403 | 537 | 256 | 309 | 471 | 283 | 240 | 291 |
| Highlander priest | 336 | 329 | 194 | 132 | 756 | 575 | 432 | 298 | 251 | 426 | **424** | 395 | 264 | 255 | 184 | 140 | 234 | 150 | 148 | 95 |
| Galakrond rogue | 618 | 572 | 402 | 234 | 1536 | 585 | 497 | 593 | 336 | 711 | 395 | **710** | 321 | 461 | 224 | 275 | 449 | 276 | 274 | 247 |
| Highlander rogue | 260 | 263 | 201 | 89 | 686 | 395 | 251 | 228 | 168 | 403 | 264 | 321 | **170** | 180 | 108 | 108 | 212 | 140 | 161 | 137 |
| Galakrond shaman | 402 | 425 | 317 | 155 | 879 | 424 | 352 | 412 | 234 | 537 | 255 | 461 | 180 | **344** | 133 | 231 | 275 | 161 | 186 | 179 |
| Highlander shaman | 158 | 208 | 105 | 56 | 409 | 261 | 206 | 171 | 140 | 256 | 184 | 224 | 108 | 133 | **74** | 73 | 134 | 84 | 72 | 70 |
| Totem shaman | 258 | 245 | 198 | 92 | 553 | 268 | 243 | 238 | 165 | 309 | 140 | 275 | 108 | 231 | 73 | **78** | 159 | 106 | 99 | 96 |
| Quest warlock | 389 | 400 | 252 | 128 | 1056 | 435 | 329 | 343 | 244 | 471 | 234 | 449 | 212 | 275 | 134 | 159 | **298** | 133 | 190 | 154 |
| Zoo warlock | 245 | 165 | 157 | 81 | 513 | 268 | 197 | 236 | 130 | 283 | 150 | 276 | 140 | 161 | 84 | 106 | 133 | **82** | 97 | 82 |
| Control warrior | 262 | 214 | 170 | 79 | 644 | 250 | 173 | 214 | 153 | 240 | 148 | 274 | 161 | 186 | 72 | 99 | 190 | 97 | **114** | 84 |
| Enrage Warrior | 246 | 229 | 170 | 64 | 567 | 231 | 162 | 168 | 116 | 291 | 95 | 247 | 137 | 179 | 70 | 96 | 154 | 82 | 84 | **116** |

Note: Bemærk at en større og mere letlæselig version af tabellen kan ses i Appendix B.2.

Vi husker fra afsnit 6.1, at spiller $i$'s strategi $j$ betegnes $s_i^j$ og antager, at spiller $i$ har $m_i$ strategier. Spillere af level $k$ spiller en strategi med sandsynlighederne $P_k(s_i^j) \geq 0 \; \forall \; j$. Vi betragter ligeledes modspilleren -$i$'s strategi, $s_{-i}^j$. Spiller $i$'s payoff er dermed $u_i(s_i^j, s_{-i}^j)$.

$m_i$ dækker derved over de 20 mulige strategier i tabel 8.1, som spillerne potentielt kan anvende: Aggro Demon Hunter, Spell Druid, Dragon Hunter, Face Hunter, Highlander Hunter, Highlander Mage, Spell Mage, Murloc Paladin, Pure Paladin, Galakrond Priest, Highlander Priest, Galakrond Rogue, Highlander Rogue, Galakrond Shaman, Highlander Shaman, Totem Shaman, Quest Warlock, Zoo Warlock, Control Warrior og Enrage Warrior.[42] Hvor Camerer et al. [2004] antager, at der eksisterer op til fire kognitive niveauer i populationen, antager vi, at der eksisterer op til fem. Dette antages, da modellens design ellers potentielt kunne være for indskrænkende i store spil, hvor højere levels netop vil have mulighed for at udskille sig fra de lavere levels, således at fordelingen af kognitive niveauer mere præcist kan estimeres (Costa-Gomes og Crawford [2006]).

Vi betragter et nulsumsspil med normaliserede winrates, der repræsenterer payoffs mellem $-1$ ved et garanteret nederlag og $+1$ ved en garanteret sejr. Ligning (12) viser det forventede payoff i Level-K setuppet, hvor den normaliserede winrate for spiller $i$ ved strategien $\bar{j}$ er:

$$\hat{u}_i(s_i^{\bar{j}}, s_{-i}^{j}) = \frac{u_i(s_i^{\bar{j}}, s_{-i}^{j})}{50} - 1 \tag{28}$$

Eksempelvis ville det normaliserede payoff for spiller $i$ ved at spille $s_i^4$, svarende til strategien "Face Hunter", når spiller $-i$ spiller $s_{-i}^3$, der svarer til strategien "Dragon Hunter", være:

$$\begin{aligned}\hat{u}_i(s_i^4, s_{-i}^3) &= \frac{54,17}{50} - 1 \\ &= 0,0834\end{aligned}$$

Hvorfor det forventede payoff for spiller $i$ ved netop dette matchup[43] er 0,0834 - hvilket er højere end, hvis de begge havde spillet en tilfældig strategi og da opnået et forventet payoff på 0. Dette kommer til at have en effekt på spiller $i$'s strategivalg, når hun laver antagelser om de mulige strategier, hun kan møde.

[42] Strategien navngives ved en kombination af to elementer: En class samt en taktik. Der er 10 mulige classes, hvor Hunter eller Mage er eksempler på disse. Der er forskellige taktikker, som kan være sæsonafhængigt, eksempelvis "Highlander". Kombinationen af disse to bliver da til strategien "Highlander Hunter".

[43] Et matchup er defineret som kampen mellem to specifikke deck. I dette tilfælde er matchuppet altså mellem Face Hunter og Dragon Hunter.

Med udgangspunkt i winrate-matricen i tabel 8.1 gennemgår vi analysen bag Level-K modellen, hvor vi fokuserer på det forventede payoff:

$$E[\sum_{j=0}^{m_{-i}} u_i(s_i^{\bar{j}}, s_{-i}^{j})] \tag{29}$$

Da winrate-matricen er en $20 \times 20$ matrix,[44] vil spillere af level 0 spille uniformt med sandsynlighederne $p_0(s_i^j) = \frac{1}{20}$. En spiller af level 1 vil best-responde til en uniform fordelingen af alle strategier, og spilleren vælger derfor den strategi med det højest forventede payoff ud fra en transformeret payoff-matrix, der betegnes ved:

$$\bar{u}_i(s_i^{\bar{j}}, s_{-i}^{j}) = \frac{1}{20} u_i(s_i^{\bar{j}}, s_{-i}^{j}) \; \forall \; \bar{j}, j = 1, 2, ..., 20 \tag{30}$$

hvor $\bar{u}_i(s_i^{\bar{j}}, s_{-i}^{j})$ er elementerne af den transformerede payoff-matrix. En spiller af level 1 spiller derfor $s_i^{\bar{j}}$ med sandsynligheden $p_1(s_i^{\bar{j}}) = 1$, hvis $s_i^{\bar{j}} = \max_{s_i^{\bar{j}}} E[\sum_{j=0}^{m_{-i}} \bar{u}_i(s_i^{\bar{j}}, s_{-i}^{j})]$ og $p_1(s_i^{\bar{j}}) = 0$ ellers. Dette er ækvivalent til at spille det deck med de højest gennemsnitlige payoff svarende til Highlander Hunter med et gennemsnitligt payoff på 0,089 - som er højere end ved at spille tilfældigt.[45] En spiller af level 2 vil best-responde til strategien for en spiller af level 1, hvor hun spiller den strategi, der har det højest forventede payoff mod level 1's strategi $s_i^{\bar{j}}$. For højere levels foretages samme iterative metode, hvor spillerne best-responder til den strategi, der spilles af det level, der er et trin under dem selv.

Når vi skal løse CH-modellerne, bliver det mere komplekst, da spillerne best-responder i disse på baggrund af en fordeling af de kognitive niveauer under dem. Spillere af level 0 spiller fortsat uniformt tilfældigt med sandsynlighederne $p_0(s_i^j) = \frac{1}{20}$. Endvidere er strategien for spillere af level 1 i en CH-model lig med spillere af level 1 i Level-K modellen, da fordelingen af underliggende levels (L0) er uændret. Spillere af level 1 løser derved for det højest forventede payoff i ligning (30) og spiller fortsat Highlander Hunter.

For de enkelte kognitive niveauer definerer vi en sandsynlighedsfordeling, som i denne artikel er hhv. poisson- og betafordelingen. Sandsynlighedsfordelingen beskrives ved

[44] Og vi da har, at $m_i = 20$.
[45] Og da spille det samme som level 0.

$\mathbf{Z}_k = (z_0, z_1, z_2, ..., z_{k-1})$ og angiver en normaliseret fordeling af spillere, der hhv. er af niveau $0, 1, 2, ..., k-1$. Til at beregne strategisandsynlighederne, der spilles af de enkelte kognitive niveauer, anvender vi en dummyvariabel til at returnere 1, hvis en spiller af level $k$ spiller decket og 0 ellers. Dummyvariablen beskrives for en level $k$ spiller som: $I_k^{\bar{j}}$ og er lig 1 for den optimale strategi $\bar{j}$ og lig 0 for alle ikke-optimale strategier. Selve løsningsmetoden for CH-modellen er iterativ fra level 0 til level k, hvor den iterative proces beskrives således:

Initielt foretager vi beregningerne for level 0, der betragter den *normale* winrate-matrix bestående af matrixelementerne, der betegnes $u_i(s_i^{\bar{j}}, s_{-i}^{j})$. De benytter strategisandsynlighederne $p_0(s_i^j) = I_0^j \frac{1}{20} \ \forall \ j = 1, 2, ..., 20$.[46] For spillere af level 1 vægtes denne med den normaliserede fordeling for de kognitive niveauer $\mathbf{Z}_1 = (z_0)$ hvor $z_0 = 1$. Det forventede payoff for en level 1 bliver da:

$$\bar{u}_{i,1}(s_i^{\bar{j}}, s_{-i}^{j}) = I_0^j \cdot \frac{1}{20} \cdot z_0 \cdot u_i(s_i^{\bar{j}}, s_{-i}^{j}) \ \forall \ \bar{j}, j = 1, 2, ..., 20 \tag{31}$$

Hvor $\bar{u}_{i,1}(s_i^{\bar{j}}, s_{-i}^{j})$ er elementerne af den transformerede payoff-matrix for spiller $i$, der er af level 1. Det bemærkes, at denne er ækvivalent til ligning (30), da $z_0 = 1$, og $I_0^j = 1$ for samtlige strategier. Vi vælger dog at benytte denne notation, da den beskriver grundlaget for beregningerne for de højere kognitive niveauer samt anvendes i den generelle formel for transformering af payoff-matricen, der kan ses i ligning (33).

En level 1 spiller optimerer altså ud fra ligning (31) og benytter denne til at danne sin best-response strategi. Vi husker fra Level-K modellen, at spillere af level 1 best-responder til denne transformerede payoff-matrix i rene strategier og spiller "Highlander Hunter". Vi angiver derfor dummyvariablen $I_1^j = 1$, hvis strategi $j$ = Highlander Hunter. En spiller af level 2 har en forventning om, at sandsynlighederne for at møde de forskellige decks afhænger af strategierne, som L0 og L1 spiller, såvel som hvor mange der er af hver.

[46] For en level 0 er $I_0^j = 1 \ \forall \ j$.

Den transformerede payoff-matrix, som spillere af level 2 best-responder til, fremgår af ligning (32):

$$\bar{u}_{i,2}(s_i^{\bar{j}}, s_{-i}^{j}) = z_0 \cdot \frac{I_0^j}{20} \cdot u_i(s_i^{\bar{j}}, s_{-i}^{j}) + z_1 \cdot I_1^j \cdot \bar{u}_{i,1}(s_i^{\bar{j}}, s_{-i}^{j}) \; \forall \; \bar{j}, j = 1, 2, ..., 20 \tag{32}$$

Hvor $\bar{u}_{i,2}(s_i^{\bar{j}}, s_{-i}^{j})$ angiver elementerne i den transformerede payoff-matrix for spiller $i$, der er af level 2. En spiller af level 2 best-responder til den transformerede payoff-matrix ved at spille den strategi, der sikrer det højeste forventede payoff.

På baggrund af ligning (32) kan vi opstille en generel løsning for elementerne i den transformerede payoff-matrix. Spillere af level 0 spiller *altid* den samme strategi, hvorfor udtrykket $z_0 \cdot \frac{I_0^j}{20} \cdot u_i(s_i^{\bar{j}}, s_{-i}^{j})$ er uændret uanset det kognitive niveau, der betragtes. For et givent kognitivt niveau $k$ vil en spiller best-responde til en transformeret payoff-matrix bestående af forventninger til fordelingen af de kognitive niveauer $0, 1, ...,k$-1, samt de strategier de spiller. Dermed summeres de vægtede strategier for kognitive niveauer $< k$, således at den transformerede payoff-matrix afspejler level $k$'s forventninger.

En generel løsning af elementerne i den transformede payoff-matrix for et vilkårligt level $k \geq 2$ fremgår af ligning (33):

$$\bar{u}_{i,k}(s_i^{\bar{j}}, s_{-i}^{j}) = z_0 \cdot \frac{I_0^j}{20} \cdot u_i(s_i^{\bar{j}}, s_{-i}^{j}) + \sum_{p=1}^{k-1} z_p \cdot I_p^j \cdot \bar{u}_{i,p}(s_i^{\bar{j}}, s_{-i}^{j}) \; \forall \; \bar{j}, j = 1, 2, ..., 20 \tag{33}$$

Hvor $\bar{u}_{i,k}(s_i^{\bar{j}}, s_{-i}^{j})$ angiver elementerne i den transformerede payoff-matrix for spiller $i$, der er af level k. Hertil bør det påpeges, at den absolutte andel af spillere af et givent niveau er aftagende, når $k$ stiger (hvis $z_k > 0$). Den relative fordeling mellem to vilkårlige levels vil dog være konstant, når $k$ stiger. Dette kommer af, at vi antager, at spillere af et vilkårligt level estimerer den sande fordeling af lavere kognitive levels, selvom de fejler i deres vurdering om at være det højeste level. Det betyder, at hvis en level 2 estimerer en relativ fordeling mellem spillere af level 0 og level 1 på $\mathbf{Z}_2 = (0.5, 0.5)$, vil det relative forhold være konstant for alle levels $\geq 2$. En spiller af level 5 vil derfor vurdere, at der er det samme relative forhold mellem L0 og L1'ere, men de kan godt udgøre en mindre absolut andel end

estimatet fra en spiller af level 2.[47] En spiller af level 5 kan derfor have en forventning til fordelingen af laverestående kognitive niveauer, der svarer til $\mathbf{Z}_5 = (0.2, 0.2, 0.3, 0.2, 0.2)$, hvor det relative forhold mellem level 0 og level 1 er identisk med level 2's forventninger.

Vi foretager ligeledes analyser af vores CH-modeller, hvor vi ændrer på best-response kravene. I disse *modificerede* modeller tillader vi, at en spiller kan vælge at spille en strategi, hvis det forventede payoff ved at spille den er marginalt tæt på payoffet ved at spille den optimale strategi.[48] I dette tilfælde vil spilleren mikse ligeligt blandt de strategier, som ligger inden for intervallet.

Vi indfører nu betegnelsen *naiv Nash-ligevægt*. I en verden af adfærdsmæssig spilteori er Nash-ligevægten ikke altid optimal, da antagelser om rationelle forventninger ikke overholdes. Crawford et al. [2013] finder dog, at nogle individer spiller Nash-ligevægten blot fordi, at de besidder kendskab til ligevægtsteori. Nash-ligevægten betragtes dermed som en *naiv* strategi, da den fortsat bygger på antagelsen om rationelle forventninger, som ikke opretholdes i en CH-model. Vi opstiller en model, der inddrager den naive Nash-ligevægt i et kognitivt hierarki. Denne specifikation er i stil med Stahl og Wilson [1995], der i en stokastisk level-k model[49] klassificerer individer, der best-responder på en kombination af kognitive hierarkier og Nash-ligevægten som "worldly types". Denne model betegnes fremover som en Nash-CH model.

Konceptet for Nash-CH modellen er, at vi fortsat har ustrategiske individier, samt individer der spiller med diverse strategiske udgangspunkter. Det kognitive hierarki i vores specifikation er: Level 0, Level 1, Level 2, Level 3, Naivt Nash, Level 5.

Ligning (34) beskriver den generelle løsning til den transformerede payoff-matrix, hvor beslutningskriterierne i Nash-CH modellen er ækvivalente til dem i vores CH-model. Spillere af level $k \geq 2$ best-responder til en fordeling af individer af niveau 0, 1,..., k-2, nash. Den naive Nash-strategi afhænger ikke af strategivalg fra andre spillere men udelukkende af winrate-matricen.

[47] Dette da L2 ikke indser, at der findes højere kognitive niveauer end hende selv.
[48] Vi benytter 1 pct. point som denne grænse.
[49] Deres model er som udgangspunkt lig vores Level-K model, men hvert level har en vis sandsynlighed for at "komme til" at best-responde på et level, der er to levels under dem selv, i stedet for lige under.

Vi tager udgangspunkt i den transformerede payoff-matrix fra den klassiske CH-model (ligning 33). Vi introducerer herefter en sandsynlighedsfunktion, der beskriver de sandsynligheder, som en Nash-ligevægt spiller de forskellige strategier med. Funktionen beskrives ved $P(s^j_{-i}|I^j_{k-1} = 1)$ og angiver dermed sandsynligheden, som den naive Nash-ligevægt tillægger en strategi $j$, givet at den spilles. Den transformerede payoff-matrix for en Nash-CH model, hvor Nash-ligevægten spilles af det kognitive niveau $k$-1, fremgår af ligning (34):

$$\bar{u}_{i,k}(s^{\bar{j}}_i, s^j_{-i}) = z_0 \cdot \frac{I^j_0}{20} \cdot u_i(s^{\bar{j}}_i, s^j_{-i}) + \sum_{p=1}^{k-2} z_p \cdot I^j_p \cdot \bar{u}_{i,p}(s^{\bar{j}}_i, s^j_{-i}) + z_{k-1} \cdot P(s^j_{-i}|I^j_{k-1} = 1) \cdot \bar{u}_{i,k-1}(s^{\bar{j}}_i, s^j_{-i}) \; \forall \; \bar{j}, j = 1, 2, ..., 20 \tag{34}$$

En spiller $i$ af level $k$ best-responder i rene strategier ved at spille den strategi, der giver hende det højeste forventede payoff i den transformerede payoff-matrix.

Til estimering af fordelingen af de kognitive niveauer tildeler vi frie vægte til alle kognitive niveauer $0, 1, ...,k$-1, således at fordelingen af individer bestemmes udelukkende af vægten tilhørende det enkelte niveau. Dette sikrer, at vi opnår en retvisende fordeling af de kognitive niveauer i Nash-CH modellen, da den naive Nash-ligevægt ikke betragter den underliggende fordeling. Vi løser da Nash-CH modellen ligesom en regulær CH-model ved iteration over samtlige levels $k$, hvor Nash-ligevægten betragtes som level $k-1$. Modellen er altså identisk med den normale CH-model for level 0, 1, 2 og 3, med den forskel at hvert level frit vælger andelene under sig selv i stedet for at estimere dem ved brug af en fordeling. Herefter spiller level 4 det naive Nash, og optimeringsproblemet for en spiller af level k (k=5) kan derved løses ved først at estimere de optimale vægte for alle levels $<$ k og dernæst beregne best-response til den transformerede payoff-matrix.

I det kommende afsnit fremvises resultaterne på baggrund af metoden beskrevet i afsnit 3.3, 6, 7 såvel som dette.

# 9 Resultater

Dette afsnit har til formål at belyse vores resultater på baggrund af setuppet i afsnit 8 samt teorigennemgangen i afsnittene 3, 6 og 7. Resultaterne tager afsæt i Nash-ligevægten, som vi løser for vores datasæt. Derefter estimerer vi fordelingen af kognitive niveauer i hhv. Level-K modellen samt vores CH-modeller, som opdeles afhængigt af, om fordelingen benytter en poisson- eller betafordeling. Til sidst fremgår resultaterne fra vores Nash-CH model.

## 9.1 Iterated Elimination of Strictly Dominated Strategies

Vores algoritme[50] gennemgår for hver strategi, $s_i^j$, som spiller $i$ har i winrate-matricen (Tabel 8.1), hvorvidt der findes dominerede strategier. Vi itererer over alle kombinationer af strategier, der findes i winrate-matricen, hvorved at vi sammenligner payoffs for samtlige udfald for en given strategi. Hvis en strategi er svagt domineret betyder det, at en anden strategi giver et payoff, der er *mindst* lige så stort for alle mulige udfald.

Vi finder ingen svagt dominerede strategier i vores datasæt (og da ingen stærkt dominerede). Agenterne behøver derved ikke at inkorporere elimination af dominerede strategier i dette spil, da det ikke er muligt.

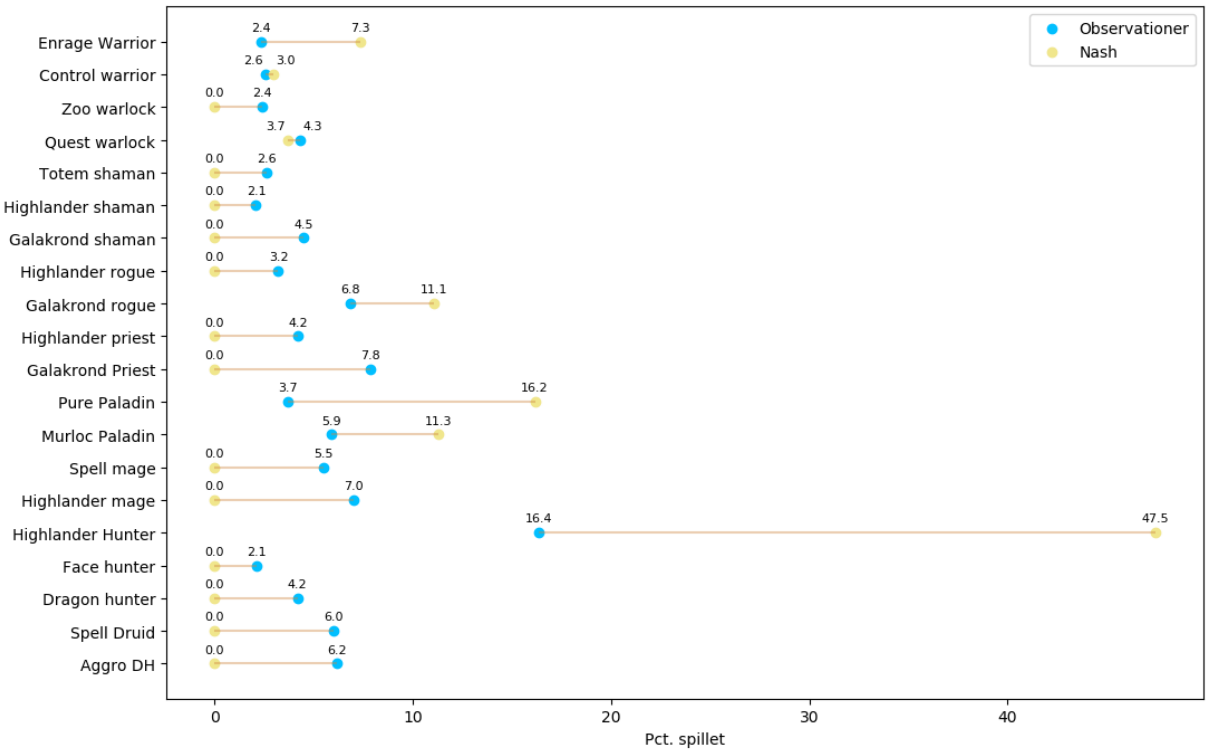
## 9.2 Pure og Mixed Nash Equilibrium

Ved at benytte minimax-metoden beskrevet i afsnit 3.3 finder vi ét enkelt MSNE ved at løse spillet, der fremgår af vores winrate-matrix $\mathbf{A}$, som ses i tabel 8.1. $\mathbf{p}^T$ angiver spiller $i$'s vægte, og $\mathbf{q}$ angiver spiller $-i$'s vægte, men her betragter vi 20 mulige strategier i stedet for 3.

Vi finder, at $\mathbf{p}^T = \mathbf{q}$ =(0, 0, 0, 0, 0.475, 0, 0, 0.113, 0.162, 0, 0, 0.111, 0, 0, 0, 0, 0.037, 0, 0.029, 0.073) som er afbilledet i figur 9.1 nedenfor, hvor der sammenlignes med det observerede data, hvilken vi angiver $\mathbf{o}$ = (0.0617, 0.0599, 0.419, 0.0213, 0.1639, 0.0702, 0.0549, 0.0587, 0.0369, 0.0785, 0.0423, 0.0684, 0.0322, 0.0452, 0.0210, 0.0263, 0.042, 0.0241, 0.0258, 0.0237).[51]

[50] Se appendiks C.
[51] Andele er beregnet ved brug af tabel 8.2.

**Figur 9.1:** Mixed Strategy Nash Equilibrium og Observeret Data




Til optimering af modellerne, samt alt resultatbehandling, anvendes en faktor 100 på andelsmatricerne: $100 \cdot \mathbf{p}^T = \tilde{\mathbf{p}}^T$, $100 \cdot \mathbf{A} = \tilde{\mathbf{A}}$ og $100 \cdot \mathbf{o} = \tilde{\mathbf{o}}$.[52]

Vi finder da $l^2$-Normen og derved også $SS_{res}$ for denne fordeling jf. afsnit 7.2:

$$\begin{aligned}\sum &(6.17-0)^2+(5.99-0)^2+(4.19-0)^2+(2.13-0)^2+(16.39-47.5)^2+(7.02-0)^2\\ &+(5.49-0)^2+(5.87-11.3)^2+(3.69-16.2)^2+(7.85-0)^2+(4.23-0)^2+(6.84-11.1)^2\\ &+(3.22-0)^2+(4.52-0)^2+(2.1-0)^2+(2.63-0)^2+(4.3-3.7)^2\\ &+(2.41-0)^2+(2.58-2.9)^2+(2.37-7.3)^2\\ &=1496=SS_{res}=l^2\text{-norm}\end{aligned}$$

[52]Dette har betydning for $l^2$-skalaen.

Vi beregner $R^2$ jf. afsnit 7.3, hvor vi først beregner $SS_{tot}$:

$$\begin{aligned} SS_{tot} = \sum &(6.17-5)^2 + (5.99-5)^2 + (4.19-5)^2 + (2.13-5)^2 + (16.39-5)^2 + (7.02-5)^2 \\ &+(5.49-5)^2 + (5.87-5)^2 + (3.68-5)^2 + (7.85-5)^2 + (4.23-5)^2 + (6.84-5)^2 \\ &+(3.22-5)^2 + (4.52-5)^2 + (2.1-5)^2 + (2.63-5)^2 + (4.32-5)^2 + (2.41-5)^2 \\ &+(2.58-5)^2 + (2.37-5)^2 \\ &= 197 \end{aligned}$$

Og da beregner $R^2$:

$$\begin{aligned} R^2 &= 1 - \frac{SS_{res}}{SS_{tot}} \\ &= 1 - \frac{1496}{197} \\ &= -6,59 \end{aligned}$$

Hvilket er negativt. En negativ $R^2$ betyder, at modellen beskriver variansen i data *dårligere* end en model med prædiktioner lig middelværdien, hvilket ville medføre et $R^2$ på 0.[53] Bemærk, at denne negative værdi i vores tilfælde kan være ekstremt misvisende, dels fordi at vi har at gøre med andele, der er negativt korrelerede, men særligt da vi betragter et så komplekst spil, hvor en antagelse om at en andel af populationen spiller tilfældigt, ikke nødvendigvis er urealistisk. Mere om dette kan læses i de følgende resultatafsnit.

Som beskrevet i afsnit 7.1 og 7.3 vægter vi alle observationer ligeligt ved vores udregning af $l^2$ og $R^2$. Dette gør, at resultaterne for disse er sårbare over for outliers, som MSNE prædikerer. Dette sker f.eks., da man i MSNE spiller "Highlander Hunter" i 47,5 pct. af sine spil, hvilket trækker $SS_{res}$ ekstremt op. Modellen prædikerer, at 13 deck slet ikke bør spilles i MSNE, da de beregnes med en andel på 0 pct. Trods et så højt antal forkerte prædiktioner er det akkumulerede bidrag til $SS_{res}$ fra alle disse på 302, mens det alene fra Highlander Hunter er på 965. Denne outlier vægter altså 65 pct. af den samlede afvigelse, hvorfor at $R^2$-værdien dermed ikke er specielt robust over for outliers. Tabel 9.1 sammenligner $R^2$ og $l^2$-normen for modellen.

[53] Modelprædiktionerne er lig middelværdien ved $\pi_i = \bar{\pi}_i$.

**Tabel 9.1:** $R^2$ samt $l^2$-norm (MSNE)

| Model | $R^2$ | $l^2$-norm |
|---|---|---|
| Mixed Nash | -6,59 | 1496 |

Der kan dog fremhæves visse vigtige indsigter fra disse resultater. På trods af at ikke alle spiller fuldt rationelt, er "Higlander Hunter", med en spilleandel på 16,4 pct., den strategi, der spilles mest i data. Denne strategi er ligeledes den strategi i MSNE med den højeste andel, hvilket tyder på, at spillere foretager en vis grad af rationel overvejelse, når de skal vælge deres strategi.

Man kunne da forestille sig, at folk spillede deck med en høj gennemsnitlig winrate. Dette ville dog ikke forklare f.eks. den relativt høje andel "Aggro Demon Hunter". Dette deck er dog stærkt mod netop Highlander Hunter (52,29 pct. winrate), kun overgået af "Face Hunter" (55 pct.) og "Murloc Paladin" (52,94 pct.). Det kunne altså ligne, at folk foretager en vis grad for iteration over, hvilke strategier, som de kommer til at spille imod. Netop denne form for tankeproces testes i vores Level-K model, hvor resultaterne af denne kan ses i afsnit 9.3 nedenfor.

## 9.3 Level-K

Som beskrevet i afsnit 6.1 antager vi i vores model, at en spiller af level 0 spiller tilfældigt. Camerer et al. [2004] og lignende litteratur laver samme antagelser om agenter af dette level. Dette svarer i vores tilfælde til, at en spiller $i$ af level 0 spiller strategi $j$ med sandsynligheden $p_0(s_i^j) = 1/m_i \ \forall \ j$, som i vores tilfælde bliver $p_0(s_i^j) = \frac{1}{20} \ \forall \ j$.

En L1 vil antage, at hun udelukkende spiller mod en L0, som benytter disse vægte på samtlige af deres strategier. Dette betyder essentielt, at L1 vil løse en payoff-matrix lignende tabel 3.3 for $\mathbf{p}^T$, der er beskrevet i afsnit 6.1, men i stedet for en $3 \times 3$ matrix er matricen af størrelsen $20 \times 20$, hvor $q_1 = q_2 = ... = q_{20} = 0.05$, $K = L_0$, $R = L_1$ og hvor payoffs, f.eks. $x_{11}$, angiver L1's payoffs fra winrate-matricen, mens $y_{11}$ angiver payoffs til L0'eren.[54]

Essentielt svarer dette til, grundet den uniforme fordeling af strategier for level 0, at en L1

[54] $y_{11}$ er da per konstruktion $1 - x_{11}$.

vil vælge den strategi, der har den højeste gennemsnitlige winrate, hvilket er $p_1(s_i^5) = 1$, hvor $s_i^5$ ="Highlander Hunter". Denne har en gennemsnitlig winrate på 54,44 pct., mens den andenhøjeste er "Murloc Paladin" med en gennemsnitlig winrate på 54,25 pct. En L2 vil da best-responde til den strategi, L1 spiller, ved at vælge den strategi, der har det højeste payoff, netop når $j = 5$. dvs. $BR_i(s_{-i}^5) = s_i^4$ = "Face Hunter". Den samme metode gentages, hvor resultatet af de forskellige levels kan ses i Tabel 9.2.

**Tabel 9.2:** Level-K Best-Responses

| Level | Deck |
|---|---|
| L0 | Tilfældigt |
| **L1** | **Highlander Hunter** |
| L2 | Face Hunter |
| L3 | Pure Paladin |
| L4 | Galakrond rogue |
| L5 | Spell Druid |
| L6 | Murloc Paladin |
| L7 | Enrage Warrior |
| L8 | Highlander Priest |
| **L9** | **Highlander Hunter** |
| *L10* | *Face Hunter* |

Det bemærkes, at en spiller af level 9 svarer til en level 1, hvorfor de fremtidige levels $\geq 9$ da vil iterere over de samme deck som for level 1 til 8. Det er netop denne cykliske adfærd, som Chong et al. [2005] påpeger, og som vi ser i eksemplet fra afsnit 6.1.

Level-K modellen benytter altså otte ud af tyve mulige deck, hvor Nash-ligevægten benytter syv. For de resterende decks tilegnes der en vægt på 0 i Nash-ligevægten, mens Level-K modellen tillader, at disse 12 deck bliver spillet, da de kan spilles af L0-agenter. Dermed undgår vi potentielle problemer, som kan opstå under parameterestimation ved SQP, som nævnt i afsnit 7.1.1.

Som beskrevet i afsnit 6.1 benytter vi en poissonfordeling til at estimere, hvilken populationsfordeling der passer bedst på data. Vi anvender Minimum Distance Estimatoren, der er beskrevet i afsnit 7.1, og finder, at den parameterværdi, der giver den mindste afvigelse fra observeret data, er $\tau = 0,13$. For en poissonfordeling svarer dette til, at 87,78 pct. af populationen er level 0. Fordelingen af levels kan ses i tabel 9.3 nedenfor. Der sammenlignes ligeledes med den estimerede fordeling af kognitive niveauer, som Camerer et al.

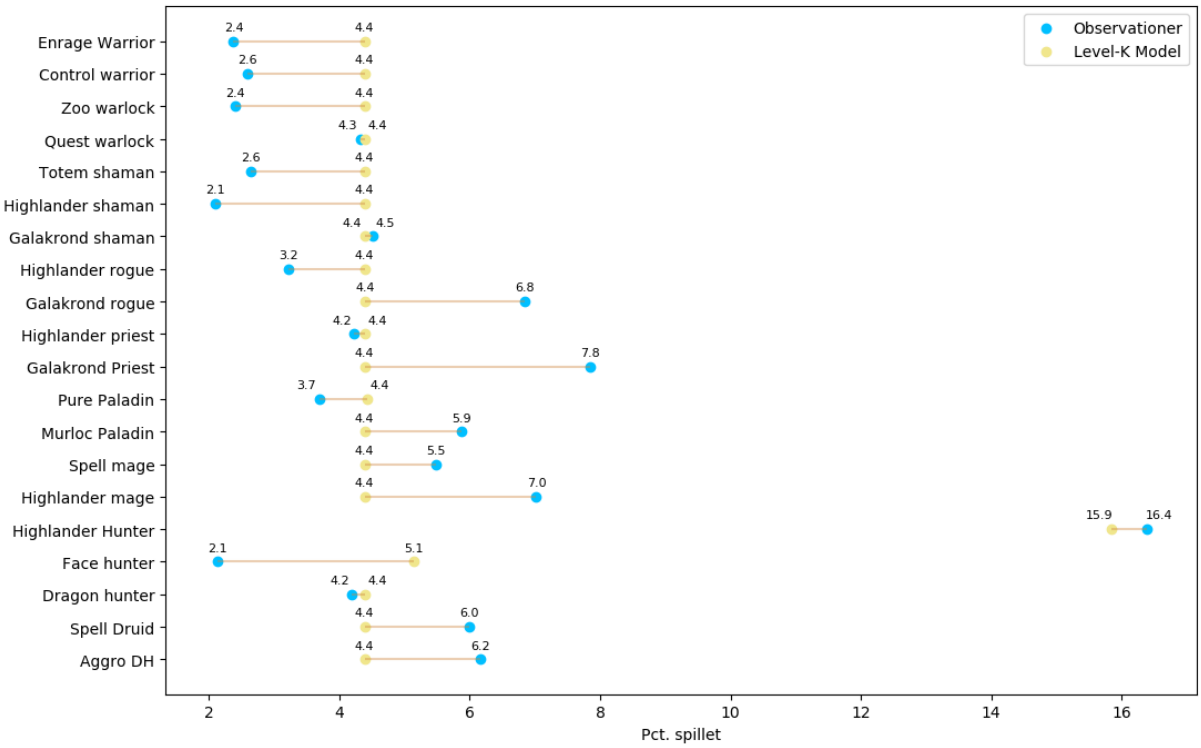
[2004] finder.[55]

**Tabel 9.3:** Levelfordeling afhængigt af model (Level-K)

| Level | Andel af population (pct.) | |
|---|---|---|
| | Level-K | Camerer (2004) |
| L0 | 87,75 | 32,93 |
| L1 | 11,47 | 37,54 |
| L2 | 0,75 | 21,40 |
| L3 | 0,03 | 8,13 |
| L4 | $< 0,01$ | —- |

Med en sådan levelfordeling prædikerer modellen strategiandelene $\mathbf{p}^T = (0.0439, 0{,}0439, 0{,}0439, 0.0514, 0.1585, 0.0439, 0.0439, 0.0439, 0.0442, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439, 0.0439)$, som sammenlignes med det observerede data i Figur 9.2.

**Figur 9.2:** Level-K og Observeret data




Det bemærkes, at selvom Level-K modellen har muligheden for at lægge vægt på levels op til level 8, er vægten herpå tæt på nul. Dette underbygger vores valg af at kigge på

[55] Bemærk, at de kigger fra en Level 4's synspunkt, hvorfor de ikke har nogen vægt på Level 4. Endvidere er deres model en CH-model og *ikke* en level-k.

maks 5 levels i CH-modellerne. Det ses dog, at både level 2 og level 3 også har en relativt lille andel af populationen, hvorfor modelprædiktionerne i stort omfang er ens for de fleste strategier undtagen Highlander Hunter (L1) og Face Hunter (L2), netop da der er en klar overvægt af level 0 spillere. Camerer et al. [2004] finder belæg for et relativt konsistent estimatet for $\tau$ på 1.14, hvilket svarer til, at 31.9 pct. af populationen agerer som et individ af level 0.[56] Vores Level-K model estimerer altså 54,82 pct.point. flere agenter, der spiller tilfældigt, end Camerer et al. [2004].

En vigtig pointe, der bør bemærkes her, er, at antallet af spillere af L1 og L2 afhænger kraftigt af, at outlieren (Highlander Hunter) netop er L1. Havde dette deck imidlertid været det, som blev spillet af en L2'er, ville den fordeling, der bedst passede på data, højst sandsynligt ændre sig markant, da MDE fortsat ville minimere afstanden til denne strategi. I et sådant tilfælde ville der dog være to modsatrettede effekter ved at skrue andelen af L2'ere op. $l^2$-normen vil blive lavere grundet det bedre match mellem data og L2, mens at strategien, der spilles af en L1'er, højst sandsynligt ville blive spillet for meget, hvilket ville hæve $l^2$.[57]

På trods af at dette blot er spekulationer, kan vi bruge tankestrømmen til at sige noget om de resultater, vi *faktisk* får. Det tyder på, at spillerne i høj grad enten spiller tilfældigt eller laver ét skridts tænkning, da Highlander Hunter spilles med en frekvens, der er over tre gange så høj som et gennemsnitligt deck. For at sammenligne modellen med data beregner vi $l^2$-normen på 64,88 og får derved et $R^2$ på 0,67, som kan aflæses i tabel 9.4.

**Tabel 9.4:** $R^2$ samt $l^2$-norm (MSNE/Level-K)

| Model | $R^2$ | $l^2$-norm |
|---|---|---|
| Mixed Nash | -6,59 | 1496 |
| Level-K | 0,67 | 64,88 |

Level-K modellen forklarer altså variansen i data markant bedre, end Nash-ligevægten gør, men også bedre end en baseline model. Vi bemærker, at især Highlander Hunter (L1), som var en markant outlier i Nash-ligevægten, bliver prædikeret meget tættere på

[56] Bemærk, at dette *ikke* er resultater fra en level-k model, men derimod en CH-model.
[57] Umiddelbart ville sidstnævnte effekt dominere, men dette kan ikke konkluderes i et tænkt eksempel.

observeret data. Dette kunne dog forklares ved, at MDE jo netop i overvejende grad minimerer outliers. Det betyder, at estimatoren i stor grad finder den værdi af $\tau$, der sikrer, at netop afstanden mellem denne relativt høje observerede deckandel bliver internaliseret.

Sammenlignet med Nash-ligevægten passer modellen betydeligt bedre på data, hvor størstedelen af spillerne spiller tilfældigt kombineret med en del, der best-responder netop på dette. Dette tyder på, at der foregår en vis form for overvejelse, når agenten vælger sit deck. I næste afsnit gennemgås resultaterne fra vores CH-modeller, hvor agenten danner en overbevisning om fordelingen af levels, som de modstandere, hun dyster imod, består af.

## 9.4 Kognitivt Hierarki

I dette afsnit beskrives analysen og resultaterne fra vores fem kognitive hierarkimodeller. Vi anvender to poisson CH-modeller og to beta CH-modeller. Hver af disse findes i både en standard samt modificeret version, og udover disse fremlægges resultaterne fra Nash-CH modellen, der benytter frie vægte til estimation af populationsfordelingen. Foruden at fremlægge vores resultater uddrages sammenhænge og forskelle til lignende litteratur (Camerer et al. [2004], Crawford et al. [2013][58]).

Inden vi gennemgår resultatprocessen, og dertilhørende mulige forklaringer for de enkelte modeller, summerer vi kort hovedresultaterne. For en mere detaljeret forklaring, af hvordan de enkelte delresultater opnås, se da i de tilhørende underafsnit.

Med udgangspunkt i metoden beskrevet i afsnit 8, teorien for modellerne i afsnit 6 samt parameterestimationen beskrevet i afsnit 7 viser tabel 9.5 $R^2$, $l^2$ samt de dertilhørende optimale parametre for modellerne. Vi husker dog, at værdierne for $R^2$ ikke er super interessante, men det relative forhold mellem dem gør det muligt at sammenligne resultater på tværs modellerne. Det ses for avancerede spil af denne art, at fuldt rationelle modeller såsom MSNE passer ekstremt dårligt på data, mens Level-K modellen passer markant bedre, og CH-modellerne[59] passer bedst. Der er altså belæg for, at CH-modeller, forklarer

[58] Crawford et al. [2013] fremlægger et litteraturreview, som vi sammenligner med.
[59] Eksklusiv de modificerede.

menneskelig adfærd bedre end modeller, der antager fuldkommen rationalitet.

**Tabel 9.5:** $R^2$, $l^2$-norm og parameterestimat

| Model | $R^2$ | $l^2$-norm | Parameterestimat | |
|---|---|---|---|---|
| | | | $\tau$ | $\alpha$ ; $\beta$ |
| Mixed Nash | -6,59 | 1496 | – | – |
| Level-K | 0,67 | 65,42 | 0,13 | – |
| SP-CH | 0,70 | 58,83 | 0,141 | – |
| MP-CH | 0,42 | 114 | 0,15 | – |
| SB-CH | 0,70 | 58,34 | – | 0,206 ; 2,416 |
| MB-CH | 0.42 | 114 | – | 0,281 ; 3,215 |
| Nash-CH | 0,70 | 58,34 | – | – |

SB-CH (Standard beta-CH) samt Nash-CH modellerne har de laveste $l^2$ værdier[60] og fitter derved bedst på data. Ligeledes bør nævnes, at SP-CH (Standard poisson-CH) er meget tæt på de to. For modellerne, der benytter en poissonfordeling, er $\tau$ lavest for Level-K modellen på 0,13 og højest for MP-CH (Modificeret poisson-CH) modellen på 0,15, mens SP-CH modellen ligger på 0,141. Sammenlignes der med Camerer et al. [2004]'s estimat på 1,14 for simple spil, tyder det på, at modellerne har svært ved at forklare lige så meget af data for avancerede spil. Det er dog bemærkelsesværdigt, at vores egen SB-CH passer marginalt bedre på data, da fordelingen ikke, så vidt vi ved, er blevet benyttet af andre forfattere.

Parameterværdierne resulterer i levelfordelingen, som fremgår af tabel 9.6 nedenfor. Afhængigt af model finder vi mellem 8,58 og 12,97 pct. af populationen, som er svagt kognitive (level = 1), mellem 0.9 og 5.02 pct. der er kognitivt stærke (level $\geq$ 2) og derved omkring 86 pct. ustrategiske spillere.

**Tabel 9.6:** Levelfordeling afhængigt af model

| Level | Andel af population (pct.) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Nash-CH | MB-CH | SB-CH | MP-CH | SP-CH | Level-K | Camerer (2004) |
| L0 | 86,38 | 86,38 | 86,38 | 86,00 | 86,87 | 87,75 | 32,93 |
| L1 | 12,06 | 9,59 | 8,58 | 12,97 | 12,22 | 11,47 | 37,54 |
| L2 | $< 0,01$ | 3,12 | 3,48 | 0,98 | 0,86 | 0,75 | 21,40 |
| L3 | 1,55 | 0,81 | 1,30 | 0,049 | 0,04 | 0,03 | 8,13 |
| L4 | 0 | $< 0,01$ | 0,24 | $< 0,01$ | $< 0,01$ | 0,00 | — |

[60] SB-CH er marginalt bedre på 8. decimal.

Resultaterne viser et konsistent estimat på levelfordelingen, men afviger fra Camerer et al. [2004]. For at teste om der er belæg for, at nogle agenter laver flere kognitive iterationer end andre, sammenligner vi på tværs af high- og low ability spillere. Dette gøres for SP-CH og SB-CH modellerne, hvor det fremgår af tabel 9.7, at flere high ability spillere er af højere hierakisk level og derved tænkere mere over sit strategivalg.

**Tabel 9.7:** Levelfordeling afhængigt af ability

| Level | Andel af population (pct.) | | | | | |
|---|---|---|---|---|---|---|
| | *High ability* | | *Low ability* | | *Baseline* | |
| | SP-CH | SB-CH | SP-CH | SB-CH | SP-CH | SB-CH |
| L0 | 84,29 | 84,04 | 94,15 | 94,52 | 86,87 | 86,38 |
| L1 | 14,40 | 10,05 | 5,7 | 3,69 | 12,22 | 8,58 |
| L2 | 1,23 | 4,10 | 0,17 | 1,30 | 0,86 | 3,48 |
| L3 | 0,07 | 1,51 | <0,01 | 0,42 | 0,04 | 1,30 |
| L4 | <0,01 | 0.28 | <0,01 | 0,06 | <0,01 | 0,24 |

Selv for high ability spillerne bemærkes den høje andel tilsyneladende ustrategiske spillere. Som nævnt i afsnit 7, benytter vi vægtningsmatricen $\mathbf{W} = \mathbf{1}$ der i sit afstandsmål vægter forskellen mellem data og prædiktioner ligeligt for alle strategier. Vi prøver da at løsne op for disse antagelser for at se, hvilke implikationer det har for optimale parameterværdier, hvis vi tager udgangspunkt i strategivalgene, som Level-K modellen fra afsnit 9.3 prædikerer. Vi vælger da at vægte de otte strategier, der bliver benyttet af level 1-8 i tabel 9.2, højere, end de der ikke anvendes. Specifikt opstiller vi en diagonal vægtningsmatrix $\bar{\mathbf{W}} = diag(w)$, hvor $w$ er en vektor af størrelse 20 med værdierne $\frac{1}{10}$, når strategierne spilles i Level-K modellen og $\frac{2}{12}$[61] ellers. Vi har dermed, at $w = (2/12, 0.1, 2/12, 0.1, 0.1, 2/12, 2/12, 0.1, 0.1, 2/12, 0.1, 0.1, 2/12, 2/12, 2/12, 2/12, 2/12, 2/12, 2/12, 0.1)$.

Ved optimering af SP-CH og SB-CH modellerne med disse vægte, finder vi den optimale levelfordeling, som fremgår af tabel 9.8. Det ses, at CH-modellerne, der benytter en poissonfordeling, er identiske uafhængigt af vægtningsmatricen. Dette skyldes poissonfordelings begrænsning ved kun at have én parameter at optimere ud fra. Da Highlander Hunter fortsat er en outlier, vil fordelingen stadig finde det estimat, der internaliserer denne bedst.

[61] Den resterende vægt er $\frac{1-0,8}{12} = 1,67$.

**Tabel 9.8:** Levelfordeling med alternativ vægtmatrix

| Level | Andel af population (pct.) | | | | |
|---|---|---|---|---|---|
| | Vægtet SB-CH | SB-CH | Vægtet SP-CH | SP-CH | Camerer (2004) |
| L0 | 78,12 | 86,38 | 86,87 | 86,87 | 32,93 |
| L1 | 8,09 | 8,58 | 12,22 | 12,22 | 37,54 |
| L2 | 5,35 | 3,48 | 0,86 | 0,86 | 21,40 |
| L3 | 4,27 | 1,3 | 0,04 | 0,04 | 8,13 |
| L4 | 4,17 | 0,24 | $< 0,001$ | $< 0,001$ | – |

Hvis resultatet skulle ændre sig, ville vægtningsmatricen skulle sætte mindre vægt på denne strategi end de andre. CH-modellerne, som estimeres via. en betafordeling, ændrer sig dog markant. Det ses, at andelen af levels $> 0$ er steget fra 13,62 til 21,88 - en stigning på 8,26 pct. point.

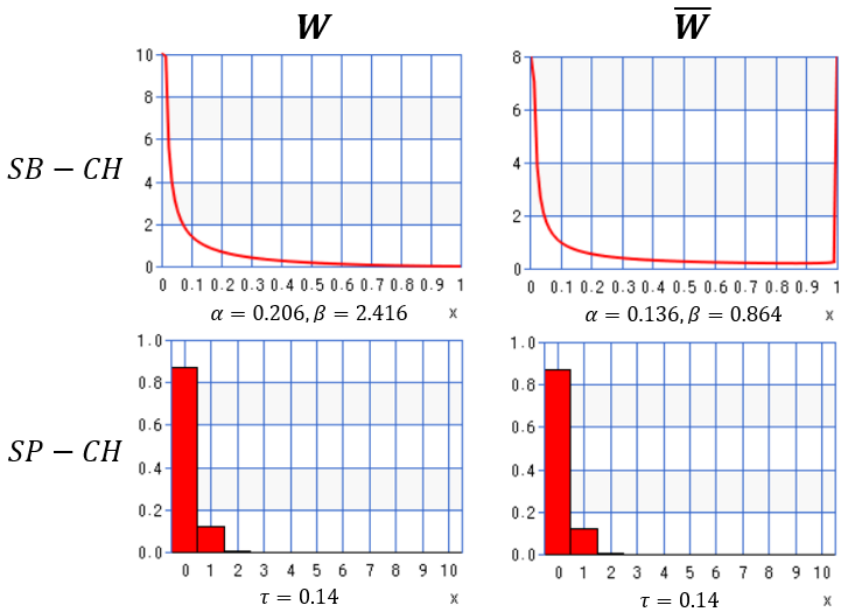
Optimeringen af parameterværdierne ved brug af de ligeligt fordelte vægte $\mathbf{W}$ har principielt været det samme som at minimere $l^2$-normen og derefter finde de parameterværdier, der gav denne værdi. Med de nye vægte $\bar{\mathbf{W}}$ er det imidlertid ikke sandt, da $l^2$-normen stadig ville benytte ligelige vægte til afstandsmålet. Det bliver derfor svært at sammenligne de to modellers fit på data, da optimeringen laves på baggrund af forskellige parametre.[62]

Hvis man i optimeringsprocessen vælger at nedprioritere de strategier, som Level-K modellen angiver som ustrategiske, prædikerer SB-CH modellen altså en højere andel af kognitivt tænkende spillere. Betafordelingens fleksibilitet giver altså en markant forbedring ift. poissonfordelingen, hvis man udelukkende prøver at estimere højere kognitive niveauer.

I figur 9.3 er fordelingerne for SP-CH og SB-CH modellerne visualiseret for hhv. $\mathbf{W}$ og $\bar{\mathbf{W}}$. Som nævnt er $\tau$ værdien den samme for SP-CH, hvorfor de to fordelinger er ens. Vi ser, at hypotesen omkring $\alpha, \beta < 1$ opstillet i afsnit 6.2.3 faktisk er overholdt for SB-CH i tilfældet med $\bar{\mathbf{W}}$. Der bliver i dette tilfælde lagt mest vægt på lave levels, men også en del vægt på levelet lige under en selv.

[62] Man kunne alternativt lave en vægtet $l^2$-norm, der ville blive $\bar{l}^2 = \sum_i^{20}((\hat{\pi}_i - \pi_i)\bar{\mathbf{W}})^2$, men dette udelades for simplicitet.

**Figur 9.3:** Fordeling visualiseret afhængigt af vægtmatrix




I de kommende underafsnit 9.4.1, 9.4.2, 9.4.3, 9.4.4 samt 9.4.5 bliver resultaterne fra de enkelte modeller uddybet. Herefter gennemgår afsnit 9.5 de robusthedstjek, vi laver, herunder sammenligning af high- og low ability spillere, samt hvor konsistente resultaterne er over tid. Vi finder, at parameter- og levelestimaterne er konsistente for alle andre rapporter, på nær én outlier.

### 9.4.1 Standard Poisson CH-Model

Med udgangspunkt i fremgangsmetoden beskrevet i afsnit 8 betragter vi SP-CH modellen, der gennem en enkelt parameter $\tau$ beskriver fordelingen af populationen. En høj værdi af $\tau$ resulterer i en overbevisning om, at man spiller mod levels tæt på ens eget, mens en lav værdi af $\tau$ resulterer i, at man overvejende antager, at man spiller mod ustrategiske individer. Dette vil sige, at hvis $\tau \approx 0$,[63] vil CH-modellen antage at spille mod en fordeling bestående udelukkende af L0'ere. Der vil da for ethvert level i SP-CH vil blive spillet Highlander Hunter, der er den optimale best-response strategi mod ustrategiske spillere, som nævnt i afsnit 9.3. For en høj værdi af $\tau$ vil man antage, at man spiller mod levels tæt på sit eget, og derved vil best-response funktionen for SP-CH konvergere mod best-response for Level-K modellen, når $\tau$ øges.

[63] Bemærk at fordelingen kræver $\tau > 0$ så der menes marginalt tæt på nul.

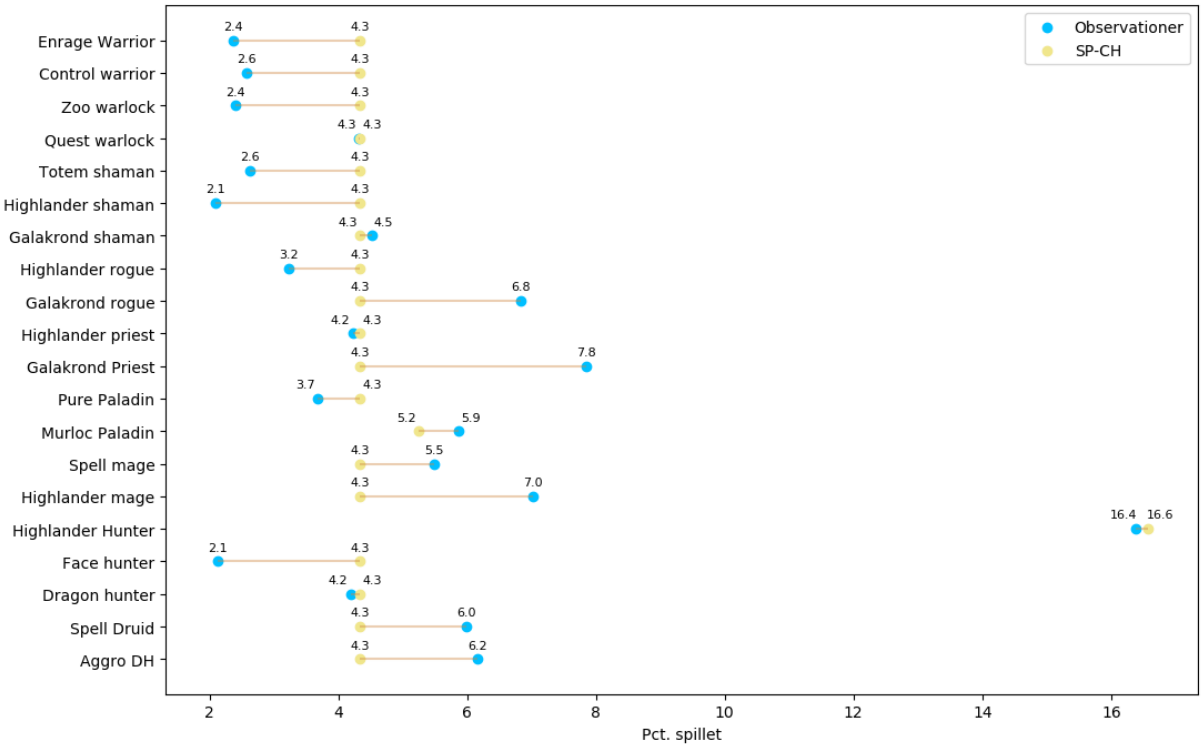
Vi estimerer $\tau$ ved at minimere afstanden fra vores modeloutput til observationerne ved brug af MDE og finder, at det $\tau$, der minimerer afstanden, er $\tau = 0.141$. I tabel 9.6 ses, at dette estimat for $\tau$ medfører, at 86,87 pct. af populationen estimeres til at være af level 0 (ustrategiske), 12,22 pct. estimeres som level 1 (svagt kognitivt tænkende) og da mindre end 1 pct. over level 1 (stærkt kognitivt tænkende). Der sammenlignes med resultater fra vores andre modeller såvel som resultater fra Camerer et al. [2004].

Vi finder en større vægt på level 0 individer end den resterende litteratur (Camerer et al. [2004], Crawford et al. [2013]) og er tæt på resultatet fra Level-K modellen. I lignende litteratur findes $\tau$ oftest til at være et sted mellem 1 og 1,5. Dette kan potentielt skyldes, at poissonfordelingen er for infleksibel til at forklare spredningen af kognitive niveauer i avancerede spil, da $\tau$ angiver middelværdien og variansen. Den kan dermed ikke forklare særlige fænomener, som f.eks. at populationen både dækker over spillere, der spiller for sjov, men også dem der spiller udelukkende for at vinde.

Camerer et al. [2004] påpeger, at der kan forekomme høje $\tau$-værdier i situationer med erfarne og højtuddannede agenter, der deltager i simple spil. Vi finder, at estimatet mindskes ved større, og derved mere komplekse spil, såsom denne $20 \times 20$ matrix. Vi ser, at andelen af højere levels i populationen er større for vores SP-CH model end for Level-K modellen.

De strategiandele, som SP-CH modellen prædikerer for $\tau = 0.141$ svarer til $\mathbf{p}^T_{SP-CH} =$ (0.0434, 0.0434, 0.0434, 0.0434, 0.1657, 0.0434, 0.0434, 0.0525, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434, 0.0434). Disse andele er afbilledet i figur 9.4, hvor der sammenlignes med observeret data. Det ses dermed, at på trods af den lave $\tau$ værdi vil strategien for en spiller af level $\geq 2$ (der alle spiller Murloc Paladin) være forskellig fra L1's strategi.

**Figur 9.4:** Standard Poisson CH-model og Observeret Data




Vi finder, at $l^2$-normen er 58,78 og er derved aftaget ift. både MSNE samt Level-K modellen, hvilket medfører en øget forklaringsgrad. Dette er i overensstemmelse med lignende litteratur, der fraskriver Nash-ligevægten som en potentiel forklaringsfaktor for, hvordan mennesker *faktisk* spiller i større spil. Normen svarer til et $R^2$ på 0,70, som sammenlignes med de andre modeller i Tabel 9.5. Vi finder, at størstedelen af forbedringen i $l^2$-normen relativt til MSNE skyldes en nedjustering af Highlander Hunter fra 47,5 til 16,6 pct. Endvidere forbedres $l^2$-normen drastisk af, at der ikke er såkaldte "døde" strategier,[64] hvilket sikrer et mindre afstandsmål for størstedelen af de strategier, der ikke spilles i Nash-ligevægten. Dette resultat indikerer derfor, på baggrund af en stigning i forklaringsgraden og et lavere afstandsmål i tabel 9.5, at SP-CH modellen beskriver strategivalget i Hearthstone bedre end Nash-ligevægten. Dette kunne skyldes, at modellen tillader spilleren at variere sin strategi afhængigt af $\tau$. Der vil altså i højere grad fokuseres på både at have et alsidigt deck (dvs. et der er godt mod L0), men også at have et, der er godt mod de specifikke deck, der bliver spillet af levels tæt på en selv. Dette medvirker, at "Murloc Paladin" i højere grad spilles.

[64] Strategier med en sandsynlighed på 0.

### 9.4.2 Modificeret Poisson CH-Model

Udover SP-CH modellen ovenfor har vi konstrueret en modificeret version, som vi kalder MP-CH. Denne version mikser sit strategivalg uniformt mellem to strategier, hvis det forventede payoff ved at spille hvert af dem er inden for et arbitrært interval (som vi sætter til 1 pct. point). Formålet med dette er at forsøge bedre at afspejle den beslutningsproces, som spilleren står over for, når hun vælger sit strategivalg. I SP-CH vil en winratefordel på $\leq 0.001$ pct.point. altid vil være til at foretrække, hvilket repræsenterer det mekaniske beslutningsgrundlag, som en spiller har i et kognitivt hierarki. MP-CH modellen prøver altså at tillade, at agenten ikke er mere fleksibel i sin beslutningstagen. Vi gør det i håb om at sænke de mentale omkostninger, når spilleren vælger sin optimale strategi.

Vi finder, at MP-CH modellen estimerer en $\tau$ værdi på 0.151, som er en svag stigning fra SP-CH. I tabel 9.6 ses levelfordelingen for MP-CH modellen. MP-CH modellen estimerer da en marginalt lavere andel af level 0 i populationen (86,00 pct.) end SP-CH. Der opleves da en snæver stigning i andelen af individer for de højere kognitive niveauer.

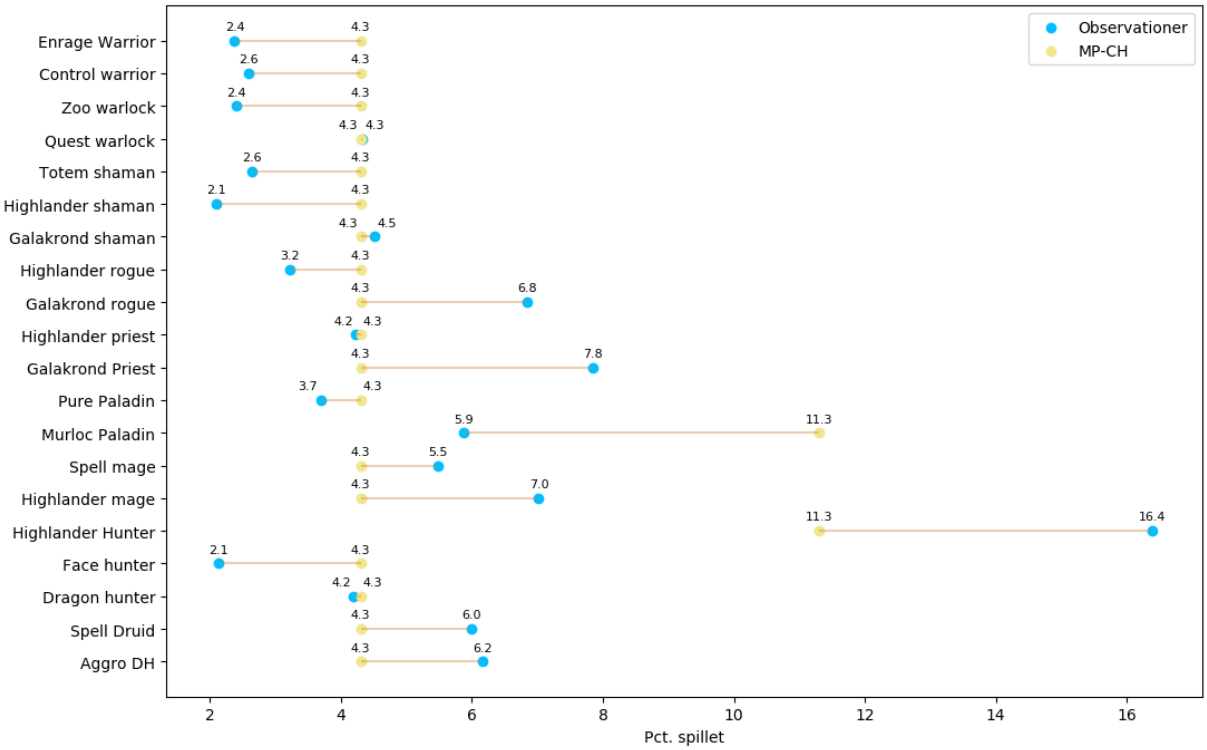
Stærkt kognitive individer (level $\geq 2$) er forsvindende lille på tværs af de to CH-modeller relativt til lignende litteratur. Dog finder Camerer et al. [2004] og Crawford et al. [2013], at andelen af individer $> 2$ udgør den mindste andel af populationen, hvorfor de hovedsageligt fokuserer på at forklare tilstedeværelsen af agenter af level 0, level 1 og level 2.[65]

MP-CH modellen estimerer strategisandsynlighederne $\mathbf{p}^T_{MP-CH}$ =(0.043, 0.043, 0.043, 0.043, 0.113, 0.043, 0.043, 0.113, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043, 0.043),[66] som er afbilledet i figur 9.5, der sammenligner med observeret data.

[65] De inddrager højere niveauer i deres analyser, men fokuserer hovedsageligt på at forklare tilstedeværelsen af kognitive niveauer op til L2.

[66] Bemærk, at sandsynlighederne for decks spillet af L0 faktisk er forskellige fra SP-CH modellen, den er blot lav.

**Figur 9.5:** Modificeret Poisson CH-model og Observeret Data




MP-CH modellens $l^2$-norm er 114 med et $R^2$ på 0,42, som kan ses i tabel 9.5. Trods et højere $\tau$ passer modellen umiddelbart dårligere på data, end SP-CH gør. Endvidere passer den faktisk værre, end Level-K modellen gør. Dette skyldes vores antagelse om, at to deck, der giver approksimativt samme payoff, bliver mikset imellem 50/50 (eller 33/33/33 hvis der er tre osv.). Et sådan 'tvungent' miks har muligheden for at fitte bedre på data, hvis observerede deckandele på de to deck er meget lig hinanden, men kan også skævvride prædiktionen, hvis de er langt fra hinanden. Sidstnævnte er tilfældet hos os, hvor vi observerer en andel på 16,4 pct. for Highlander Hunter og 5,9 pct. for Murloc Paladin. Payoff fra disse to decks er imidlertid inden for de omtalte 1 pct. point., hvorfor MP-CH modellen tvinger nogle spillere til at mikse ligeligt mellem de to deck. Dette resulterer i, at begge deck spilles med en sandsynlighed på 11,3 pct., som det ses i figur 9.5, hvilket forårsager et større $l^2$.

Havde vi derimod givet spilleren lov til selv at disponere sin vægt på de givne deck, ville modellen passe mindst ligeså godt som SP-CH.[67] Dette da spilleren ville have mulig-

[67] Bemærk at modellen ser fra en Level 5'ers perspektiv. At give muligheden for at mikse ville udelukkende give et bedre fit på data, men det level der mikser ville være indifferent.

heden for at spille det ene deck med 100 pct. sandsynlighed (og modellen er da identisk med SP-CH), men også muligheden for at mikse lidt imellem de to (eller flere) deck. Dette ville medføre, at MDE kunne udnytte et sådant 'frit' miks til at minimere den samlede afstand fra begge deck til data, fremfor kun ét af dem som i SP-CH. I vores tilfælde ville Highlander Hunter altså kunne sænkes fra de 16,6 pct., som SP-CH prædikerer, imens Murloc Paladin hæves fra 5,2, hvorfor MP-CH i dette tilfælde vil fitte bedre på data.[68]

### 9.4.3 Standard Beta CH-Model

Vi anvender en Standard beta-CH model, som vi kalder SB-CH, til at estimere fordelingen af kognitive niveauer blandt Hearthstonespillere. Metodisk benytter vi samme fremgangsmåde som SP-CH modellen beskrevet i afsnit 9.4.1, dog hvor agentens forestilling om populationen ikke estimeres via. en poissonfordeling, men derimod en betafordeling. Betafordelingen spænder fra 0 til 1 og beskrives ved to parametre $\alpha$ og $\beta$, der angiver formen på fordelingen. Denne tillader os da at beskrive en mere fleksibel fordeling end poisson-CH modellerne. Som beskrevet i afsnit 6.2.3 kan betafordelingen tillade os at antage, at der er en høj andel af spillere af level 0, men ligeledes afbillede en relativt stor andel af højere kognitive niveauer.

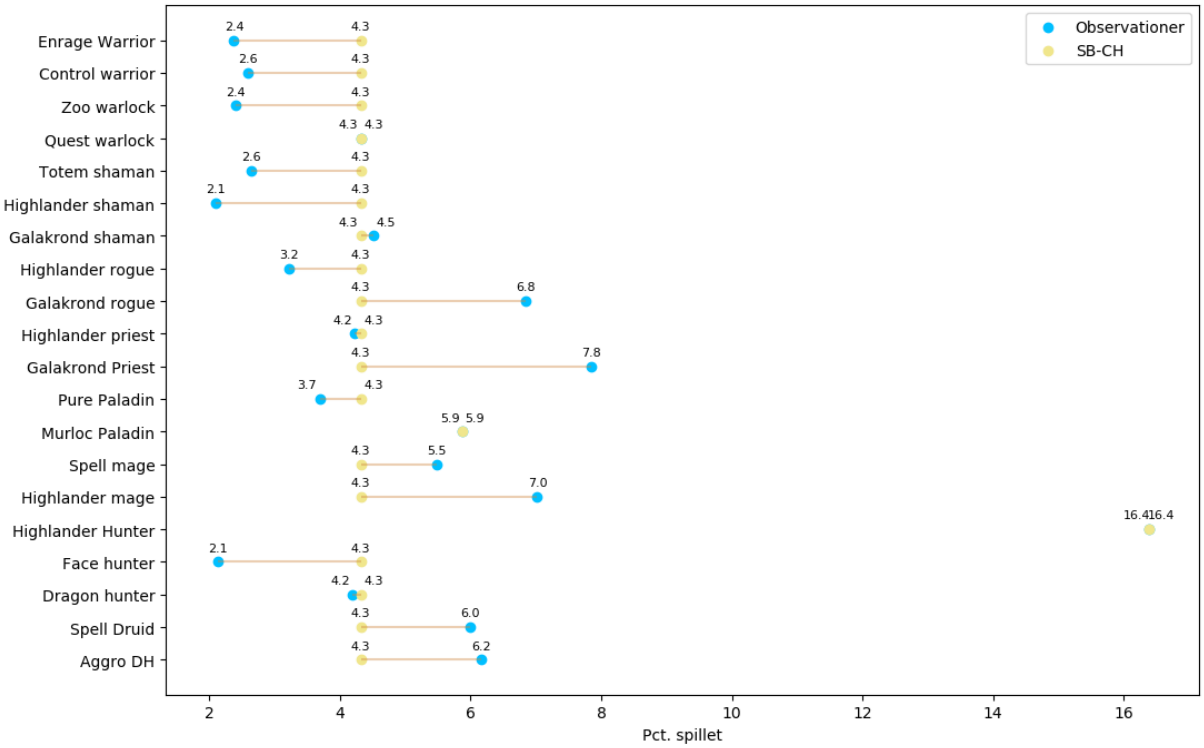
De optimale parameterværdier estimeres til $\alpha = 0.206$ og $\beta = 2.416$,[69] svarende til 86 pct. spillere af level 0 som set i tabel 9.6. Det bemærkes dog, at SB-CH modellen estimerer 8,58 svagt kognitivt tænkende individer og 5,02 stærkt kognitivt tænkende. Modellen prædikerer deckandelene $\mathbf{p}_{SB-CH}^T$=(0.0432, 0.0432, 0.0432, 0.0432, 0.164, 0.0432, 0.0432, 0.0587, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432), som er afbilledet i figur 9.6.

Overordnet udgør individer af level 0 en høj andel af populationen i begge poisson modellerne samt SB-CH modellen, og resultatet er derfor umiddelbart robust over for forskellige fordelinger. Det interessante ved SB-CH modellen er dog, at andelen af svagt kognitive individer er mindsket, medens andelen af stærkt kognitive er steget til ikke længere at være en forsvindende lille andel af den samlede befolkning. Det er stadig en lavere andel relativt til lignende litteratur, der estimerer, at omkring 30 pct. af alle individer er kogni-

[68] Den ville altså gøre afstanden fra prædiktion til data ens for de to deck.
[69] Bemærk at vores gæt fra afsnit 6.2.3 på, at $\alpha < 1$ og $\beta < 1$ ville være optimalt, altså ikke bliver bekræftet.

tivt stærke (eksempelvis ved $\tau = 1,14$ som Camerer et al. [2004] estimerer). Dette tyder altså, på trods af betafordelingens egenskaber og den dertil øgede fleksibilitet, at kognitive hierarkimodeller ikke formår at afbillede spillernes beslutningstagen i avancerede spil i lige så høj grad som for simple spil.

**Figur 9.6:** Standard Beta CH-model og Observeret Data



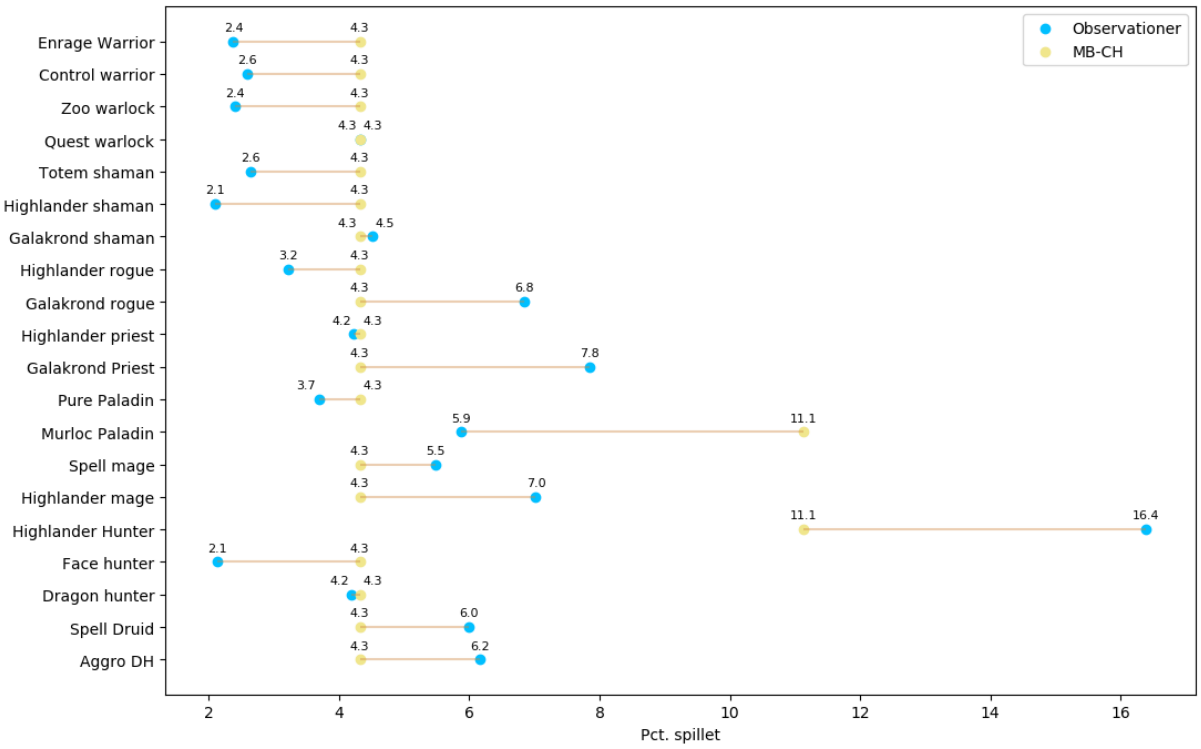

Vi bemærker, at vores SB-CH model faktisk outperformer SP-CH modellen marginalt, baseret på $l^2$-normen på 58 og $R^2$ på 0,70, som ses i tabel 9.5. Den højere andel af kognitivt stærke agenter, der følger en betafordeling, forklarer altså data mindst ligeså godt som den anderkendte model, Camerer et al. [2004] opstiller. Camerer et al. [2004] påpeger, at poisson-CH modellen klarer sig godt sammenlignet med andre fordelinger, og at der ikke opleves signifikante forskelle på poissonfordelingen og andre fordelinger. Netop disse resultater korroborerer vi.

### 9.4.4 Modificeret Beta CH-Model

Ligesom for poisson-CH modellen bestemmes en beta-CH model, hvor beslutningskriterierne modificeres. Den modificerede beta-CH model mikser ligeligt blandt strategier, hvis payoff er inden for 1 pct. point fra hinanden; vi kalder modellen MB-CH.

De optimale parameterværdier for MB-CH modellen er $\alpha = 0,281$ og $\beta = 3,215$. Dette svarer til, at der fortsat er 86,38 pct. ustrategiske spillere, men andelen af svagt kognitive er steget til 9,59 pct., og andelen af stærkt kognitive er faldet til 3,94 pct. Fordelingen fremgår af tabel 9.6. MB-CH modellen prædikerer deckandelene $\mathbf{P}^T_{MB-CH}$=(0.0432, 0.0432, 0.0432, 0.0432, 0.111, 0.0432, 0.0432, 0.111, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432), som er afbilledet i Figur 9.7:

**Figur 9.7:** Modificeret Beta CH-model og Observeret Data




Endvidere fås $l^2$ på 114 og $R^2$ på 0,42, som sammenlignes med de andre modeller i tabel 9.5. Det bemærkes, at MB-CH og MP-CH modellerne næsten er identiske, medens SP-CH og SB-CH ligeledes er så.

SB-CH og MB-CH modellerne angiver begge en identisk andel af ustrategiske individer blandt Hearthstonespillere. Beta-CH modellerne estimerer dog op til en 3 gange så høj andel af kognitivt stærke individer end poissonfordelingen, hvilket kan tilskrives den ekstra parameter, der kan forme fordelingen.

### 9.4.5 Nash-CH model

I dette underafsnit fremgår resultaterne af analysen af vores Nash-CH model, som er beskrevet i afsnit 8. Nash-CH modellen afviger fra de tidligere kognitive hierarkimodeller, da den bestemmes af frie parametre fremfor en specifik fordeling. Når Nash-ligevægten indgår som et trin i det kognitive hierarki, kan denne type model ikke længere betragtes udelukkende som en iterativ proces for en spiller af level k. Dette da spillere, der spiller den *naive* Nash-ligevægt, ikke påvirkes af, hvad de andre kognitive niveauer spiller.

Vi betragter denne model, da det er muligt, at en andel af de individer, der spiller Hearthstone, spiller en form for *naiv* Nash-ligevægt, uanset hvad andre spiller. Strategien i Nash-ligevægten fremgår af afsnit 9.2 og spilles uafhængigt af fordelingen af spillere af lavere kognitive niveauer i populationen.

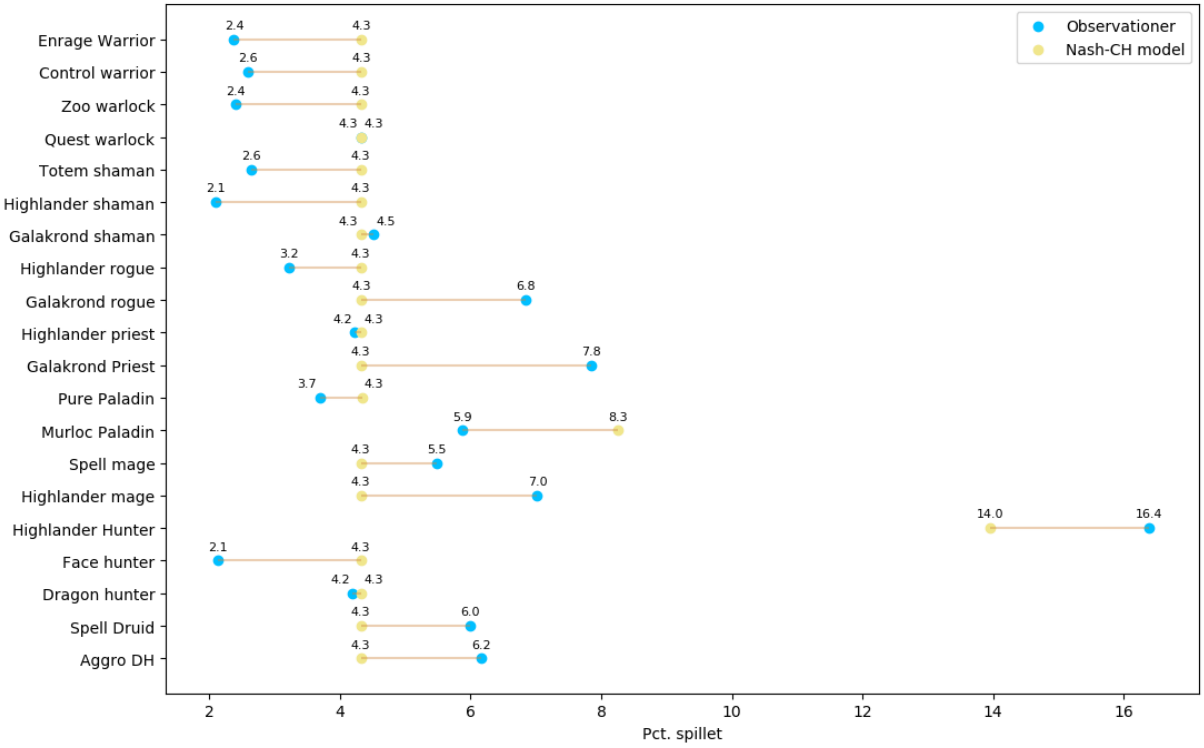
Vi betragter fortsat modellen ud fra en level 5 spiller's synspunkt. Fordelingen af levels under hende er da level 0, level 1, level 2, level 3 og Nash-ligevægten. Fordelingen af Nash-CH modellen bestemmes med frie parametre, hvor vi finder, at andelen af spillere, der agerer ustrategisk, fortsat udgør en høj andel på 86,38 pct. af den samlede population. Fordelingen kan ses af tabel 9.6. Modellen prædikerer deckandelene $\mathbf{P}^T_{Nash-CH}$=(0.0432, 0.0432, 0.0432, 0.0432, 0.164, 0.0432, 0.0432, 0.0587, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432, 0.0432),[70] som visualiseres i figur 9.8.

Resultatet er bemærkelsesværdigt af flere årsager. Først og fremmest indikerer det, at selv med frie parametre estimerer Nash-CH modellen en andel af ustrategiske individer, der er tæt på identisk med poisson og beta-CH modellerne. Ligeledes estimerer Nash-CH modellen, at størstedelen af strategiske individer udgøres af spillere af level 1 (12,06 pct.). Dette resultat er i overensstemmelse med vores poisson og beta-CH modeller, der angiver gennemsnitlige andele på henholdsvis omtrent 12,5 og 9 pct. af populationen. Nash-CH modellen estimerer endvidere, at der er flere spillere af level 3 end level 2, hvilket ved første øjekast indikerer, at de individer, der udviser højere niveau af kognitiv iteration end level 1, udviser en dyb strategisk tankegang. Antallet af L2'ere udgør dog en forsvindende lille andel af den samlede population, hvilket medfører, at en spiller af level 3 benytter samme

[70] Bemærk at disse andele er marginalt tæt på dem vi ser for SB-CH.

strategi som level 2. Vi kan derfor ikke adskille spillere af level 2 og level 3 i Nash-CH modellen.

**Figur 9.8:** Nash-CH og Observeret Data




Vi finder, at Nash-CH modellen estimerer, at der ikke findes individer i vores data, der spiller Nash-ligevægten. Crawford et al. [2013] og Stahl og Wilson [1995] finder dog i de simple spil, at en lille andel af populationen spiller Nash-ligevægten, selvom den ikke er optimal, når andre spillere afviger fra den. Disse naive nash-spillere er således ikke til stede i vores setup. Resultatet understøtter dog konceptet bag CH-modellerne, hvor Nash-ligevægten ikke er optimal at spille, da antagelsen om rationelle forventninger ikke opretholdes, som beskrevet i afsnit 8.

Vi får et $l^2$ på 58,34 og $R^2$ på 0,70, som sammenlignes med de andre modeller i tabel 9.5. Det bemærkes, at Nash-CH og MP-CH modellernes fit på data er identisk, hvorfor benyttelsen af frie vægte altså ikke bidrager til højere forklaringsgrad end betafordelingen.

Vi finder altså et konsistent resultat af andelen af ustrategiske spillere i alle vores modeller. Dette kunne til dels skyldes, at vi betragter data fra Hearthstonespillere på samtlige

kompetitive niveauer. Disse spillere fordeler sig imidlertid over seks ligaer. Vores antagelse om, at man "spiller for at vinde" er ikke holdbar, hvis størstedelen af dataindberetterne ikke spiller på et niveau, hvor de er strategisk 'gode nok', eller kompetitive nok, til at foretage optimale beslutninger. I afsnit 9.5 tester vi, hvorvidt individer fra højt kompetetive miljøer (high ability) spilleres adfærd divergerer fra individer fra lavt kompetetive (low ability). Vi tester ligeledes, hvorvidt resultaterne er konsistente over tid, ved at estimere den optimale levelfordeling for andre observerede datasæt.

## 9.5 Robusthedstjek

### 9.5.1 High- og low ability

CH-modellerne i afsnit 9.4 estimerer en konsistent andel ustrategiske spillere på ca. 86 pct. Vi er interesserede i at vurdere, hvorvidt en spillers ranking har en inflydelse på deres kognitive niveauer. Hearthstone består af seks ligaer, som afspejler, hvor god en spiller man er. De er rangeret i stigende rækkefølge: Bronze, Silver, Gold, Platinum, Diamond og Legend, hvor Legend er bedst, og Bronze er værst. Vi betragter de tre øverste ligaer i datasættet som high ability spillere. Det må antages, at spillere i denne gruppering i højere grad spiller for at vinde, er mere dedikerede til spillet, har adgang til flere midler ift. deckkonstruktion[71] samt holder sig mere opdateret på ændringer og optimale strategier. Ligeledes defineres low ability spillere som dem, der findes i de tre nederste ligaer. Disse faktorer gør, at vi kan opstille en hypotese om, at andelen af kognitivt tænkende individer vil være højere, når man betragter high ability, end når man betragter low ability.

Vi estimerer ved SP-CH modellen et optimalt $\tau = 0.171$ for high ability og $\tau = 0.06$ for low ability. Det indikerer en stigning i andelen af strategiske individer for de bedre spillere, hvor levelfordelingen kan ses af tabel 9.7. Vi finder, at den samlede fordeling af ustrategiske spillere for low ability er på 94,15, mens den er på 84,29 for high ability. Hypotesen, om at high ability består af mere strategiske spillere, kan da bekræftes. Trods at high ability har mere end dobbelt så mange strategiske individer, repræsenterer de stadig en mindre andel af den samlede population. Det skal nævnes, at faldet i andelen af individer med kognitive kompetencer, kan skyldes at der ikke er lige så meget data fra low ability spillere.

[71] Således at de er mere sandsynlige til at tilpasse sig.

Vi foretager ligeledes en analyse af high ability og low ability for vores SB-CH model. For high ability estimerer vi $\alpha = 0.247$ og $\beta = 2.47$, svarende til 84,04 pct. ustrategiske spillere. Estimationen af low ability giver parametrene $\alpha = 0,093$ og $\beta = 2,65$, hvor fordelingen af populationen for de kognitive niveauer kan ses i tabel 9.7, og tabel 9.9 nedenfor sammenligner parameterestimaterne. Det bemærkes, at SB-CH modellen estimerer tre gange så mange kognitivt stærke individer for high ability, end SP-CH gør. Andelen i SB-CH er 5,89 pct. af populationen mod 1,3 pct. i SP-CH.

Overordnet set viser SP-CH og SB-CH modellerne lignende tendenser for de to undergrupper high- og low ability. High ability består af en væsentligt højere andel kognitive tænkende individer, hvor low ability indeholder en betydelig større andel af ustrategiske. Vi finder ca. 10 pct. points flere strategiske individer i high ability end i low, for både SP-CH og SB-CH modellerne.

**Tabel 9.9:** Parameterværdier på tværs af liga

| Model | SP-CH | SB-CH | |
|---|---|---|---|
| | $\tau$ | $\alpha$ | $\beta$ |
| Baseline | 0,141 | 0,206 | 2,416 |
| *High ability* | 0,171 | 0,247 | 2,470 |
| *Low ability* | 0,063 | 0,093 | 2,636 |

Vi finder, at spillerne i high ability kun klarer sig marginalt bedre rent strategisk end baseline. Dette er i stil med vores argument i afsnit 5.1, hvor vi påpeger, at det hovedsageligt er spillere, der har opbygget en moderat erfaring og ønsker at forbedre sig, der vælger at bidrage med data. Resultatet indikerer, at størstedelen af de individer, der er i datasættet, er high ability, hvorfor at resultaterne ville være endnu mere markante, hvis man havde mere data fra spillere i lave ligaer.

### 9.5.2 Tidskonsistens

For at sikre, at estimaterne af $\tau$ samt $\alpha$ og $\beta$ er konsistente over tid, betragter vi observerede deckandele i de forgangne uger, inden det datasæt vi har benyttet i resten af afhandlingen. Vores baseline datasæt er trukket fra den sidste uge i sæsonen, og vi benytter datasæt fra hhv. en, to og tre uger før sæsonafslutningen til at se, hvorvidt vores

parameterestimater er konsistente.

Som nævnt i afsnit 5 tager vi udgangspunkt i data fra ViciousSyndicate [2020] med brug af rapport #169. I tabel 9.10 sammenlignes andelen af hhv. stærkt kognitive individer (Level $\geq 2$), svagt kognitive (L1) samt ustrategiske (L0) fra rapport #166, #167, #168 og #169 (baseline) i vores standard SP-CH og SB-CH modeller.

**Tabel 9.10:** Andelen af kognitivt stærke spillere afhængigt af rapportnummer og model

| | Andel af population (pct.) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Rapport #166 | | Rapport #167 | | Rapport #168 | | Rapport #169 (Baseline) | |
| | SP-CH | SB-CH | SP-CH | SB-CH | SP-CH | SB-CH | SP-CH | SB-CH |
| Level 0 | 84,29 | 84,13 | 95,09 | 95,28 | 85,15 | 85,20 | 86,87 | 86,38 |
| Level 1 | 14,40 | 8,67 | 4,78 | 4,21 | 13,69 | 9,82 | 12,22 | 8,58 |
| Level $\geq 2$ | 1,30 | 7,18 | 0,12 | 0,50 | 1,16 | 4,97 | 0,90 | 5,02 |

Det ses, at resultaterne fra rapport #167 varierer betydeligt ift. de andre. Andelen af kognitivt stærke individer (eksklusiv rapport #167) varierer mellem 0.9 og 1.3 pct. i SP-CH og mellem 4.97 og 7.18 pct. i SB-CH. Dette svarer til et sving i $\tau$ mellem 0.14 i vores baseline og 0.17 i rapport #166. Andelen af svagt kognitive varierer mellem 12,22 og 14,4 pct. for SP-CH modellen og mellem 8,58 og 9,82 i SB-CH. Dette medfører det konsistente resultat, at andelen af ustrategiske individer varierer mellem 84,29 og 86,87 for SP-CH og mellem 84,13 og 86,38 for SB-CH modellen. Resultaterne er altså meget konsistente i estimeringen af levelfordelingen pånær rapport #167, som har $\tau = 0.05$ (SP-CH). Vi vil ikke kommentere på de modificerede modeller eller Nash-CH, men resultaterne er ligeledes konsistente her, hvor der generelt også findes marginalt højere værdier for $\tau$ end baseline.

Rapport #167 afviger altså markant fra de andre uger, men kan muligvis forklares ved, at Activision Blizzard foretog en opdatering på spillet i denne uge, som påvirkede winrate-matricen $\mathbf{A}$ (tabel 8.1). Som argumenteret for i afsnit 5 antager vi, at der går noget tid, før der indfindes en ligevægt af, hvad de enkelte kognitive niveauer spiller. Denne ændring på winrate-matricen tyder på at have givet et stød til ligevægten, som forårsager, at der forekommer ca. 9 pct. flere ustrategiske spillere. Tilvænningen til den nye ligevægt tyder imidlertid på at være indfundet allerede i rapport #168, hvor andelen af ustrategiske spillere er faldet til ca. 85 pct. af den samlede population. Andelen af ustrategiske individer, som ses i tabel 9.10, er visualiseret i figur A.5 i appendiks, hvor stigningen i rapport #167

ses tydeligt.

Det tyder, på trods af rapport #167, at vores resultater for andelen af kognitive niveauer er konsistente over tid. Endvidere bekræftes vores antagelser om niveauforskelle i forskellige ligaer.

I afsnit 10 vil vi diskutere mulige årsager til, at resultaterne divergerer, fra det man finder i anden litteratur, samt videre betragte potentieller usikkerheder, der kan have en større indflydelse på estimaterne for vores Level-K og CH-modeller. Afsnit 10 afklarer endvidere de potentielle målefejl, der er nævnt i afsnit 5.1, deres potentielle effekt på analysens resultater, samt giver foreslag til videre forskning inden for feltet.

# 10 Diskussion

Vi finder i afsnit 9, at mange Hearthstonespillere i overvejende grad spiller tilfældigt, mens en lav andel af populationen foretager ét eller flere trin i overvejelsen af deres strategivalg. Vores resultater er konsistente over tid, og vi finder, at high ability spillere foretager markant flere kognitive iterationer end low ability. Vi finder, at et sted mellem 0.9 pct. og 5.02 pct. af populationen er spillere af level $\geq 2$, mens Camerer et al. [2004] finder ca. 30 pct. Dette tyder umiddelbart på, at agenter i avancerede spil altså ikke laver lige så mange kognitive iterationer, som de ville i simple spil. Med afsæt i de potentielle fejlkilder, der er nævnt i afsnit 5.1, vil vi i dette afsnit gennemgå mulige forklaringer på de resultater, som vi finder i afsnit 9.

Det faktum, at vores modeller ikke giver de samme resultater som oversigten i Camerer et al. [2004], skyldes formentlig ikke, at vores modeller er fejlagtige, da de alle giver et relativt konsistent estimat på den optimale levelfordeling i populationen. Hvis vores modeller ikke estimerede en konsistent levelfordeling blandt populationen, burde fokus rettes mod vores setup for modellerne. Da dette imidlertid ikke er tilfældet, rettes vores fokus mod at forklare, hvorfor vi oplever en høj andel af ustrategiske individer sammenlignet med lignende litteratur.

### Begrænsninger ved modelkriterierne

Afvigelsen kunne potentielt skyldes, at spilleren ikke formår at lave lige så mange udregninger og kognitive step i dette setup, som ved simple $3 \times 3$ matrixspil. Vores $20 \times 20$ matrix gør det ekstremt kompliceret for agenten manuelt at skulle beregne optimale strategier for levels $k \geq 2$. Hjemmesider såsom ViciousSyndicate [2020] og HSReplay [2020] kan dog benyttes af spillere til hurtigt at danne sig et overblik over de optimale L1 strategier (ved blot at vælge den strategi med det højeste gennemsnitlige payoff). Dette kan muligvis forklare, hvorfor at der netop er relativt mange individer, der spiller "Highlander Hunter", og derved klassificeres som L1.

Vi har igennem denne afhandling antaget, at alle spillerne har muligheden for at vælge mellem alle 20 deck. Dette er imidlertid ikke nødvendigvis korrekt, som beskrevet i afsnit 5.1.2. I realiteten vedhæftes der en omkostning for spilleren, når hun skal ændre sin strategi. En strategiændring kræver nemlig, at man skal tilegne sig de nødvendige kort, der kræves for at spille strategien. Denne restriktion er dog dobbeltsidet, da det også umiddelbart vil styrke en antagelse om, at spilleren vil foretage en grundig strategisk beslutning, inden hun vælger sin strategi, da der er en *entry-fee* for at benytte det deck, der indgår i hendes optimale strategi. Man kan endvidere forestille sig, at når spillere *har* betalt for et deck, så vil de være mere loyale til denne strategi, selvom den muligvis ikke er optimal.

Kahneman et al. [1990] tester denne såkaldte *Endowment Effect* og finder, at forsøgsdeltagerne tilsyneladende foretager færre handler, end de burde, og forfatterne forklarer dette ved, at agenterne tillægger deres objekter en form for affektionsværdi. Netop denne effekt kan forklare, hvorfor visse agenter i vores datasæt konsistent benytter deck, der ikke er optimale, selvom det er muligt at købe nye og sælge gamle deck.[72] På trods af at de fleste spillere har råd til at købe mere end et deck, koster det dem fortsat in-game valuta (som man også kan købe for virkelighedens valuta). Den umiddelbare marginale fordel ved at købe et nyt deck, opvejer måske ikke omkostningerne for det. Udover dette er der også en forskel i prisen for de enkelte decks, så på trods af at et deck muligvis har et lavere forventet payoff end et andet, kan det give mening for en spiller at anvende det

[72] Det er muligt at sælge sit deck til mellem 12,5 og 25 pct. af købsværdien.

billigere deck i sin strategi.

Endvidere, som beskrevet i afsnit 4, udkommer der flere gange årligt nye sæsoner, hvor spillerne skal revurdere deres deckvalg. Dette skaber et grundlag for langsigtede strategiske overvejelser. Hvis en spiller vælger ikke at købe et deck i en sæson, kan det give dem en fordel i den næste sæson, hvor de har flere midler til rådighed. Der kan altså forekomme en form for investering i fremtiden ved at bibeholde ens nuværende decks, hvilket kan bidrage til, at en andel af spillerne ikke når at opdatere deres deckporteføljе i vores datasæt. Dette medfører, at der vil være en naturlig forekomst af strategier, der ikke anses som optimale.

En mulig forklaring, på at vi finder en høj andel ustrategiske spillere i data, kunne være, at de endnu ikke har lært at optimere deres strategivalg i den pågældende sæson. Selvom vi betragter den sidste uge i en sæson, er det muligt, at den avancerede setting ikke tillader, at alle individer når at lære at spille optimalt. Dette er i overensstemmelse med teorien om *reinforcement learning*, som eks. Littman [1994] beskriver, hvor at spillere prøver at forbedre deres spil ved at balancere de to motiver: exploration og exploitation. Spillerne afprøver muligvis forskellige deck for at finde deres winrate. Herefter vil de benytte denne viden til at opstille en optimal strategi. I læringsprocessen vil de dog fremgå som L0'ere i data.

Vi har i afhandlingen argumenteret for, at spilleren optimerer sit strategivalg ud fra én parameter: At vinde ved at have det højest gennemsnitlige payoff. Der kan imidlertid være flere aspekter, der spiller ind, når agenten tager sit valg. Forskellige strategier varierer betydeligt i deres kampvarigheder, f.eks. spænder den gennemsnitlige kampvarighed på strategierne fra 4,8 minutter til 12,9 minutter ifølge HSReplay [2020].[73] Hvis vi beholder antagelsen om, at forbrugeren fortsat maksimerer sit payoff, men inkorporerede at nogle deck spilles hurtigere end andre i sin payoff-matrix, vil det muligvis give et bedre indblik i folks beslutningsgrundlag. Eksempelvis, hvis vi betragter en spiller af L1, som spiller strategien Highlander Hunter med den gennemsnitlige winrate på 54,44 pct. og en gennemsnitlig kampvarighed på 7,1 minutter, vil hun opnå et *effektivt payoff*, ved at spille en

[73] Disse tider er ikke for deck inkluderet i analysen her, da der ikke findes historisk data for dette.

time, på 460.[74] Hvis vi i stedet antager, at individ af L1 ville foretrække at spille strategien Face Hunter, der har en winrate på 51,99, hvor den gennemsnitlige kampvarighed er 6,1 minutter, ville Face Hunter have et effektivt payoff på 511, hvis det spilles i én time. Derfor vil en payoff-matrix, hvor vi i stedet betragter de effektive payoffs, muligvis kunne ændre vores resultater markant.

I forlængelse af ovenstående er der muligvis en anden årsag til diskrepansen mellem vores modellers output og observeret data, som kan skyldes antagelsen om, at spillere udelukkende spiller for at vinde. Hvis vi udelukkende betragtede individer i de øverste ligaer, samt turneringsdata, så ville denne antagelse sandsynligvis være overholdt. En andel af spillerbasen vil dog muligvis foretrække at spille en bestemt strategi, som udløser en individuel glæde. Ligesom at man kan inkorporere tidspræferencer i sin beslutningstagen, kan man inkorporere denne glæde. Man kunne opstille nyttefunktioner, der er positivt afhængige af payoff ved at vinde, samt et individuelt led afhængigt af personlige præferencer. Et eksempel kunne være ligning (35) nedenfor, der beskriver forbrugerens nytte givet ved det *effektive* payoff samt individuelle præferencer:

$$u_i = \tau^j \cdot d^j + \epsilon_i^j \tag{35}$$

hvor $\tau^j$[75] angiver antal kampe per time, $d^j$ det gennemsnitlige payoff for deck $j$ og $\epsilon_i^j$ den personlige nytte for spiller $i$ ved at spille deck $j$. Denne funktion opfanger det grundlæggende payoff ved at spille et deck, men skaleres med kampvarigheden for at opnå det *effektive* payoff. Disse mulige udvidelser vil være interessante at kigge på i fremtidig forskning, hvor en blanding af hhv. traditionel nytteoptimering og beslutningskriteriererne i kognitive hierarkier måske kan give et mere præcist indblik i, hvordan individer agerer i denne form for avancerede spil.

## Begrænsninger i data

En udfordring i vores analyse er, at selvom vores datasæt er sammensat af over 150.000 individuelle kampe, kan det potentielt betragtes som bestående af kun én observation for hhv. winrates og frekvenser. Det er derfor umuligt at analysere de enkelte spilleres stra-

[74] Hvis vi antager at der udelukkende spilles mod en uniform fordeling af alle strategier.
[75] Hvor $\tau^j = \frac{60}{\text{Kampvarighed}^j}$ angiver antallet af kampe per time for deck $j$.

tegivalg, og hvilke parametre der påvirker disse. Det er imidlertid stadig muligt at se de overordnede tendenser, som den samlede population udviser, men resultaterne er sårbare over for uforklarede svingninger. Som nævnt i afsnit 7.1 betyder dette, at vi ikke observerer nogen varians i winrate-matricen. Denne matrix er reelt dannet af en masse individuelle spilleres bidrag af, hvilke deck de har benyttet, og om de har vundet eller tabt deres kamp. Vores data består imidlertid kun af den gennemsnitlige winrate for de givne deck. Dette medfører, at vi ikke kan lave bootstrapping på data, da vi ikke kan trække enkelte kampe ud og benytte disse til at lave fiktive winrate-matricer. Den winrate-matrix fremvist i tabel 8.1 stammer fra ViciousSyndicate [2020], mens at HSReplay [2020] ligeledes opstiller en winrate-matrix for de samme deck. Denne kan ses for samme uge i appendiks B.3, hvor det ses, at denne er tæt på identisk med den, vi benytter. Sammenlignede man alle winrate-matricer tilgængelige på nettet, kunne man finde et estimat på usikkerheden ved disse. Givet at de to winrate-matricer er tæt på identiske, bør de generelle tendenser observeret i data være retvisende, mens udsving i observationen dog ikke kan identificeres. Hvis vi havde adgang til data fra hvert enkelt individ og deres strategivalg, ville vi kunne aggregere disse valg og opnå samme resultat som i afsnit 9. Det ville endvidere give os muligheden for at opdele i mindre samples og teste, hvor konsistente resultaterne er. Dette ville være ækvivalent til det, bootstrapping simulerer. Vi approksimerer denne metode med observationerne, når vi analyserer rapporterne fra andre uger i sæsonen.

Vores datasæt er også sårbart over for selektionsbias. Selektionsbias er en systematisk fejlkilde i et datasæt, der forekommer af et ikke-repræsentativt sample, som skævrider andelen af specifikke typer i data. Denne problematik kan også betragtes som bortfaldsproblematikken, nævnt i afsnit 5.1.1, og kan mere specifikt overføres til volunteering bias, da samtlige individer i vores datasæt har valgt at benytte tilføjelsesprogrammet. Det er sandsynligt, at man skal have over gennemsnitlig interesse i at forbedre sig i Hearthstone, før man downloader dette tredjepartsprogram. I afsnit 9.5 finder vi, at langt størstedelen af datasættet består af high ability spillere, hvilket tyder på, at der er problemer med repræsentativiteten i data.

Vi har valgt ikke at inddrage et stokastisk led i beslutningskriterierne for spillerne, hvilket i vores specifikationer ville svare til, at når en spiller skal vælge sit deck inden en kamp, så

vil hun $n$ pct. af gangene komme til at ryste en smule på sin hånd og vælge det forkerte deck. Selvom introduktionen af et stokastisk led muligvis vil medføre en højere forklaringsgrad i modellen. Et stokastisk led ville tillade MDE at antage færre L0'ere, da en del af de ustrategiske observationer, vi betragter, da kunne forklares af dette led. Spillernes best-response funktion i CH-modellerne ville dog ændres, da de nu, alt andet lige, ville tro, at der er større sandsynlighed for at møde et tilfældigt deck. Desto større middelværdi i det stokastiske led, desto mere ville best-response funktionen for alle levels $> 1$ konvergere mod best-response for level 1. Stokastiske led anvendes dog eksempelvist i *quantum response equilibrium* modeller (McKelvey og Palfrey [1995]), hvor spillernes forventninger til modstanderen samt deres beslutningskriterier indeholder fejlled. Camerer et al. [2004] påpeger, at der er også grundlag for at inkorporere det i CH-modeller.

Til optimering af parameterestimationen i MDE anvender vi en vægtningsmatrix, der er ligeligt fordelt på samtlige strategier. Vi argumenterer i afsnit 7 for, at vægtningsmatricen derfor vil medføre minimal bias i vores data. En grundlæggende antagelse i denne analyse er, at data for vores winrates er konvergeret til den sande værdi på baggrund af det høje antal kampe. Det ses dog i frekvenstabel 8.2, at nogle strategier spilles med en langt højere andel end andre. Antallet af kampe spillet i individuelle matchups mellem to deck varierer mellem 56 og 1796. Der er derfor en stor diskrepans i, hvor mange observationer der ligger til grund for winraten mellem to specifikke strategier. Vi kan derfor ikke med sikkerhed sige, at winrate-matricen indeholder de sande winrates for matchups med et lavt antal observationer. Vi kunne tackle dette problem ved at opstille en ny vægtningsmatrix, der, i stedet for at være ligeligt fordelt, er proportionel med antal kampe, som et deck har spillet. Denne fremgangsmåde minder om processen i *Generalized Method of Moments* (GMM), således at vi har fokus på de estimater, hvor den statistiske usikkerhed er lavest.

Med forbehold for de potentielle fejlkilder fremlagt i dette afsnit vil afsnit 11 nedenfor, på baggrund af resultaterne fra afsnit 9, konkludere på forskningspørgsmålet præsenteret i afsnit 1.

# 11 Konklusion

Vi har i dette kandidatspeciale undersøgt, hvordan individer agerer i avancerede spil. Formålet med specialet har været at besvare forskningsspørgsmålet:

*Hvordan agerer individer i avancerede spil, og kan kognitive hierarkimodeller anvendes til at forklare denne adfærd?*

Vi finder empirisk belæg for, at nogle individer udfører en vis form for kognitiv tankegang i deres valg af strategier, selv når de står over for komplekse valg. Vi finder dog, at andelen af disse individer er markant lavere, end hvad anerkendt litteratur finder for simple spil. Specifikt finder vi, afhængigt af modelvalg, en andel mellem 8,58 og 12,97 pct. af populationen, som er svagt kognitive (Level 1), samt en andel mellem 0.9 og 5.02 pct. der er stærkt kognitive (Level $\geq 2$). Andelen af ustrategiske individer varierer derved mellem 86,00 og 87,75 pct. af populationen.

Vi tester i alt syv forskellige modeller, hvoraf én er identisk med den, Camerer et al. [2004] fremstiller. Vi finder, at alle modellerne, på nær MSNE, giver et konsistent resultat for andelen af ustrategiske spillere - også når man tester over flere perioder. Ved at sammenligne modellernes output med observeret data finder vi, at vores Standard beta-CH model forklarer data marginalt bedre end Camerer et al. [2004]'s model, mens begge beskriver adfærden bedre end Level-K modellen, som Costa-Gomes et al. [2001] opstiller. MSNE giver et dårligere fit på data end en baseline model med fuldkommen ustrategiske individer, hvilket understøtter vigtigheden af alternative modeller til at forklare agenters adfærd i komplekse spil.

Vi undersøger, hvorfor vi finder en så høj andel af ustrategiske individer relativt til lignende litteratur. Vi analyserer derfor, hvorvidt spillere fra højere kompetetive miljøer udviser stærkere strategiske overvejelser, samt undersøger hvilke effekter vores antagelser om vægtningsmatricen har for resultaterne. Vi finder, at high ability individer foretager flere kognitive iterationer end low ability. Vi ser en andel på mellem 15,72 og 15,96 pct. af befolkningen, der er enten svagt eller stærkt kognitive for high ability mod mellem 5,47 og 5,87 for low ability. Ændrer man derimod vægtningsgrundlaget, i den Minimum Distance Estimator vi benytter til at finde optimale parametre, ændres resultaterne fra modellen,

der benytter en poissonfordeling, sig ikke. Modellen, der benytter en betafordeling, ændres dog fra at estimere en andel af strategiske spillere på 13,62 pct. til at estimere en andel på 21,88 pct. Dette skyldes sandsynligvis, at betafordelingen er mere fleksibel end poissonfordelingen, da den kan beskrive populationen ud fra to parametre fremfor én.

Vi påpeger i afsnit 10, at mange af de antagelser, vi laver i analysen, kan have effekt på de resultater, vi får. I forlængelse af dette belyser vi mulige udvidelser og aspekter ved modellerne, der kunne være spændende at inkorporere i fremtidig forskning, herunder en nyttefunktion der inddrager tidsmæssige præferencer.

På baggrund af disse resultater må det konkluderes, at individer i høj grad agerer ustrategisk i avancerede spil. Af de modeller som dette speciale tester, konkluderer vi, at kognitive hierarkimodeller forklarer observerede strategivalg bedst. Vi finder empirisk belæg for, at omkring 86 pct. af populationen kun kan forklares som ustrategiske individer, men at modifikationer til modellerne har muligheden for mere præcist at identificere deres strategivalg. Vores speciale kommer da med en klar opfordring til at udforske kognitive hierarkimodeller mere i fremtidige analyser. Dette for at få et bedre indblik i, hvordan individer rent faktisk agerer, når de står over for komplekse beslutninger.

# A Appendiks

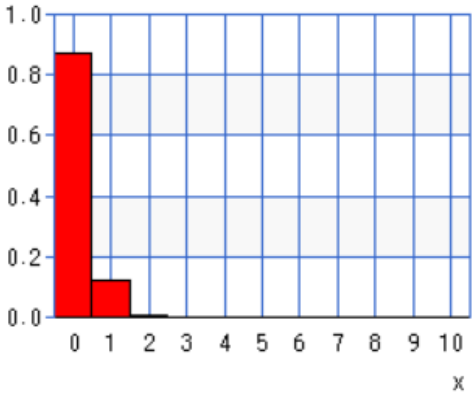
## A.1 Fordelinger

I dette appendiksafsnit vil fordelingen for de enkelte modeller blive fremvist.

### A.1.1 Poissonfordelinger

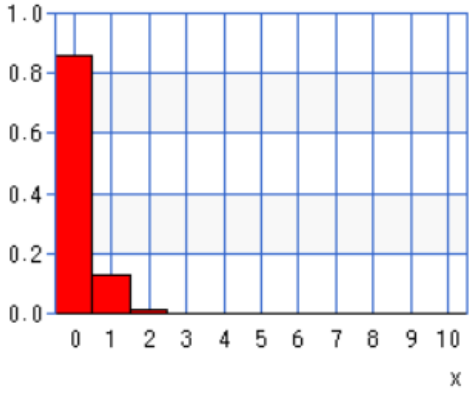
Poissonfordelingen for SP-CH med $\tau = 0,14$ kan ses i figur A.1:

**Figur A.1:** Poissonfordeling med $\tau = 0.14$ (SP-CH)



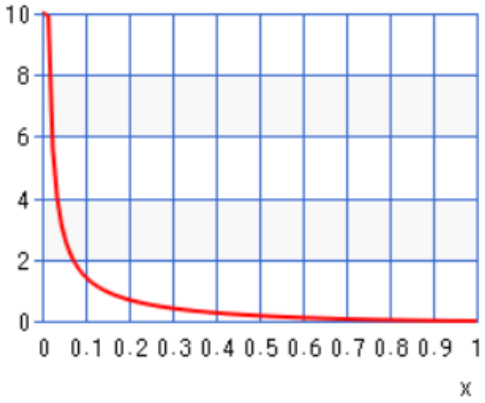

Poissonfordelingen for MP-CH med $\tau = 0,15$ kan ses i figur A.2:

**Figur A.2:** Poissonfordeling med $\tau = 0.15$ (MP-CH)



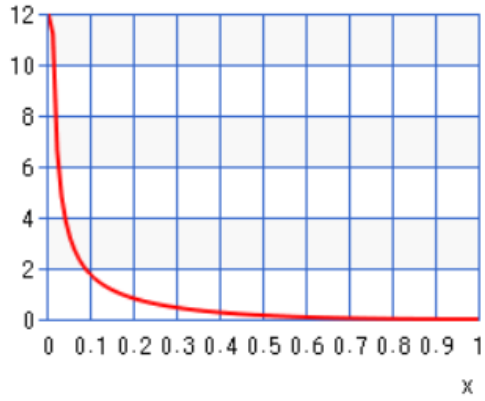
Betafordelinger Betafordelingen for SB-CH med $\alpha = 0,206$ og $\alpha = 2,416$ kan ses i figur A.3:

**Figur A.3:** Betafordeling $\alpha = 0,206$ og $\alpha = 2,416$ (SB-CH)




Betafordelingen for MB-CH med $\alpha = 0,281$ og $\alpha = 3,215$ kan ses i figur A.4:

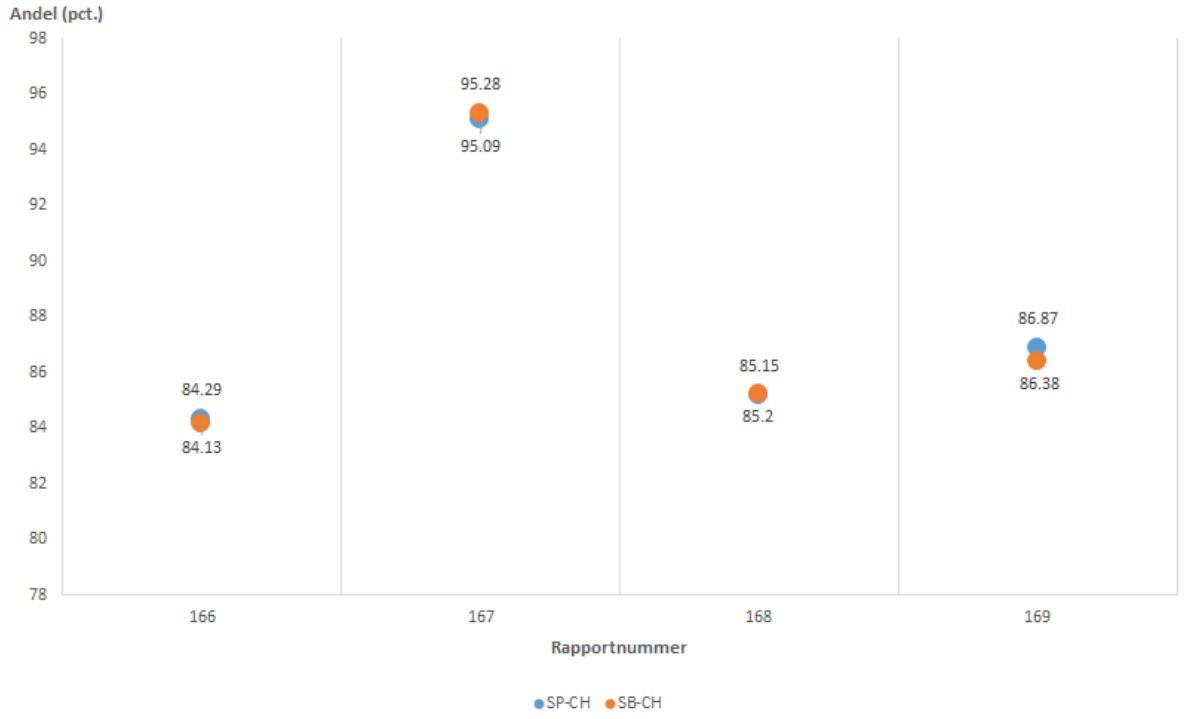
**Figur A.4:** Betafordeling $\alpha = 0,281$ og $\alpha = 3,215$ (MB-CH)



## A.2 Andre figurer og tabeller

Andelen af ustrategiske individer i SP-CH og SB-CH for hhv. rapport #166, #167, #168 og #169 kan ses i figur A.5 nedenfor:

**Figur A.5:** Andel af ustrategiske individer over tid



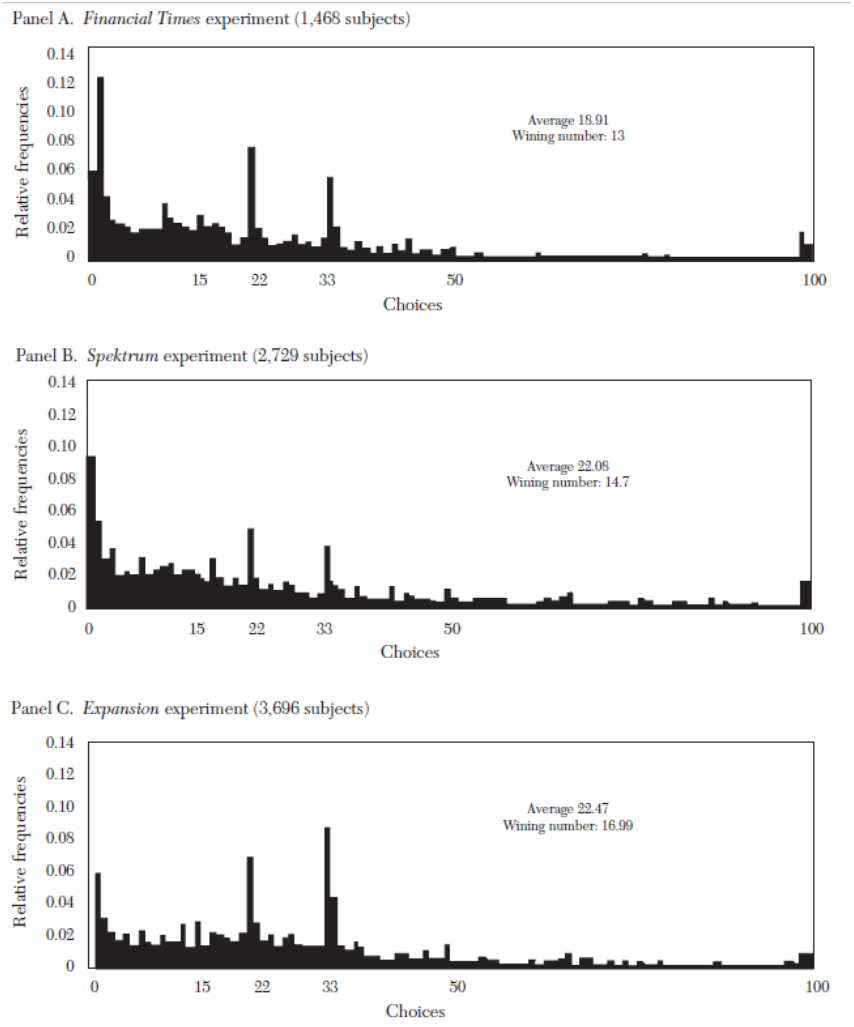

Tabel A.1 nedenfor viser en større udgave af den tabel, som ses i algoritme 1.

**Tabel A.1:** Eksempel på pivoteringstabel

| | $x_1$ | $y_2$ | $y_3$ | ... | $y_n$ | |
|---|---|---|---|---|---|---|
| $y_1$ | $\frac{1}{a_{11}}$ | $\frac{a_{12}}{a_{11}}$ | $\frac{a_{13}}{a_{11}}$ | ... | $\frac{a_{1n}}{a_{11}}$ | $\frac{1}{a_{11}}$ |
| $x_2$ | $-\frac{a_{21}}{a_{11}}$ | $a_{22} - \frac{a_{12}a_{21}}{a_{11}}$ | $a_{23} - \frac{a_{13}a_{21}}{a_{11}}$ | ... | $a_{2n} - \frac{a_{1n}a_{21}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{21}}{a_{11}}$ |
| $x_3$ | $-\frac{a_{31}}{a_{11}}$ | $a_{32} - \frac{a_{12}a_{31}}{a_{11}}$ | $a_{33} - \frac{a_{13}a_{31}}{a_{11}}$ | ... | $a_{3n} - \frac{a_{1n}a_{31}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{31}}{a_{11}}$ |
| ... | ... | ... | ... | ... | ... | $1 - \frac{a_{1(n+1)}a_{k1}}{a_{11}}$ |
| $x_m$ | $-\frac{a_{m1}}{a_{11}}$ | $a_{m2} - \frac{a_{12}a_{m1}}{a_{11}}$ | $a_{m3} - \frac{a_{13}a_{m1}}{a_{11}}$ | ... | $a_{mn} - \frac{a_{1n}a_{m1}}{a_{11}}$ | $1 - \frac{a_{1(n+1)}a_{m1}}{a_{11}}$ |
| | $-\frac{-1}{a_{11}}$ | $-1 - \frac{a_{12}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{13}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{1r}a_{(m+1)1}}{a_{11}}$ | $-1 - \frac{a_{1n}a_{(m+1)1}}{a_{11}}$ | $0 - a_{1(n+1)}a_{(m+1)1}$ a$_{11}$ |

Figur A.6 nedenfor viser resultaterne fra tre beauty contest spil som Crawford et al. [2013] fremviser. Det er værd at bemærke det høje antal gæt på hhv. 22 og 33. (man skulle her gætte halvdelen af gennemsnittet og ikke 2/3.)

**Figur A.6:** Resultater fra beauty contest




Kilde: Crawford [2003]

# B Data

Tabel B.1 viser en større udgave af tabel 8.1.

**Tabel B.1:** Winratesdata

| | Aggro DH | Spell Druid | Dragon hunter | Face hunter | Highlander Hunter | Highlander mage | Spell mage | Murloc Paladin | Pure Paladin | Galakrond Priest | Highlander priest | Galakrond rogue | Highlander rogue | Galakrond shaman | Highlander shaman | Totem shaman | Quest warlock | Zoo warlock | Control warrior | Enrage Warrior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aggro Demon Hunter | **50.00** | 57.54 | 54.59 | 37.72 | 52.90 | 40.30 | 46.23 | 52.97 | 42.37 | 46.19 | 46.31 | 45.43 | 53.1 | 63.2 | 49.65 | 69.47 | 43.60 | 56.43 | 41.95 | 34.98 |
| Spell Druid | 42.46 | **50.00** | 31.35 | 50.03 | 47.21 | 50.69 | 49.48 | 26.99 | 38.42 | 50.53 | 52.42 | 55.61 | 56.70 | 42.16 | 61.14 | 30.15 | 50.99 | 41.80 | 49.21 | 58.52 |
| Dragon hunter | 45.41 | 68.65 | **50.00** | 45.83 | 51.74 | 61.07 | 70.86 | 25.43 | 30.42 | 56.98 | 54.09 | 50.78 | 49.57 | 63.84 | 53.65 | 46.97 | 63.38 | 45.67 | 50.55 | 48.48 |
| Face hunter | 62.28 | 49.97 | 54.17 | **50.00** | 55.00 | 63.01 | 73.64 | 33.36 | 26.36 | 38.86 | 39.58 | 52.31 | 60.7 | 67.74 | 58.59 | 35.62 | 60.03 | 53.05 | 53.79 | 51.76 |
| Highlander Hunter | 47.1 | 52.79 | 48.26 | 45.00 | **50.00** | 61.35 | 60.3 | 47.06 | 48.16 | 63.40 | 62.79 | 53.18 | 58.59 | 61.85 | 67.21 | 55.24 | 52.77 | 48.62 | 54.49 | 50.58 |
| Highlander mage | 59.7 | 49.31 | 38.93 | 36.99 | 38.65 | **50.00** | 49.40 | 58.67 | 61.39 | 54.67 | 56.84 | 54.68 | 57.02 | 52.14 | 65.72 | 50.08 | 51.52 | 60.77 | 46.84 | 60.12 |
| Spell mage | 53.77 | 50.52 | 29.14 | 26.36 | 39.70 | 50.6 | **50.00** | 41.72 | 54.25 | 49.64 | 49.14 | 53.19 | 60.64 | 45.80 | 63.74 | 41.16 | 52.48 | 57.79 | 44.36 | 53.37 |
| Murloc Paladin | 47.03 | 73.01 | 74.57 | 66.64 | 52.94 | 41.33 | 58.28 | **50.00** | 50.72 | 55.17 | 57.32 | 50.77 | 53.33 | 67.84 | 53.66 | 57.16 | 40.48 | 56.52 | 41.97 | 36.31 |
| Pure Paladin | 57.63 | 61.58 | 69.58 | 73.64 | 51.84 | 38.61 | 45.75 | 49.28 | **50.00** | 54.58 | 45.78 | 36.73 | 51.19 | 55.25 | 61.31 | 50.63 | 49.07 | 46.72 | 44.85 | 61.74 |
| Galakrond Priest | 53.81 | 49.47 | 43.02 | 61.14 | 36.6 | 45.33 | 50.36 | 44.83 | 45.42 | **50.00** | 56.03 | 48.13 | 49.03 | 43.67 | 57.29 | 34.28 | 38.47 | 46.47 | 63.04 | 71.17 |
| Highlander priest | 53.69 | 47.58 | 45.91 | 60.42 | 37.21 | 43.16 | 50.86 | 42.68 | 54.22 | 43.97 | **50.00** | 41.24 | 45.34 | 45.45 | 55.79 | 37.78 | 37.36 | 44.39 | 64.89 | 73.16 |
| Galakrond rogue | 54.57 | 44.39 | 49.22 | 47.69 | 46.82 | 45.32 | 46.81 | 49.23 | 63.27 | 51.87 | 58.76 | **50.00** | 56.11 | 57.70 | 60.85 | 52.75 | 45.69 | 49.32 | 44.42 | 46.95 |
| Highlander rogue | 46.90 | 43.30 | 50.43 | 39.30 | 41.41 | 42.98 | 39.36 | 46.67 | 48.81 | 50.97 | 54.66 | 43.89 | **50.00** | 43.75 | 54.75 | 52.79 | 43.11 | 44.79 | 44.51 | 42.39 |
| Galakrond shaman | 36.8 | 57.84 | 36.16 | 32.26 | 38.15 | 47.86 | 54.2 | 32.16 | 44.75 | 56.33 | 54.55 | 42.30 | 56.25 | **50.00** | 55.66 | 36.18 | 41.41 | 40.22 | 65.59 | 58.69 |
| Highlander shaman | 50.35 | 38.86 | 46.35 | 41.41 | 32.79 | 34.28 | 36.26 | 46.34 | 38.69 | 42.71 | 44.21 | 39.15 | 45.25 | 44.34 | **50.00** | 37.86 | 34.22 | 44.34 | 35.94 | 51.55 |
| Totem shaman | 30.53 | 69.85 | 53.03 | 64.38 | 44.76 | 49.92 | 58.84 | 42.84 | 49.37 | 65.72 | 62.22 | 47.25 | 47.21 | 63.82 | 62.14 | **50.00** | 59.18 | 44.17 | 43.57 | 35.48 |
| Quest warlock | 56.40 | 49.01 | 36.62 | 39.97 | 47.23 | 48.48 | 47.52 | 59.52 | 50.93 | 61.53 | 62.64 | 54.31 | 56.89 | 58.59 | 65.78 | 40.82 | **50.00** | 55.02 | 27.89 | 53.62 |
| Zoo warlock | 43.57 | 58.20 | 54.33 | 46.95 | 51.38 | 39.23 | 42.21 | 43.48 | 53.28 | 53.53 | 55.61 | 50.68 | 55.21 | 59.78 | 55.66 | 55.83 | 44.98 | **50.00** | 39.06 | 45.12 |
| Control warrior | 58.05 | 50.79 | 49.45 | 46.21 | 45.51 | 53.16 | 55.64 | 58.03 | 55.15 | 36.96 | 35.11 | 55.58 | 55.49 | 34.41 | 64.06 | 56.43 | 72.11 | 60.94 | **50.00** | 35.73 |
| Enrage Warrior | 65.02 | 41.48 | 51.52 | 48.24 | 49.42 | 39.88 | 46.63 | 63.69 | 38.26 | 28.83 | 26.84 | 53.05 | 57.61 | 41.31 | 48.45 | 64.52 | 46.38 | 54.88 | 64.27 | **50.00** |

Tabel B.2 viser en større udgave af tabel 8.2.

**Tabel B.2:** Frekvensdata

| | Aggro DH | Spell Druid | Dragon hunter | Face hunter | Highlander Hunter | Highlander mage | Spell mage | Murloc Paladin | Pure Paladin | Galakrond Priest | Highlander priest | Galakrond rogue | Highlander rogue | Galakrond shaman | Highlander shaman | Totem shaman | Quest warlock | Zoo warlock | Control warrior | Enrage Warrior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aggro DH | **608** | 527 | 381 | 184 | 1376 | 554 | 453 | 503 | 319 | 719 | 336 | 618 | 260 | 402 | 158 | 258 | 389 | 245 | 262 | 246 |
| Spell Druid | 527 | **560** | 403 | 213 | 1113 | 622 | 482 | 467 | 356 | 785 | 329 | 572 | 263 | 425 | 208 | 245 | 400 | 165 | 214 | 229 |
| Dragon hunter | 381 | 403 | **304** | 154 | 870 | 406 | 362 | 374 | 187 | 482 | 194 | 402 | 201 | 317 | 105 | 198 | 252 | 157 | 170 | 170 |
| Face hunter | 184 | 213 | 154 | **92** | 413 | 233 | 202 | 200 | 114 | 245 | 132 | 234 | 89 | 155 | 56 | 92 | 128 | 81 | 79 | 64 |
| Highlander Hunter | 1376 | 1113 | 870 | 413 | **3652** | 1353 | 1084 | 1282 | 791 | 1796 | 756 | 1536 | 686 | 879 | 409 | 553 | 1056 | 513 | 644 | 567 |
| Highlander mage | 554 | 622 | 406 | 233 | 1353 | **774** | 580 | 548 | 417 | 749 | 575 | 585 | 395 | 424 | 261 | 268 | 435 | 268 | 250 | 231 |
| Spell mage | 453 | 482 | 362 | 202 | 1084 | 580 | **486** | 450 | 313 | 623 | 432 | 497 | 251 | 352 | 206 | 243 | 329 | 197 | 173 | 162 |
| Murloc Paladin | 503 | 467 | 374 | 200 | 1282 | 548 | 450 | **598** | 331 | 696 | 298 | 593 | 228 | 412 | 171 | 238 | 343 | 236 | 214 | 168 |
| Pure Paladin | 319 | 356 | 187 | 114 | 791 | 417 | 313 | 331 | **214** | 427 | 251 | 336 | 168 | 234 | 140 | 165 | 244 | 130 | 153 | 116 |
| Galakrond Priest | 719 | 785 | 482 | 245 | 1796 | 749 | 623 | 696 | 427 | **756** | 426 | 711 | 403 | 537 | 256 | 309 | 471 | 283 | 240 | 291 |
| Highlander priest | 336 | 329 | 194 | 132 | 756 | 575 | 432 | 298 | 251 | 426 | **424** | 395 | 264 | 255 | 184 | 140 | 234 | 150 | 148 | 95 |
| Galakrond rogue | 618 | 572 | 402 | 234 | 1536 | 585 | 497 | 593 | 336 | 711 | 395 | **710** | 321 | 461 | 224 | 275 | 449 | 276 | 274 | 247 |
| Highlander rogue | 260 | 263 | 201 | 89 | 686 | 395 | 251 | 228 | 168 | 403 | 264 | 321 | **170** | 180 | 108 | 108 | 212 | 140 | 161 | 137 |
| Galakrond shaman | 402 | 425 | 317 | 155 | 879 | 424 | 352 | 412 | 234 | 537 | 255 | 461 | 180 | **344** | 133 | 231 | 275 | 161 | 186 | 179 |
| Highlander shaman | 158 | 208 | 105 | 56 | 409 | 261 | 206 | 171 | 140 | 256 | 184 | 224 | 108 | 133 | **74** | 73 | 134 | 84 | 72 | 70 |
| Totem shaman | 258 | 245 | 198 | 92 | 553 | 268 | 243 | 238 | 165 | 309 | 140 | 275 | 108 | 231 | 73 | **78** | 159 | 106 | 99 | 96 |
| Quest warlock | 389 | 400 | 252 | 128 | 1056 | 435 | 329 | 343 | 244 | 471 | 234 | 449 | 212 | 275 | 134 | 159 | **298** | 133 | 190 | 154 |
| Zoo warlock | 245 | 165 | 157 | 81 | 513 | 268 | 197 | 236 | 130 | 283 | 150 | 276 | 140 | 161 | 84 | 106 | 133 | **82** | 97 | 82 |
| Control warrior | 262 | 214 | 170 | 79 | 644 | 250 | 173 | 214 | 153 | 240 | 148 | 274 | 161 | 186 | 72 | 99 | 190 | 97 | **114** | 84 |
| Enrage Warrior | 246 | 229 | 170 | 64 | 567 | 231 | 162 | 168 | 116 | 291 | 95 | 247 | 137 | 179 | 70 | 96 | 154 | 82 | 84 | **116** |

Tabel B.3 viser winrates fra HSReplay.

**Tabel B.3:** Winratedata fra HSReplay

| | Aggro DH | Spell Druid | Dragon hunter | Face hunter | Highlander Hunter | Highlander mage | Spell mage | Murloc Paladin | Pure Paladin | Galakrond Priest | Highlander priest | Galakrond rogue | Highlander rogue | Galakrond shaman | Highlander shaman | Totem shaman | Quest warlock | Zoo warlock | Control warrior | Enrage Warrior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aggro DH | 50.00 | 57.98 | 55.82 | 37.82 | 54.06 | 39.27 | 46.98 | 52.62 | 41.88 | 44.86 | 45.63 | 44.95 | 53.29 | 62.78 | 48.97 | 68.26 | 44.17 | 55.04 | 40.59 | 33.90 |
| Spell Druid | 42.02 | 50.00 | 30.81 | 49.50 | 48.12 | 49.38 | 49.05 | 27.31 | 39.74 | 50.23 | 51.67 | 54.34 | 55.24 | 41.90 | 59.68 | 29.39 | 51.95 | 40.95 | 48.46 | 60.01 |
| Dragon hunter | 44.18 | 69.19 | 50.00 | 44.89 | 51.13 | 62.15 | 70.32 | 25.05 | 30.26 | 56.84 | 53.48 | 51.45 | 48.69 | 62.53 | 52.68 | 46.25 | 64.26 | 45.31 | 50.49 | 49.63 |
| Face hunter | 62.18 | 50.50 | 55.11 | 50.00 | 56.48 | 62.53 | 72.85 | 34.21 | 27.19 | 38.13 | 38.44 | 53.64 | 60.25 | 66.45 | 58.47 | 34.73 | 59.72 | 52.44 | 53.43 | 52.53 |
| Highlander Hunter | 45.94 | 51.88 | 48.87 | 43.52 | 50.00 | 60.47 | 60.19 | 45.77 | 48.69 | 62.03 | 64.28 | 52.39 | 59.59 | 61.21 | 66.29 | 55.49 | 52.37 | 49.67 | 53.54 | 49.57 |
| Highlander mage | 60.73 | 50.62 | 37.85 | 37.47 | 39.53 | 50.00 | 50.81 | 58.41 | 60.89 | 55.15 | 56.40 | 53.18 | 55.80 | 50.67 | 65.17 | 48.78 | 52.90 | 60.00 | 47.04 | 60.65 |
| Spell mage | 53.02 | 50.95 | 29.68 | 27.15 | 39.81 | 49.19 | 50.00 | 40.78 | 53.63 | 49.98 | 49.71 | 53.18 | 60.37 | 44.69 | 62.69 | 40.29 | 51.05 | 57.85 | 43.06 | 52.23 |
| Murloc Paladin | 47.38 | 72.69 | 74.95 | 65.79 | 54.23 | 41.59 | 59.22 | 50.00 | 50.15 | 54.25 | 56.97 | 50.25 | 53.47 | 67.09 | 52.84 | 56.25 | 39.03 | 55.82 | 42.94 | 35.67 |
| Pure Paladin | 58.12 | 60.26 | 69.74 | 72.81 | 51.31 | 39.11 | 46.37 | 49.85 | 50.00 | 54.03 | 45.28 | 36.30 | 50.45 | 54.99 | 60.10 | 50.26 | 47.82 | 45.37 | 46.28 | 61.19 |
| Galakrond Priest | 55.14 | 49.77 | 43.16 | 61.87 | 37.97 | 44.85 | 50.02 | 45.75 | 45.97 | 50.00 | 57.25 | 47.01 | 47.53 | 44.30 | 56.59 | 35.08 | 37.74 | 45.80 | 62.30 | 69.71 |
| Highlander priest | 54.37 | 48.33 | 46.52 | 61.56 | 35.72 | 43.60 | 50.29 | 43.03 | 54.72 | 42.75 | 50.00 | 42.47 | 45.36 | 45.30 | 54.36 | 37.20 | 36.81 | 45.43 | 64.33 | 73.66 |
| Galakrond rogue | 55.05 | 45.66 | 48.55 | 46.36 | 47.61 | 46.82 | 46.82 | 49.75 | 63.70 | 52.99 | 57.53 | 50.00 | 54.83 | 59.09 | 61.71 | 51.39 | 45.49 | 48.58 | 45.55 | 46.34 |
| Highlander rogue | 46.71 | 44.76 | 51.31 | 39.75 | 40.41 | 44.20 | 39.63 | 46.53 | 49.55 | 52.47 | 54.64 | 45.17 | 50.00 | 43.38 | 53.52 | 52.66 | 42.94 | 45.74 | 45.52 | 41.33 |
| Galakrond shaman | 37.22 | 58.10 | 37.47 | 33.55 | 38.79 | 49.33 | 55.31 | 32.91 | 45.01 | 55.70 | 54.70 | 40.91 | 56.62 | 50.00 | 57.12 | 35.89 | 40.08 | 39.46 | 64.40 | 58.20 |
| Highlander shaman | 51.03 | 40.32 | 47.32 | 41.53 | 33.71 | 34.83 | 37.31 | 47.16 | 39.90 | 43.41 | 45.64 | 38.29 | 46.48 | 42.88 | 50.00 | 37.79 | 34.14 | 45.65 | 35.46 | 51.44 |
| Totem shaman | 31.74 | 70.61 | 53.75 | 65.27 | 44.51 | 51.22 | 59.71 | 43.75 | 49.74 | 64.92 | 62.80 | 48.61 | 47.34 | 64.11 | 62.21 | 50.00 | 58.83 | 45.31 | 43.25 | 34.59 |
| Quest warlock | 55.83 | 48.05 | 35.74 | 40.28 | 47.63 | 47.10 | 48.95 | 60.97 | 52.18 | 62.26 | 63.19 | 54.51 | 57.06 | 59.92 | 65.86 | 41.17 | 50.00 | 55.91 | 26.65 | 52.71 |
| Zoo warlock | 44.96 | 59.05 | 54.69 | 47.56 | 50.33 | 40.00 | 42.15 | 44.18 | 54.63 | 54.20 | 54.57 | 51.42 | 54.26 | 60.54 | 54.35 | 54.69 | 44.09 | 50.00 | 37.65 | 45.54 |
| Control warrior | 59.41 | 51.54 | 49.51 | 46.57 | 46.46 | 52.96 | 56.94 | 57.06 | 53.72 | 37.70 | 35.67 | 54.45 | 54.48 | 35.60 | 64.54 | 56.75 | 73.35 | 62.35 | 50.00 | 35.06 |
| Enrage Warrior | 66.10 | 39.99 | 50.37 | 47.47 | 50.43 | 39.35 | 47.77 | 64.33 | 38.81 | 30.29 | 26.34 | 53.66 | 58.67 | 41.80 | 48.56 | 65.41 | 47.29 | 54.46 | 64.94 | 50.00 |

# C Kode

Forneden findes koden som vi har benyttet i Python til udarbejdelsen af specialet. Koden kan findes på https://github.com/madsaloise/speciale2020 og har en kørselstid på ca. 49 minutter.

Koden kan løse alle $n \times n$ matrix-spil i et kognitivt hierarki for ethvert vilkårligt level $> 0$. Koden er blot et udklip af de mest relevante programmer. For en komplet oversigt henvises til vores github.

### Program til at køre al koden

ExecAllFinished.py kører alle vores programmer.

```
#Imports
import numpy as np
import pandas as pd
from scipy.optimize import linprog
from scipy import optimize
import matplotlib.pyplot as plt
import time
from pathlib import Path
#M ler tid
t0= time.process_time()

#S t sti
dirname = 'C:\\speciale2020\\Data'
#Data er i excel
suffix = '.xlsx'

#Navne p  ark
WinrateNavne = ['Winrates_Data_2_169', 'Winrates_Data_166', '
    Winrates_Data_167', 'Winrates_Data_168']
FrekvensNavne = ['Frekvenser_169', 'Frekvenser_166', 'Frekvenser_167',
     'Frekvenser_168' ,'Frekvenser_169_PlatToLegend', '
    Frekvenser_169_UnderPlatinium']

#Importerer DataPrep
from DataPrep import ImportExcelFile
```

```
from DataPrep import ImportFrekvenser
#Syntax:
# ImportExcelFile(Kolonner, R kker, dataframe, Path)
# ImportFrekvenser(Path)
# Det, som man gerne vil gemme fra funktionen angives som 1, de andre
    som 0. Stien angives med R'Sti.xlsx'
# V lger man flere input med 1 vil den bare returnere kolonnenavnene,
     just dont

#Winrates Data
#PathWin = r'C:\Users\Mads\Desktop\Speciale\Kode\Git\Data\
    Winrates_Data_2_169.xlsx'
PathWin = Path(dirname, WinrateNavne[0]).with_suffix(suffix)
#Frekvens Data
#PathFrek = r'C:\Users\Mads\Desktop\Speciale\Kode\Git\Data\
    Frekvenser_169.xlsx'
PathFrek = Path(dirname, FrekvensNavne[0]).with_suffix(suffix)
deck_names = ImportExcelFile(1,0,0, PathWin)
winrates = ImportExcelFile(0,1,0, PathWin)
data = ImportExcelFile(0,0,1, PathWin)
frekvenser = ImportFrekvenser(PathFrek)

#S tter parametre for poisson
tau = 0.1407035175879397
tau_afrund = 0.15075376884422112
tau_levelk = 0.1306532663316583

#Vi betragter level 5
level = 5

#Dominans, Syntax: EliminineringDomStrat(deck, winrates)
from EliminineringDominededeStrat import EliminineringDomStrat
EliminineringDomStrat(deck_names, winrates)

#Importerer Nash
#Syntax: solvemixednash(decks, winrates)
from MixedEquilibriumWinrates import solvemixednash
#Printer
```

```
print("Optimal sammens tning af deck i et mixed-nash equilibrium er: " + str(solvemixednash(deck_names, winrates, 0)) + ". Andre decks spilles med en sandsynlighed p  0.")

#Optimale tau beregnet ved OptLS_Standard
tau = 0.1407035175879397
tau_afrund = 0.15075376884422112
tau_levelk = 0.1306532663316583
level = 5

###/POISSON###
print("POISSONFORDELING:")
print("LEVEL-K:")
#Syntax: levelksolve(decks, winrates, levels), level 0 antages at spille uniformt. For k spillere skrives levels som k-1.
from LevelKModelTeori import levelksolve
print(list(levelksolve(deck_names, winrates, level-1)))
from LevelKModelPoisson import levelksolvepoisson
print("Deckandele for level-k modellen: ")
print(levelksolvepoisson(deck_names, winrates, level, tau_levelk))

#CH Model, syntax: CHSolve(decks, winrates, levels, kommentarer, tau, decksandsynligheder = 1):, level 0 antages at spille uniformt.
#"Kommentarer" skal v re en, hvis man vil se sandsynligheder og payoffs, 0 ellers.
print("CH-MODEL:")
from CHModelAfrund import CHSolveAfrund
from CHModel import CHSolve
from CHModel import player_distributionpoisson

print("Standard Poisson-CH model: ")
print(CHSolve(deck_names, winrates, level, 0, tau, 0))
print(CHSolve(deck_names, winrates, level, 0, tau, 1))
print("tilh rende levelfordeling: ")
print(player_distributionpoisson(tau, level))
print("Afrundet Poisson-CH model: ")
print(CHSolveAfrund(deck_names, winrates, level, 0, tau, 0))
print(CHSolveAfrund(deck_names, winrates, level, 0, tau, 1))
print("tilh rende levelfordeling: ")
```

```
print(player_distributionpoisson(tau_afrund, level))

#Optimerer tau
from LeastSquares import OptLS_Standard
OptLS_Standard(deck_names, winrates, frekvenser, level+1)


###POISSON/###


###/BETA###
print("BETAFORDELING:")
from CHModelBetaDistAfrund import CHSolveBetaAfrund
from CHModelBetaDist import CHSolveBeta
from CHModelBetaDist import player_distribution
from CHModelFreeWeights import CHModelFree
from AlphaBetaOptimizer import f_one
from AlphaBetaOptimizer import f_two
from AlphaBetaOptimizer import f_three
from AlphaBetaOptimizer import f_four
from AlphaBetaOptimizer import f_five
from AlphaBetaOptimizer import f_six
from AlphaBetaOptimizer import f_seven
from BR_Til_Nash_CHMODEL import NashCHModelCH
#from BR_Til_Nash_NashLigev gt import NashCHModelNash
import math

sum_func1 = lambda x: sum(f_one(x[0], x[1], deck_names, winrates,
    frekvenser, level))
sum_func2 = lambda x: sum(f_two(x[0], x[1], deck_names, winrates,
    frekvenser, level))
sum_func3 = lambda x: sum(f_four(x[0], x[1], x[2], x[3], x[4],
    deck_names, winrates, frekvenser))
sum_func4 = lambda x: sum(f_five(x[0], x[1], x[2], x[3], x[4],
    deck_names, winrates, frekvenser))
sum_func6 = lambda x: sum(f_six(x[0], x[1], deck_names, winrates,
    frekvenser, level))
sum_func7 = lambda x: sum(f_seven(x[0], x[1], deck_names, winrates,
    frekvenser, level))
```

```
124
125 initial_guess = [0.5, 0.5]
126 initial_guess2 = [0.2, 0.2, 0.2, 0.2, 0.2]
127
128 sol_case1 = optimize.minimize(sum_func1, initial_guess, method='SLSQP'
    , bounds=[(0,None), (0, None)])
129 sol_case2 = optimize.minimize(sum_func2, initial_guess, method='SLSQP'
    , bounds=[(0,None), (0, None)])
130 sol_case3 = optimize.minimize(sum_func3, initial_guess2, method='SLSQP
    ', bounds=[(0,None), (0, None),(0,None), (0, None), (0,None)])
131 sol_case4 = optimize.minimize(sum_func4, initial_guess2, method='SLSQP
    ', bounds=[(0,None), (0, None),(0,None), (0, None), (0,None)])
132 sol_case6 = optimize.minimize(sum_func6, initial_guess, method='SLSQP'
    , bounds=[(0,None), (0, None)])
133 sol_case7 = optimize.minimize(sum_func7, initial_guess, method='SLSQP'
    , bounds=[(0,None), (0, None)])
134
135 print("Alpha,Beta for Standard CH: ")
136 print(sol_case1['x'])
137 print("Alpha,Beta for Afrundet CH: " )
138 print(sol_case2['x'])
139
140 NormSolDist1 = []
141 count = 0
142 for i in sol_case3['x']:
143     NormSolDist1.append(sol_case3['x'][count]/sum(sol_case3['x']))
144     count += 1
145 print("V gte for Nash-CH model: " )
146 print(NormSolDist1)
147
148 NormSolDist2 = []
149 count = 0
150 for i in sol_case4['x']:
151     NormSolDist2.append(sol_case4['x'][count]/sum(sol_case4['x']))
152     count += 1
153 print("V gte for Standard CH model med frie v gte: ")
154 print(NormSolDist2)
155
156 print("Alpha, Beta for CH model skaleret med frekvenser: ")
```

```
157 print(sol_case6['x'])
158
159 print("Alpha, Beta for CH model v gtet med 1 p  level-k modellens
    prediktioner og 20/12 p  resterende decks")
160 print(sol_case7['x'])
161
162
163 #Optimale alpha og beta bruges til at beregne CH-modellerne
164 print("Deck andele")
165 print("Beta Standard")
166 print(CHSolveBeta(deck_names, winrates, level,sol_case1['x'][0],
    sol_case1['x'][1], 0, MLE = 1))
167 print("tilh rende levelfordeling: ")
168 print(player_distribution(level,sol_case1['x'][0], sol_case1['x'][1]))
169 print("Beta Afrundet")
170 print(CHSolveBetaAfrund(deck_names, winrates, level, sol_case2['x'
    ][0], sol_case2['x'][1], 0, MLE = 1))
171 print("tilh rende levelfordeling: ")
172 print(player_distribution(level,sol_case2['x'][0], sol_case2['x'][1]))
173 print("Nash-CH")
174 print(NashCHModelCH(solvemixednash(deck_names, winrates, 1),
    deck_names, winrates, sol_case3['x'][0], sol_case3['x'][1],
    sol_case3['x'][2], sol_case3['x'][3], sol_case3['x'][4], MLE = 1))
175 print("tilh rende levelfordeling: ")
176 print(NormSolDist1)
177 print("Standard CH med frie parametre")
178 print(CHModelFree(deck_names, winrates, sol_case4['x'][0], sol_case4['
    x'][1], sol_case4['x'][2], sol_case4['x'][3], sol_case4['x'][4],
    MLE = 1))
179 print("tilh rende levelfordeling: ")
180 print(NormSolDist2)
181 print("Standard CH v gtet med frekvenser")
182 print(CHSolveBeta(deck_names, winrates, level,sol_case6['x'][0],
    sol_case6['x'][1], 0, MLE = 1))
183 print("tilh rende levelfordeling: ")
184 print(player_distribution(level,sol_case6['x'][0], sol_case6['x'][1]))
185 print("Standard CH v gtet med 1 p  level-k modellens prediktioner og
     20/12 p  resterende decks")
```

```
print(CHSolveBeta(deck_names, winrates, level,sol_case7['x'][0],
    sol_case7['x'][1], 0, MLE = 1))
print("tilh rende levelfordeling: ")
print(player_distribution(level,sol_case7['x'][0], sol_case7['x'][1]))
###BETA/###

###DUMBELL PLOTS###
from DumbellPlotEnkelte import MixedEqGraphNash
from DumbellPlotEnkelte import MixedEqGraphCHPoissonStandard
from DumbellPlotEnkelte import MixedEqGraphCHPoissonAfrundet
from DumbellPlotEnkelte import MixedEqGraphCHBetaAfrundet
from DumbellPlotEnkelte import MixedEqGraphCHBetaStandard
from DumbellPlotEnkelte import MixedEqLevelK
from DumbellPlotEnkelte import NashCH

MixedEqGraphNash(solvemixednash(deck_names, winrates, 1), frekvenser)
MixedEqGraphCHPoissonStandard(deck_names, CHSolve(deck_names, winrates
    , level, 0, tau, 1), frekvenser)
MixedEqGraphCHPoissonAfrundet(deck_names, CHSolveAfrund(deck_names,
    winrates, level, 0, tau_afrund, 1), frekvenser)
MixedEqGraphCHBetaStandard(deck_names, CHSolveBeta(deck_names,
    winrates, level, sol_case1['x'][0], sol_case1['x'][1], 0, MLE = 1),
     frekvenser)
MixedEqGraphCHBetaAfrundet(deck_names, CHSolveBetaAfrund(deck_names,
    winrates, level, 0.0899432,  0.79928276, 0, MLE = 1), frekvenser)
MixedEqLevelK(deck_names, levelksolvepoisson(deck_names, winrates,
    level, tau_levelk), frekvenser)
NashCH(deck_names, NashCHModelCH(solvemixednash(deck_names, winrates,
    1), deck_names, winrates, 0.8638375736535256, 0.09599785625939583,
    0.03119252040390985, 0.00813884245046459, 0.0008332072327040807,
    MLE = 1), frekvenser)


#from BR_Til_Nash_NashLigev gt import NashCHModelNash
#Tests af andre uger
WinratesList = [Path(dirname, WinrateNavne[1]).with_suffix(suffix),
    Path(dirname, WinrateNavne[2]).with_suffix(suffix), Path(dirname,
    WinrateNavne[3]).with_suffix(suffix)]
```

```
FrekvenserList = [Path(dirname, FrekvensNavne[1]).with_suffix(suffix),
    Path(dirname, FrekvensNavne[2]).with_suffix(suffix), Path(dirname,
    FrekvensNavne[3]).with_suffix(suffix)]
WeekList = [166, 167, 168]
count = 0
initial_guess1 = [0.5, 0.5]
for i in WeekList:
    print(WeekList[count])
    deck_names = ImportExcelFile(1,0,0, WinratesList[count])
    winrates = ImportExcelFile(0,1,0, WinratesList[count])
    frekvenser = ImportFrekvenser(FrekvenserList[count])
    OptLS_Standard(deck_names, winrates, frekvenser, level+1)
    sum_func1 = lambda x: sum(f_one(x[0], x[1], deck_names, winrates,
   frekvenser, level))
    sol_case1 = optimize.minimize(sum_func1, initial_guess1, method='
   SLSQP', bounds=[(0,None), (0, None)])
    print("alpha, beta")
    print(sol_case1['x'])
    print(CHSolveBeta(deck_names, winrates, level,sol_case1['x'][0],
   sol_case1['x'][1], 0, MLE = 1))
    count += 1

#Tests af highability og lowability
WinratesList = [Path(dirname, WinrateNavne[0]).with_suffix(suffix),
   Path(dirname, WinrateNavne[0]).with_suffix(suffix)]
FrekvenserList = [Path(dirname, FrekvensNavne[4]).with_suffix(suffix),
    Path(dirname, FrekvensNavne[5]).with_suffix(suffix)]
RankingList = ["Highability", "Lowability"]
count = 0
initial_guess1 = [0.5, 0.5]
for i in RankingList:
    print(RankingList[count])
    deck_names = ImportExcelFile(1,0,0, WinratesList[count])
    winrates = ImportExcelFile(0,1,0, WinratesList[count])
    frekvenser = ImportFrekvenser(FrekvenserList[count])
    OptLS_Standard(deck_names, winrates, frekvenser, level+1)
    sum_func1 = lambda x: sum(f_one(x[0], x[1], deck_names, winrates,
   frekvenser, level))
```

```
    sol_case1 = optimize.minimize(sum_func1, initial_guess1, method='
    SLSQP', bounds=[(0,None), (0, None)])
    print("alpha, beta")
    print(sol_case1['x'])
    print(CHSolveBeta(deck_names, winrates, level,sol_case1['x'][0],
    sol_case1['x'][1], 0, MLE = 1))
    count += 1

#Skal v re til sidst
t1 = time.process_time() - t0
print("Tid, der er brugt p  at k re koden: ", t1) # CPU seconds
    elapsed (floating point)
print("R^2 og l^2 er beregnet i excel og fremg r derfor ikke af dette
    program.")
plt.show()
```

### Program der forbereder vores datasæt til analysen

DataPrep.py henter datasættet og opstiller det på liste form.

```
#Preparing Data from excel to multiple lists for rows and columns
import numpy as np
import pandas as pd

#Importerer
def ImportExcelFile(decknames, winrates, dataframe, Path):
    df = pd.read_excel (Path)

    column_names = list(df.columns)

    if decknames == 1:
        return column_names
    else:
        if dataframe == 1:
            return df
        else:
            if winrates == 1:
                Row_list =[]
                # Iterate over each row
                for i in range((df.shape[0])):
```

```
                    # the current row denoted by "i"
                    Row_list.append(list(df.iloc[i, :]))
                return Row_list
            else:
                "Set one of the inputs to 1"

def ImportFrekvenser(Path):
    df = pd.read_excel (Path)
    column_names = list(df.columns)
    Row_list =[]
    # Iterate over each row
    for i in range((df.shape[0])):
        # the current row denoted by "i"
        Row_list.append(list(df.iloc[i, :]))
    return Row_list
```

### Program der tjekker for dominerede strategier

EliminieringDomineredeStrat.py tjekker for dominerede strategier.

```
import numpy as np

def EliminieringDomStrat(deck_names, winrates):
    count = 0
    DomineredeStrat = []
    P2Winrates = np.array(winrates)
    for j in range(len(deck_names)-1):
        PlusEn = 0
        for i in winrates[:-1]:
            if all(P2Winrates[count,:] <= P2Winrates[PlusEn,:]) and
   any(P2Winrates[count,:] < P2Winrates[PlusEn,:]) == True:
                DomineredeStrat.append(count)
                print("R kke " + str(count) + " er svagt domineret af
    " + str(PlusEn))
            PlusEn += 1
        count += 1
    if not DomineredeStrat:
        print("Ingen svagt dominerede strategier")
```

### Program der løser Nash-ligevægten

MixedEquilibriumWinrates.py løser Nash-ligevægten.

```
import numpy as np
from scipy.optimize import linprog

def solvemixednash(decks, winrates, GrafData):
    num_decks=len(decks)
    # Lav C
    c = [0 for i in range(num_decks)]
    c.append(1)

    # Beregn payoff
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    # append -1 til brug for  vre  gr nse
    for r in payoffs:
        r.append(-1.0)

    # h jreside-gr nse
    b_ub = [0 for i in range(num_decks)]

    # equality constraint
    ones = [1 for i in range(num_decks)]
    ones.append(0)
    A_eq = [ones]

    # x summerer til 1
    b_eq = [1]

    # x og c must skal v re positive
    bounds = [(0,None) for i in range(num_decks+1)]
    solution = linprog(c, A_ub=payoffs, b_ub=b_ub, A_eq=A_eq, b_eq=
   b_eq, bounds=bounds)

    if GrafData == 0:
        MixedStrat = [(i,j) for i,j in zip(decks,solution['x']) if j
   != 0]
        return MixedStrat
    elif GrafData == 2:
```

```
        MixedStrat = [j for i,j in zip(decks,solution['x'])]
        return MixedStrat
    else:
        return zip(decks,solution['x'])
```

### Program der løser et Level K hierarki

LevelKModelTeori.py løser et level-k hierarki.

```
import numpy as np

def levelksolve(decks, winrates, levels):

    num_decks=len(decks)

    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Inds tter player 0's valg i en liste
    level_0_maxpayoff = max(avg_payoff)
    level_0_index = avg_payoff.index(level_0_maxpayoff)

    maks_index = []
    deckID = [level_0_index]
    for p in range(levels):
        for i in payoffs:
            maks_index.append(payoffs[level_0_index])
        deckID.append(np.argmin(maks_index))
        level_0_index = np.argmin(maks_index)
        maks_index = []
    #Danner en liste med forskellige spilleres valg
    counter=0
    plays = []
    print("I et level-k hierarki har vi f lgende:")
    for i in deckID:
        ilevel_k = counter
```

```
        deckIDcounter = deckID[ilevel_k]
        leveliplay = decks[deckIDcounter]
        plays.append("Level-" + str(counter+1) + " spiller: "+ str(
    leveliplay))
        counter += 1
    return plays
```

### Program der løser SP-CH modellen

CHModel.py løser en kognitiv hierarkimodel med en poissonfordeling.

```
import numpy as np
from math import factorial
from math import exp

def player_distribution(tau, levels):
    def poisson_distribution(tau, levels):
        distribution = (exp(-tau))*(tau**levels)/(factorial(levels))
        return distribution
    fractions = []
    truncated_fractions = []
    for i in range(levels):
        fractions.append(poisson_distribution(tau, i))
    for i in range(levels):
        truncated_fractions.append(fractions[i]/sum(fractions))
    return truncated_fractions
def player_distributionpoisson(tau, levels):
    def poisson_distribution(tau, levels):
        distribution = (exp(-tau))*(tau**levels)/(factorial(levels))
        return distribution
    fractions = []
    truncated_fractions = []
    for i in range(levels):
        fractions.append(poisson_distribution(tau, i))
    for i in range(levels):
        truncated_fractions.append(fractions[i]/sum(fractions))
    return truncated_fractions
#SSH p  1/antallet af decks, hvis det kun er lvl 0. 1 ellers
def player_plays(winrates, level, deckID, indeks_tal):
    if level == 0:
```

```
        prob = 1/len(winrates)
        return prob
    elif indeks_tal == deckID and level > 0:
        return 1
    else:
        return 0
#print(sum(player_distribution(0.1708542713567839,5)[1:]))
def CHSolve(decks, winrates, levels, kommentarer, tau = 0.5, MLE = 0):
    num_decks=len(decks)
    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Lvl1 og op
    deckID = []
    maks_index = []
    payoff_index = []
    deck_prob = []
    for p in range(levels+1):
        if p > 0:
            #Kopierer liste
            A = list.copy(winrates)
            i_list = []
            count1 = 0
            #Beregner sandsynlighed for at du m der et deck betinget
    p   dit level og beliefs omkring de andres levels.
            for i in A:
                temp_dist = 0
                #Summerer ssh for alle levels
                for q in range(p):
                    if q == 0:
                        temp_dist = temp_dist + (1/len(A))*
    player_distribution(tau, p)[q]
                    else:
                        temp_dist = temp_dist + player_distribution(
    tau, p)[q] * player_plays(winrates, q, deckID[q-1], count1)
```

```
                i_list.append(temp_dist)
                count1 += 1
            count2 = 0
            #normaliserer
            for i in i_list:
                deck_prob.append(i_list[count2]/sum(i_list))
                count2 += 1
            maks_index = list.copy(deck_prob)
            #Beregner gennemsnitligt payoff
            for i in A:
                temp_sum = []
                count3 = 0
                for j in i:
                    temp_sum.append(j * maks_index[count3])
                    count3 += 1
                payoff_index.append(sum(temp_sum))
                temp_sum = []
            if kommentarer == 1:
                print("Level-" + str(p) + "--->")
                print("Sandsynligheden for at m de andre decks: " +
str(maks_index))
                print("Payoffs: " + str(payoff_index))
            #Tilf jer deck-indekset til deckID-listen
            deckID.append(payoff_index.index(max(payoff_index)))
            if p == levels:
                return_prob = maks_index
            else:
                #Nulstiller lister
                maks_index = []
                payoff_index = []
                deck_prob = []
    if MLE == 1:
        return return_prob
    else:
        #Danner en liste med forskellige spilleres valg
        counter=0
        plays = []
        print("I en standard CH-model har vi f lgende:")
        for i in deckID:
```

```
            ilevel_k = counter
            deckIDcounter = deckID[ilevel_k]
            leveliplay = decks[deckIDcounter]
            plays.append("Level-" + str(counter+1) + " spiller: "+ str
    (leveliplay))
            counter += 1
        return plays
```

### Program der løser MP-CH modellen

CHModelAfrund.py løser en modificeret kognitiv hierarkimodel med en poissonfordeling.

```
import numpy as np
from math import factorial
from math import exp

def player_distribution(tau, levels):
    def poisson_distribution(tau, levels):
        distribution = (exp(-tau))*(tau**levels)/(factorial(levels))
        return distribution
    fractions = []
    truncated_fractions = []
    for i in range(levels):
        fractions.append(poisson_distribution(tau, i))
    for i in range(levels):
        truncated_fractions.append(fractions[i]/sum(fractions))
    return truncated_fractions
#SSH p  1/antallet af decks, hvis det kun er lvl 0. 1 ellers
def player_plays(winrates, level, deckID, indeks_tal):
    if level == 0:
        prob = 1/len(winrates)
        return prob
    elif indeks_tal == deckID and level > 0:
        return 1
    else:
        return 0

def CHSolveAfrund(decks, winrates, levels, kommentarer, tau = 0.5, MLE
    = 0):
    num_decks=len(decks)
```

```
    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Lvl1 og op
    deckID = []
    maks_index = []
    payoff_index = []
    deck_prob = []
    for p in range(levels+1):
        if p > 0:
            #Kopierer liste
            A = list.copy(winrates)
            i_list = []
            count1 = 0
            #Beregner sandsynlighed for at du m der et deck betinget
p   dit level og beliefs omkring de andres levels.
            for i in A:
                temp_dist = 0
                #Summerer ssh for alle levels
                for q in range(p):
                    if q == 0:
                        temp_dist = temp_dist + (1/len(A))*
player_distribution(tau, p)[q]
                    elif not isinstance(deckID[q-1], list):
                        temp_dist = temp_dist + player_distribution(
tau, p)[q] * player_plays(winrates, q, deckID[q-1], count1)
                    else:
                        count4 = 0
                        for i in deckID[q-1]:
                            temp_dist = temp_dist +
player_distribution(tau, p)[q] * player_plays(winrates, q, deckID[q
-1][count4], count1) / len(deckID[q-1])
                            count4 += 1
                i_list.append(temp_dist)
                count1 += 1
```

```
            count2 = 0
            #normaliserer
            for i in i_list:
                deck_prob.append(i_list[count2]/sum(i_list))
                count2 += 1
            maks_index = list.copy(deck_prob)
            #Beregner gennemsnitligt payoff
            for i in A:
                temp_sum = []
                count3 = 0
                for j in i:
                    temp_sum.append(j * maks_index[count3])
                    count3 += 1
                payoff_index.append(sum(temp_sum))
                temp_sum = []
            if kommentarer == 1:
                print("Level-" + str(p) + "--->")
                print("Sandsynligheden for at m de andre decks: " +
str(maks_index))
                print("Payoffs: " + str(payoff_index))
            #Tilf jer deck-indekset til deckID-listen
            max_payoff = round(max(payoff_index))
            multiple_max = [round(i) for i, j in enumerate(
payoff_index) if round(j) == max_payoff]
            if len(multiple_max) == 1:
                deckID.append(payoff_index.index(max(payoff_index)))
            else:
                deckID.append(multiple_max)
            if p == levels:
                return_prob = maks_index
            else:
                #Nulstiller lister
                maks_index = []
                payoff_index = []
                deck_prob = []
    if MLE == 1:
        return return_prob
    else:
        #Danner en liste med forskellige spilleres valg
```

```
        counter=0
        plays = []
        print("I en afrundet CH-model har vi f lgende:")
        for i in deckID:
            ilevel_k = counter
            deckIDcounter = deckID[ilevel_k]
            if isinstance(deckIDcounter, list):
                counter1 = 0
                temp_list = []
                for j in deckIDcounter:
                    temp_list.append(decks[deckIDcounter[counter1]])
                    counter1 += 1
                plays.append("Level-" + str(counter+1) + " spiller er
   mix af: " + str(temp_list))
            else:
                leveliplay = decks[deckIDcounter]
                plays.append("Level-" + str(counter+1) + " spiller: "+
    str(leveliplay))
            counter += 1
        return plays
```

### Program der løser for det optimale $\tau$ i Level K, SP-CH og MP-CH

LeastSquares.py løser et optimerer $\tau$ i en poisson CH model og level-k hierarki.

```
from scipy import optimize
import numpy as np
from CHModel import CHSolve
from CHModelAfrund import CHSolveAfrund
from LevelKModelPoisson import levelksolvepoisson
import matplotlib.pyplot as plt
import numpy as np
def OptLS_Standard(decks, winrates, Obs_Frequences, levels):
    Afrund_Val = []
    Standard_Val = []
    LevelK_Val = []
    v gtetLevelL_Val = []
    #LevelKDecks = [20/12, 10, 20/12, 10, 10, 20/12, 20/12, 10, 10,
   20/12, 10, 10, 20/12, 20/12, 20/12, 20/12, 20/12, 20/12, 20/12, 10]
    testlist = []
```

```
    count = 0
    '''
    for i in LevelKDecks:
        if count == 4:
            testlist.append(1)
        else:
            testlist.append(0)
        count += 1
    print(testlist)
    '''
    tau_values = np.linspace(0, 2, 200)
    for q in range(levels):
        NumberOfGames = []
        count = 0
        for i in Obs_Frequences:
            NumberOfGames.append(sum(i)+i[count])
            count += 1
        ShareOfGames = []
        for i in NumberOfGames:
            ShareOfGames.append(100 * i / sum(NumberOfGames))
        Sum_Each = []
        Sum_Each_Afrund = []
        Sum_Each_LevelK = []
        #Sum_Each_v gtet_levelk = []
        for i in tau_values:
            Diff_Probs = []
            Diff_probs_Afrund = []
            Diff_Probs_LevelK = []
            #Diff_Probs_v gtet_levelk = []
            count2 = 0
            for j in ShareOfGames:
                Diff_Probs.append((ShareOfGames[count2] - 100* CHSolve
(decks, winrates, q+1, 0, i, 1)[count2])**2)
                Diff_probs_Afrund.append((ShareOfGames[count2] - 100*
CHSolveAfrund(decks, winrates, q+1, 0, i, 1)[count2])**2)
                Diff_Probs_LevelK.append((ShareOfGames[count2] - 100*
levelksolvepoisson(decks, winrates, q+1, i)[count2])**2)
                #Diff_Probs_v gtet_levelk.append(((ShareOfGames[
count2] - 100* CHSolve(decks, winrates, q+1, 0, i, 1)[count2])**2)*
```

```
LevelKDecks[count2])
                count2 += 1
            Sum_Each.append(sum(Diff_Probs))
            Sum_Each_Afrund.append(sum(Diff_probs_Afrund))
            Sum_Each_LevelK.append(sum(Diff_Probs_LevelK))
            #Sum_Each_v gtet_levelk.append(sum(
Diff_Probs_v gtet_levelk))
        Min_Func = Sum_Each.index(min(Sum_Each))
        Min_Func_Afrund = Sum_Each_Afrund.index(min(Sum_Each_Afrund))
        Min_Func_LevelK = Sum_Each_LevelK.index(min(Sum_Each_LevelK))
        #Min_Func_v gtet_levelk = Sum_Each_v gtet_levelk.index(min(
Sum_Each_v gtet_levelk))
        print("Level-" + str(q))
        print("Standard:" + str(min(Sum_Each)) +"med tau p : " + str(
tau_values[Sum_Each.index(min(Sum_Each))]) + ". Afrundet: " + str(
min(Sum_Each_Afrund))+"med tau p : "+str(tau_values[
Sum_Each_Afrund.index(min(Sum_Each_Afrund))]) + "og LevelK: " + str
(min(Sum_Each_LevelK)) + "med en tau v rdi p : " + str(tau_values
[Sum_Each_LevelK.index(min(Sum_Each_LevelK))]))
        #print("CH model v gtet med levelK strategier:" + str(
tau_values[Sum_Each_v gtet_levelk.index(min(
Sum_Each_v gtet_levelk))]))
        #nplevelkdecks = Sum_Each_v gtet_levelk/sum(np.array(
LevelKDecks))
        #V gtSum =  Sum_Each_v gtet_levelk/sum(LevelKDecks)
        fig, ax = plt.subplots(figsize=(12, 8))
        plt.title("Least Squares by tau for level" + str(q))
        plt.scatter(tau_values, Sum_Each, color='navy', alpha=1, label
='Standard')
        plt.scatter(tau_values, Sum_Each_Afrund, color='red', alpha=1,
 label='Afrundet')
        plt.scatter(tau_values, Sum_Each_LevelK, color='yellow', alpha
=1, label='LevelK')
        #plt.scatter(tau_values, nplevelkdecks, color='brown', alpha
=1, label='V gtet med lvlk')
        plt.legend()
        Afrund_Val.append(tau_values[Sum_Each_Afrund.index(min(
Sum_Each_Afrund))])
        Standard_Val.append(tau_values[Sum_Each.index(min(Sum_Each))])
```

```
        LevelK_Val.append(tau_values[Sum_Each_LevelK.index(min(
    Sum_Each_LevelK))])
        #v gtetLevelL_Val.append(tau_values[Sum_Each_v gtet_levelk.
    index(min(Sum_Each_v gtet_levelk))])
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.title("Optimale tau for forskellige levels")
    plt.scatter(range(levels), Standard_Val, color='navy', alpha=1,
    label='Standard')
    plt.scatter(range(levels), Afrund_Val, color='red', alpha=1, label
    ='Afrundet')
    plt.scatter(range(levels), LevelK_Val, color='yellow', alpha=1,
    label='LevelK')
    #plt.scatter(range(levels), v gtetLevelL_Val, color='brown',
    alpha=1, label='V gtet med lvlk')
    plt.legend()

    return Min_Func
```

### Program der løser SB-CH modellen

CHModelBetaDist.py løser en kognitiv hierarkimodel med en betafordeling.

```
import numpy as np
from math import factorial
from math import exp
from scipy.stats import beta
import matplotlib.pyplot as plt
def player_distribution(levels, alpha_val, beta_val):
    def beta_distribution(mean, alpha_val, beta_val):
        distribution = beta.cdf(mean, alpha_val, beta_val)
        return distribution
    fractions = []
    fractions_temp = []
    truncated_fractions = []
    for i in range(levels+1):
        fractions_temp.append(beta_distribution((1+i)/levels,
    alpha_val, beta_val))
    for i in range(len(fractions_temp)-1):
        if i == 0:
            fractions.append(fractions_temp[i])
```

```
        elif i > 0:
            fractions.append(fractions_temp[i]-fractions_temp[i-1])
    for i in range(levels):
        truncated_fractions.append(fractions[i]/sum(fractions))
    return truncated_fractions
def player_plays(winrates, level, deckID, indeks_tal):
    if level == 0:
        prob = 1/len(winrates)
        return prob
    elif indeks_tal == deckID and level > 0:
        return 1
    else:
        return 0
print(player_distribution(5,0.24720334, 2.4701282))
def CHSolveBeta(decks, winrates, levels, alpha_val, beta_val,
   kommentarer, MLE = 0):
    num_decks=len(decks)
    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Lvl1 og op
    deckID = []
    maks_index = []
    payoff_index = []
    deck_prob = []
    for p in range(levels+1):
        if p > 0:
            #Kopierer liste
            A = list.copy(winrates)
            i_list = []
            count1 = 0
            #Beregner sandsynlighed for at du m der et deck betinget
   p   dit level og beliefs omkring de andres levels.
            for i in A:
                temp_dist = 0
```

```
                #Summerer ssh for alle levels
                for q in range(p):
                    if q == 0:
                        temp_dist = temp_dist + (1/len(A))*
player_distribution(p, alpha_val, beta_val)[q]
                    else:
                        temp_dist = temp_dist + player_distribution(p,
 alpha_val, beta_val)[q] * player_plays(winrates, q, deckID[q-1],
count1)
                i_list.append(temp_dist)
                count1 += 1
            count2 = 0
            #normaliserer
            for i in i_list:
                deck_prob.append(i_list[count2]/sum(i_list))
                count2 += 1
            maks_index = list.copy(deck_prob)
            #Beregner gennemsnitligt payoff
            for i in A:
                temp_sum = []
                count3 = 0
                for j in i:
                    temp_sum.append(j * maks_index[count3])
                    count3 += 1
                payoff_index.append(sum(temp_sum))
                temp_sum = []
            if kommentarer == 1:
                print("Level-" + str(p) + "--->")
                print("Sandsynligheden for at m de andre decks: " +
str(maks_index))
                print("Payoffs: " + str(payoff_index))
            #Tilf jer deck-indekset til deckID-listen
            deckID.append(payoff_index.index(max(payoff_index)))
            if p == levels:
                return_prob = maks_index
            else:
                #Nulstiller lister
                maks_index = []
                payoff_index = []
```

```
                deck_prob = []
    if MLE == 1:
        return return_prob
    elif MLE == 2:
        counter=0
        plays = []
        #print("I en CH-model har vi f lgende:")
        for i in deckID:
            ilevel_k = counter
            deckIDcounter = deckID[ilevel_k]
            leveliplay = decks[deckIDcounter]
            counter += 1
        return leveliplay
    else:
        #Danner en liste med forskellige spilleres valg
        counter=0
        plays = []
        #print("I en standard CH-model har vi f lgende:")
        for i in deckID:
            ilevel_k = counter
            deckIDcounter = deckID[ilevel_k]
            leveliplay = decks[deckIDcounter]
            plays.append("Level-" + str(counter+1) + " spiller: "+ str
(leveliplay))
            counter += 1
        return plays
```

### Program der løser MB-CH modellen

CHModelBetaDistAfrund.py løser en modificeret kognitiv hierarkimodel med en betafordeling.

```
import numpy as np
from math import factorial
from math import exp
from scipy.stats import beta
import matplotlib.pyplot as plt
def player_distribution(levels, alpha_val, beta_val):
    def beta_distribution(mean, alpha_val, beta_val):
```

```
        distribution = beta.cdf(mean, alpha_val, beta_val)
        return distribution
    fractions = []
    fractions_temp = []
    truncated_fractions = []
    for i in range(levels+1):
        fractions_temp.append(beta_distribution((1+i)/levels,
   alpha_val, beta_val))
    for i in range(len(fractions_temp)-1):
        if i == 0:
            fractions.append(fractions_temp[i])
        elif i > 0:
            fractions.append(fractions_temp[i]-fractions_temp[i-1])
    for i in range(levels):
        truncated_fractions.append(fractions[i]/sum(fractions))
    return truncated_fractions

#SSH p  1/antallet af decks, hvis det kun er lvl 0. 1 ellers
def player_plays(winrates, level, deckID, indeks_tal):
    if level == 0:
        prob = 1/len(winrates)
        return prob
    elif indeks_tal == deckID and level > 0:
        return 1
    else:
        return 0
def CHSolveBetaAfrund(decks, winrates, levels, alpha_val, beta_val,
   kommentarer, MLE = 0):
    num_decks=len(decks)
    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Lvl1 og op
    deckID = []
    maks_index = []
```

```
    payoff_index = []
    deck_prob = []
    for p in range(levels+1):
        if p > 0:
            #Kopierer liste
            A = list.copy(winrates)
            i_list = []
            count1 = 0
            #Beregner sandsynlighed for at du m der et deck betinget p dit level og beliefs omkring de andres levels.
            for i in A:
                temp_dist = 0
                #Summerer ssh for alle levels
                for q in range(p):
                    if q == 0:
                        temp_dist = temp_dist + (1/len(A))*player_distribution(p, alpha_val, beta_val)[q]
                    elif not isinstance(deckID[q-1], list):
                        temp_dist = temp_dist + player_distribution(p, alpha_val, beta_val)[q] * player_plays(winrates, q, deckID[q-1], count1)
                    else:
                        count4 = 0
                        for i in deckID[q-1]:
                            temp_dist = temp_dist + player_distribution(p, alpha_val, beta_val)[q] * player_plays(winrates, q, deckID[q-1][count4], count1) / len(deckID[q-1])
                            count4 += 1
                i_list.append(temp_dist)
                count1 += 1
            count2 = 0
            #normaliserer
            for i in i_list:
                deck_prob.append(i_list[count2]/sum(i_list))
                count2 += 1
            maks_index = list.copy(deck_prob)
            #Beregner gennemsnitligt payoff
            for i in A:
                temp_sum = []
```

```
                count3 = 0
                for j in i:
                    temp_sum.append(j * maks_index[count3])
                    count3 += 1
                payoff_index.append(sum(temp_sum))
                temp_sum = []
            if kommentarer == 1:
                print("Level-" + str(p) + "--->")
                print("Sandsynligheden for at m de andre decks: " +
str(maks_index))
                print("Payoffs: " + str(payoff_index))
            #Tilf jer deck-indekset til deckID-listen
            max_payoff = round(max(payoff_index))
            multiple_max = [round(i) for i, j in enumerate(
payoff_index) if round(j) == max_payoff]
            if len(multiple_max) == 1:
                deckID.append(payoff_index.index(max(payoff_index)))
            else:
                deckID.append(multiple_max)
            if p == levels:
                return_prob = maks_index
            else:
                #Nulstiller lister
                maks_index = []
                payoff_index = []
                deck_prob = []
    if MLE == 1:
        return return_prob
    else:
        #Danner en liste med forskellige spilleres valg
        counter=0
        plays = []
        #print("I en afrundet CH-model har vi f lgende:")
        for i in deckID:
            ilevel_k = counter
            deckIDcounter = deckID[ilevel_k]
            if isinstance(deckIDcounter, list):
                counter1 = 0
                temp_list = []
```

```
                for j in deckIDcounter:
                    temp_list.append(decks[deckIDcounter[counter1]])
                    counter1 += 1
                plays.append("Level-" + str(counter+1) + " spiller er
    mix af: " + str(temp_list))
            else:
                leveliplay = decks[deckIDcounter]
                plays.append("Level-" + str(counter+1) + " spiller: "+
     str(leveliplay))
            counter += 1
        return plays
```

### Program der løser en Nash-CH model

BR_Til_Nash_CHMODEL.py løser en Nash-CH model.

```
from scipy import optimize
import numpy as np
from CHModelBetaDist import CHSolveBeta
from CHModelBetaDistAfrund import CHSolveBetaAfrund
from MixedEquilibriumWinrates import solvemixednash
from DataPrep import ImportExcelFile
from DataPrep import ImportFrekvenser
from math import factorial
from math import exp
from scipy.stats import beta
from scipy.optimize import linprog
#Syntax:
# ImportExcelFile(Kolonner, R kker, dataframe, Path)
# ImportFrekvenser(Path)
# Det, som man gerne vil gemme fra funktionen angives som 1, de andre
    som 0. Stien angives med R'Sti.xlsx'
# V lger man flere input med 1 vil den bare returnere kolonnenavnene,
    just dont


def player_distribution_frievar(alpha, beta, gamma, delta, epsilon):
    dist = [alpha, beta, gamma, delta, epsilon]
    return dist
def player_plays(winrates, level, deckID, indeks_tal):
```

```
    if level == 0:
        prob = 1/len(winrates)
        return prob
    elif indeks_tal == deckID and level > 0:
        return 1
    else:
        return 0
def CHSolveBeta2(decks, winrates, levels, alpha, beta, gamma, delta,
    epsilon, kommentarer, MLE = 0):
    num_decks=len(decks)
    #Beregner gennemsnitlige payoffs
    payoffs = [[u for u in [(j/50.0)-1 for j in i]] for i in winrates]
    avg_payoff=[]
    for i in payoffs:
        avg_pay = sum(i)/num_decks
        avg_payoff.append(avg_pay)

    #Lvl1 og op
    deckID = []
    maks_index = []
    payoff_index = []
    deck_prob = []
    for p in range(levels+1):
        if p > 0:
            #Kopierer liste
            A = list.copy(winrates)
            i_list = []
            count1 = 0
            #Beregner sandsynlighed for at du m der et deck betinget
    p   dit level og beliefs omkring de andres levels.
            for i in A:
                temp_dist = 0
                #Summerer ssh for alle levels
                for q in range(p):
                    if q == 0:
                        temp_dist = temp_dist + (1/len(A))*
    player_distribution_frievar(alpha, beta, gamma, delta, epsilon)[q]
                    else:
```

```
                        temp_dist = temp_dist +
    player_distribution_frievar(alpha, beta, gamma, delta, epsilon)[q]
    * player_plays(winrates, q, deckID[q-1], count1)
                i_list.append(temp_dist)
                count1 += 1
            count2 = 0
            #normaliserer
            for i in i_list:
                deck_prob.append(i_list[count2]/sum(i_list))
                count2 += 1
            maks_index = list.copy(deck_prob)
            #Beregner gennemsnitligt payoff
            for i in A:
                temp_sum = []
                count3 = 0
                for j in i:
                    temp_sum.append(j * maks_index[count3])
                    count3 += 1
                payoff_index.append(sum(temp_sum))
                temp_sum = []
            if kommentarer == 1:
                print("Level-" + str(p) + "--->")
                print("Sandsynligheden for at m de andre decks: " +
    str(maks_index))
                print("Payoffs: " + str(payoff_index))
            #Tilf jer deck-indekset til deckID-listen
            deckID.append(payoff_index.index(max(payoff_index)))
            if p == levels:
                return_prob = maks_index
            else:
                #Nulstiller lister
                maks_index = []
                payoff_index = []
                deck_prob = []
     if MLE == 1:
         return return_prob
     elif MLE == 2:
         counter=0
         plays = []
```

```
94          for i in deckID:
95              ilevel_k = counter
96              deckIDcounter = deckID[ilevel_k]
97              leveliplay = decks[deckIDcounter]
98              counter += 1
99          return leveliplay
100     else:
101         #Danner en liste med forskellige spilleres valg
102         counter=0
103         plays = []
104         #print("I en standard CH-model har vi f lgende:")
105         for i in deckID:
106             ilevel_k = counter
107             deckIDcounter = deckID[ilevel_k]
108             leveliplay = decks[deckIDcounter]
109             plays.append("Level-" + str(counter+1) + " spiller: "+ str
    (leveliplay))
110             counter += 1
111         return plays
112
113
114 def NashCHModelCH(Our_Nash, deck_names, winrates, alpha, beta, gamma,
    delta, epsilon, MLE = 1):
115     num_decks = len(deck_names)
116     CHLVL0 = [1/num_decks for i in deck_names]
117     CHLVL1 = []
118     CHLVL2 = []
119     CHLVL3 = []
120     count = 0
121     for i in deck_names:
122         if deck_names[count] == deck_names[deck_names.index(
    CHSolveBeta2(deck_names, winrates, 1, alpha, beta, gamma, delta,
    epsilon, 0, MLE = 2))]:
123             CHLVL1.append(1)
124         else:
125             CHLVL1.append(0)
126         count += 1
127     count = 0
128     for i in deck_names:
```

```
129         if deck_names[count] == deck_names[deck_names.index(
    CHSolveBeta2(deck_names, winrates, 2, alpha, beta, gamma, delta,
    epsilon, 0, MLE = 2))]:
130             CHLVL2.append(1)
131         else:
132             CHLVL2.append(0)
133         count += 1
134     count = 0
135     for i in deck_names:
136         if deck_names[count] == deck_names[deck_names.index(
    CHSolveBeta2(deck_names, winrates, 3, alpha, beta, gamma, delta,
    epsilon, 0, MLE = 2))]:
137             CHLVL3.append(1)
138         else:
139             CHLVL3.append(0)
140         count += 1
141     MixedEq = list(Our_Nash)
142     MixedEq_Decks = []
143     NashProbs = []
144     for a, b in MixedEq:
145         MixedEq_Decks.append(a)
146         NashProbs.append(b)
147     dist_probs = player_distribution_frievar(alpha, beta, gamma, delta
    , epsilon)
148     ListProbs = [CHLVL0, CHLVL1, CHLVL2, CHLVL3, NashProbs]
149     CombinedProbs = []
150     count = 0
151     for i in range(len(deck_names)):
152         temp_sum = 0
153         count2 = 0
154         for j in ListProbs:
155             temp_sum = temp_sum + j[count]*dist_probs[count2]
156             count2 += 1
157         CombinedProbs.append(temp_sum)
158         count += 1
159     NormProbs = []
160     for i in range(len(CombinedProbs)):
161         NormProbs.append(CombinedProbs[i]/sum(CombinedProbs))
162     payoff_index = []
```

```
    A = list.copy(winrates)
    #Beregner gennemsnitligt payoff
    for i in A:
        temp_sum = []
        count3 = 0
        for j in i:
            temp_sum.append(j * NormProbs[count3])
            count3 += 1
        payoff_index.append(sum(temp_sum))
        temp_sum = []
    #Tilf jer deck-indekset til deckID-listen
    if MLE == 1:
        return NormProbs
    elif MLE == 0:
        Strategi = deck_names[payoff_index.index(max(payoff_index))]
        return Strategi
```

### Program der opstiller MDE estimatoren for diverse modeller

AlphaBetaOptimizer.py opstiller vores MDE estimator.

```
from scipy import optimize
import numpy as np
from CHModelBetaDist import CHSolveBeta
from CHModelBetaDistAfrund import CHSolveBetaAfrund
import matplotlib.pyplot as plt
import math
from MixedEquilibriumWinrates import solvemixednash
from BR_Til_Nash_CHMODEL import NashCHModelCH
from BR_Til_Nash_NashLigev gt import NashCHModelNash
from CHModelFreeWeights import CHModelFree
def f_one(alpha, beta, deck_names, winrates, frekvenser, levels):
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
```

```
    Diff_Probs = []
    count2 = 0
    for j in ShareOfGames:
        Diff_Probs.append((ShareOfGames[count2] - 100* CHSolveBeta(
   deck_names, winrates, levels, alpha, beta, 0, MLE = 1)[count2])**2)
        count2 += 1
    return Diff_Probs

def f_two(alpha, beta, deck_names, winrates, frekvenser, levels):
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    Diff_Probs = []
    count2 = 0
    for j in ShareOfGames:
        Diff_Probs.append((ShareOfGames[count2] - 100*
   CHSolveBetaAfrund(deck_names, winrates, levels, alpha, beta, 0, MLE
    = 1)[count2])**2)
        count2 += 1
    return Diff_Probs
#Nash-CH model, hvor man l ser Nash-ligev gten
def f_three(alpha, beta, deck_names, winrates, frekvenser, levels):
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    Diff_Probs = []
    count2 = 0
    for j in ShareOfGames:
```

```
55          Diff_Probs.append((ShareOfGames[count2] - 100* NashCHModelNash
    (solvemixednash(deck_names, winrates, 1), deck_names, winrates,
    alpha, beta, 0)[count2])**2)
56          count2 += 1
57      return Diff_Probs
58  #Nash
59  def f_four(alpha, beta, gamma, delta, epsilon, deck_names, winrates,
    frekvenser):
60      NumberOfGames = []
61      count = 0
62      for i in frekvenser:
63          NumberOfGames.append(sum(i)+i[count])
64          count += 1
65      ShareOfGames = []
66      for i in NumberOfGames:
67          ShareOfGames.append(100 * i / sum(NumberOfGames))
68      Diff_Probs = []
69      count2 = 0
70      for j in ShareOfGames:
71          Diff_Probs.append((ShareOfGames[count2] - 100* NashCHModelCH(
    solvemixednash(deck_names, winrates, 1), deck_names, winrates,
    alpha, beta, gamma, delta, epsilon, MLE = 1)[count2])**2)
72          count2 += 1
73      return Diff_Probs
74
75  #Friev gte normal CH
76  def f_five(alpha, beta, gamma, delta, epsilon, deck_names, winrates,
    frekvenser):
77      NumberOfGames = []
78      count = 0
79      for i in frekvenser:
80          NumberOfGames.append(sum(i)+i[count])
81          count += 1
82      ShareOfGames = []
83      for i in NumberOfGames:
84          ShareOfGames.append(100 * i / sum(NumberOfGames))
85      Diff_Probs = []
86      count2 = 0
87      for j in ShareOfGames:
```

```
        Diff_Probs.append((ShareOfGames[count2] - 100* CHModelFree(deck_names, winrates, alpha, beta, gamma, delta, epsilon, MLE = 1)[count2])**2)
        count2 += 1
    return Diff_Probs

#Skaleret med frekvenser
def f_six(alpha, beta, deck_names, winrates, frekvenser, levels):
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    Diff_Probs = []
    count2 = 0
    for j in ShareOfGames:
        Diff_Probs.append(((ShareOfGames[count2] - 100* CHSolveBeta(deck_names, winrates, levels, alpha, beta, 0, MLE = 1)[count2])**2)*ShareOfGames[count2])
        count2 += 1
    return Diff_Probs


#Custom v gte p  decks
def f_seven(alpha, beta, deck_names, winrates, frekvenser, levels):
    LevelKDecks = [20/12, 10, 20/12, 10, 10, 20/12, 20/12, 10, 10, 20/12, 10, 10, 20/12, 20/12, 20/12, 20/12, 20/12, 20/12, 20/12, 10]
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    Diff_Probs = []
```

```
    count2 = 0
    for j in ShareOfGames:
        Diff_Probs.append(((ShareOfGames[count2] - 100* CHSolveBeta(
    deck_names, winrates, levels, alpha, beta, 0, MLE = 1)[count2])**2)
    *LevelKDecks[count2])
        count2 += 1
    return Diff_Probs
```

### Program der laver dumbbell plots af strategisandsynlighederne

DumbellPlotEnkelte.py laver dumbbell plots af strategisandsynlighederne.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def format(l):
    return "["+", ".join(["%.2f" % x for x in l])+"]"

def MixedEqGraphNash(Our_Nash, frekvenser):
    #Splitter tuple
    MixedEq = list(Our_Nash)
    MixedEq_Decks = []
    MixedEq_Winrates = []
    for a, b in MixedEq:
        MixedEq_Decks.append(a)
        MixedEq_Winrates.append(b*100)
    #Omdanner frekvensdata til andele
    NumberOfGames = []
    count = 0
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    CombinedList = []
    for i in range(len(MixedEq_Decks)):
        CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
    MixedEq_Winrates[i]])
```

```
    dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
Observationer", "Nash"])
    my_range = range(1, len(dfGraph.index)+1)
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
', alpha=1, label='Observationer')
    plt.scatter(dfGraph['Nash'], my_range, color='khaki', alpha=1,
label='Nash')
    plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
dfGraph['Nash'], color='peru', alpha = 0.4)
    plt.legend()
    textcordy = 7
    textcordx = 0
    for i, txt in enumerate(ShareOfGames):
        if 0 < ShareOfGames[i] - MixedEq_Winrates[i] < 1:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx+8,
textcordy),ha='center',fontsize=8)
        elif -1 < ShareOfGames[i] - MixedEq_Winrates[i] < 0:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx-8,
textcordy),ha='center',fontsize=8)
        else:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
),ha='center',fontsize=8)
    for i, txt in enumerate(MixedEq_Winrates):
        if 0 < MixedEq_Winrates[i] - ShareOfGames[i] < 1:
            ax.annotate("{:.1f}".format(txt), (MixedEq_Winrates[i],
my_range[i]),textcoords='offset points',xytext=(textcordx+8,
textcordy),ha='center',fontsize=8)
        elif -1 < MixedEq_Winrates[i] - ShareOfGames[i] < 0:
            ax.annotate("{:.1f}".format(txt), (MixedEq_Winrates[i],
my_range[i]),textcoords='offset points',xytext=(textcordx-8,
textcordy),ha='center',fontsize=8)
        else:
            ax.annotate("{:.1f}".format(txt), (MixedEq_Winrates[i],
my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
),ha='center',fontsize=8)
```

```
    #Titel og Akser
    plt.yticks(my_range, dfGraph['Decks'])
    plt.xlabel('Pct. spillet')
    plt.ylabel('Deck')

def MixedEqGraphCHPoissonStandard(MixedEq_Decks, CHInput, frekvenser):

    Standard_CH = [i*100 for i in CHInput]
    count = 0
    NumberOfGames = []
    for i in frekvenser:
        NumberOfGames.append(sum(i)+i[count])
        count += 1
    ShareOfGames = []
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    CombinedList = []
    for i in range(len(MixedEq_Decks)):
        CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
    Standard_CH[i]])
    dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
    Observationer", "CH-model"])
    my_range = range(1, len(dfGraph.index)+1)
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
    dfGraph['CH-model'], color='peru', alpha = 0.4)
    plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
    ', alpha=1, label='Observationer')
    plt.scatter(dfGraph['CH-model'], my_range, color='khaki', alpha=1,
     label='SP-CH')
    plt.legend()
    textcordy = 7
    textcordx = 0
    for i, txt in enumerate(ShareOfGames):
        if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
        elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
```

```
83              ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
84          else:
85              ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
86      for i, txt in enumerate(Standard_CH):
87          if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
88              ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
89          elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
90              ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
91          else:
92              ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
93      #Titel og Akser
94      plt.yticks(my_range, dfGraph['Decks'])
95      plt.xlabel('Pct. spillet')
96      plt.ylabel('Deck')
97
98  def MixedEqGraphCHPoissonAfrundet(MixedEq_Decks, CHInput, frekvenser):
99
100     Standard_CH = [i*100 for i in CHInput]
101     count = 0
102     NumberOfGames = []
103     for i in frekvenser:
104         NumberOfGames.append(sum(i)+i[count])
105         count += 1
106     ShareOfGames = []
107     for i in NumberOfGames:
108         ShareOfGames.append(100 * i / sum(NumberOfGames))
109     CombinedList = []
110     for i in range(len(MixedEq_Decks)):
```

```
111         CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
    Standard_CH[i]])
112     dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
    Observationer", "CH-model"])
113     my_range = range(1, len(dfGraph.index)+1)
114     fig, ax = plt.subplots(figsize=(12, 8))
115     plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
    dfGraph['CH-model'], color='peru', alpha = 0.4)
116     plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
    ', alpha=1, label='Observationer')
117     plt.scatter(dfGraph['CH-model'], my_range, color='khaki', alpha=1,
     label='MP-CH')
118     plt.legend()
119     textcordy = 7
120     textcordx = 0
121     for i, txt in enumerate(ShareOfGames):
122         if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
123             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
124         elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
125             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
126         else:
127             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
128     for i, txt in enumerate(Standard_CH):
129         if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
130             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
131         elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
132             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
133         else:
```

```
134             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
135     #Titel og Akser
136     plt.yticks(my_range, dfGraph['Decks'])
137     plt.xlabel('Pct. spillet')
138     plt.ylabel('Deck')
139
140 def MixedEqGraphCHBetaAfrundet(MixedEq_Decks, CHInput, frekvenser):
141
142     Standard_CH = [i*100 for i in CHInput]
143     count = 0
144     NumberOfGames = []
145     for i in frekvenser:
146         NumberOfGames.append(sum(i)+i[count])
147         count += 1
148     ShareOfGames = []
149     for i in NumberOfGames:
150         ShareOfGames.append(100 * i / sum(NumberOfGames))
151     CombinedList = []
152     for i in range(len(MixedEq_Decks)):
153         CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
    Standard_CH[i]])
154     dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
    Observationer", "CH-model"])
155     my_range = range(1, len(dfGraph.index)+1)
156     fig, ax = plt.subplots(figsize=(12, 8))
157     plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
    dfGraph['CH-model'], color='peru', alpha = 0.4)
158     plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
    ', alpha=1, label='Observationer')
159     plt.scatter(dfGraph['CH-model'], my_range, color='khaki', alpha=1,
     label='MB-CH')
160     plt.legend()
161     textcordy = 7
162     textcordx = 0
163     for i, txt in enumerate(ShareOfGames):
164         if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
```

```
165             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
166         elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
167             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
168         else:
169             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
170     for i, txt in enumerate(Standard_CH):
171         if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
172             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
173         elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
174             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
175         else:
176             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
177     #Titel og Akser
178     plt.yticks(my_range, dfGraph['Decks'])
179     plt.xlabel('Pct. spillet')
180     plt.ylabel('Deck')
181
182 def MixedEqGraphCHBetaStandard(MixedEq_Decks, CHInput, frekvenser):
183
184     Standard_CH = [i*100 for i in CHInput]
185     count = 0
186     NumberOfGames = []
187     for i in frekvenser:
188         NumberOfGames.append(sum(i)+i[count])
189         count += 1
190     ShareOfGames = []
191     for i in NumberOfGames:
```

```
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    CombinedList = []
    for i in range(len(MixedEq_Decks)):
        CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
Standard_CH[i]])
    dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
Observationer", "CH-model"])
    my_range = range(1, len(dfGraph.index)+1)
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
dfGraph['CH-model'], color='peru', alpha = 0.4)
    plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
', alpha=1, label='Observationer')
    plt.scatter(dfGraph['CH-model'], my_range, color='khaki', alpha=1,
     label='SB-CH')
    plt.legend()
    textcordy = 7
    textcordx = 0
    for i, txt in enumerate(ShareOfGames):
        if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx+8,
textcordy),ha='center',fontsize=8)
        elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx-8,
textcordy),ha='center',fontsize=8)
        else:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
),ha='center',fontsize=8)
    for i, txt in enumerate(Standard_CH):
        if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
            ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
my_range[i]),textcoords='offset points',xytext=(textcordx+8,
textcordy),ha='center',fontsize=8)
        elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
            ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
my_range[i]),textcoords='offset points',xytext=(textcordx-8,
```

```
    textcordy),ha='center',fontsize=8)
217         else:
218             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
219     #Titel og Akser
220     plt.yticks(my_range, dfGraph['Decks'])
221     plt.xlabel('Pct. spillet')
222     plt.ylabel('Deck')
223 
224 def MixedEqLevelK(MixedEq_Decks, CHInput, frekvenser):
225 
226     Standard_CH = [i*100 for i in CHInput]
227     count = 0
228     NumberOfGames = []
229     for i in frekvenser:
230         NumberOfGames.append(sum(i)+i[count])
231         count += 1
232     ShareOfGames = []
233     for i in NumberOfGames:
234         ShareOfGames.append(100 * i / sum(NumberOfGames))
235     CombinedList = []
236     for i in range(len(MixedEq_Decks)):
237         CombinedList.append([MixedEq_Decks[i], ShareOfGames[i],
    Standard_CH[i]])
238     dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","
    Observationer", "Level-K"])
239     my_range = range(1, len(dfGraph.index)+1)
240     fig, ax = plt.subplots(figsize=(12, 8))
241     plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax =
    dfGraph['Level-K'], color='peru', alpha = 0.4)
242     plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue
    ', alpha=1, label='Observationer')
243     plt.scatter(dfGraph['Level-K'], my_range, color='khaki', alpha=1,
    label='Level-K Model')
244     plt.legend()
245     textcordy = 7
246     textcordx = 0
247     for i, txt in enumerate(ShareOfGames):
```

```
248         if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
249             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
250         elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
251             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
252         else:
253             ax.annotate("{:.1f}".format(txt), (ShareOfGames[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
254     for i, txt in enumerate(Standard_CH):
255         if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
256             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx+8,
    textcordy),ha='center',fontsize=8)
257         elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
258             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
259         else:
260             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
261     #Titel og Akser
262     plt.yticks(my_range, dfGraph['Decks'])
263     plt.xlabel('Pct. spillet')
264     plt.ylabel('Deck')
265
266 def NashCH(MixedEq_Decks, CHInput, frekvenser):
267
268     Standard_CH = [i*100 for i in CHInput]
269     count = 0
270     NumberOfGames = []
271     for i in frekvenser:
272         NumberOfGames.append(sum(i)+i[count])
273         count += 1
274     ShareOfGames = []
```

```
    for i in NumberOfGames:
        ShareOfGames.append(100 * i / sum(NumberOfGames))
    CombinedList = []
    for i in range(len(MixedEq_Decks)):
        CombinedList.append([MixedEq_Decks[i], ShareOfGames[i], Standard_CH[i]])
    dfGraph = pd.DataFrame(CombinedList, columns = ["Decks","Observationer", "Nash-CH"])
    my_range = range(1, len(dfGraph.index)+1)
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.hlines(y = my_range, xmin = dfGraph['Observationer'], xmax = dfGraph['Nash-CH'], color='peru', alpha = 0.4)
    plt.scatter(dfGraph['Observationer'], my_range, color='deepskyblue', alpha=1, label='Observationer')
    plt.scatter(dfGraph['Nash-CH'], my_range, color='khaki', alpha=1, label='Nash-CH model')
    plt.legend()
    textcordy = 7
    textcordx = 0
    for i, txt in enumerate(ShareOfGames):
        if 0 < ShareOfGames[i] - Standard_CH[i] < 1:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i], my_range[i]),textcoords='offset points',xytext=(textcordx+8, textcordy),ha='center',fontsize=8)
        elif -1 < ShareOfGames[i] - Standard_CH[i] < 0:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i], my_range[i]),textcoords='offset points',xytext=(textcordx-8, textcordy),ha='center',fontsize=8)
        else:
            ax.annotate("{:.1f}".format(txt), (ShareOfGames[i], my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy),ha='center',fontsize=8)
    for i, txt in enumerate(Standard_CH):
        if 0 < Standard_CH[i] - ShareOfGames[i] < 1:
            ax.annotate("{:.1f}".format(txt), (Standard_CH[i], my_range[i]),textcoords='offset points',xytext=(textcordx+8, textcordy),ha='center',fontsize=8)
        elif -1 < Standard_CH[i] - ShareOfGames[i] < 0:
```

```
300             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx-8,
    textcordy),ha='center',fontsize=8)
301         else:
302             ax.annotate("{:.1f}".format(txt), (Standard_CH[i],
    my_range[i]),textcoords='offset points',xytext=(textcordx,textcordy
    ),ha='center',fontsize=8)
303     #Titel og Akser
304     plt.yticks(my_range, dfGraph['Decks'])
305     plt.xlabel('Pct. spillet')
306     plt.ylabel('Deck')
```

# Referencer

M Allais. Le comportement de l'homme rationnel devant le risque: Critique des postulats et axiomes de l'ecole americaine. *Econometrica*, 21(4):503–, 1953. ISSN 0012-9682.

Robert Aumann og Adam Brandenburger. Epistemic conditions for nash equilibrium. *Econometrica*, 63(5):1161–1180, 1995. ISSN 0012-9682.

Sourabh Banerjee, Ayanendranath Basu, Sourabh Bhattacharya, Smarajit Bose, Dalia Chakrabarty, og Soumendu Sundar Mukherjee. Minimum distance estimation of milky way model parameters and related inference. *SIAM/ASA journal on uncertainty quantification*, 3(1):91–115, 2015. ISSN 2166-2525.

Carl Barenboim. Development of recursive and nonrecursive thinking about persons. *Developmental psychology*, 14(4):419–420, 1978. ISSN 0012-1649.

Paul T Boggs og Jon W Tolle. Sequential quadratic programming for large-scale nonlinear optimization. *Journal of computational and applied mathematics*, 124(1):123–137, 2000. ISSN 0377-0427.

Bosch-Domenech, Montalvo J.G, Nagel R, og Satorra A. One, two, (three), infinity, ...: Newspaper and lab beauty-contest experiments. *The American economic review*, 92(5): 1687–1701, 2002. ISSN 0002-8282.

C. F Camerer, T.-H Ho, og J.-K Chong. A cognitive hierarchy model of games. *The Quarterly journal of economics*, 119(3):861–898, 2004. ISSN 1531-4650.

Colin Camerer og Dan Lovallo. Overconfidence and excess entry: An experimental approach. *The American economic review*, 89(1):306–318, 1999. ISSN 0002-8282.

Colin Camerer, Teck Ho, og Kuan Chong. Models of thinking, learning, and teaching in games. *The American economic review*, 93(2):192–195, 2003. ISSN 0002-8282.

Daniel Carvalho, Daniel Carvalho, Luís Santos-Pinto, og Luís Santos-Pinto. A cognitive hierarchy model of behavior in the action commitment game. *International journal of game theory*, 43(3):551–577, 2014. ISSN 0020-7276.

Alessandra Cassar og Dan Friedman. *Economics Lab: An Intensive Course in Experimental Economics*. Routledge Advances in Experimental and Computable Economics. Routledge, Abingdon, Oxon, 2004. ISBN 9780415324014.

Juin-Kuan Chong, Colin F Camerer, og Teck-Hua Ho. Cognitive hierarchy: A limited thinking theory in games. pages 203–228, 2005.

Juin-Kuan Chong, Teck-Hua Ho, og Colin Camerer. A generalized cognitive hierarchy model of games. *Games and economic behavior*, 99:257–274, 2016. ISSN 0899-8256.

Jack Cleghorn og Mark D Griffiths. Why do gamers buy "virtual assets"? an insight in to the psychology behind purchase behaviour. *Digital education review*, (27):85–104, 2015. ISSN 2013-9144.

Miguel Costa-Gomes, Vincent P Crawford, og Bruno Broseta. Cognition and behavior in normal-form games: An experimental study. *Econometrica*, 69(5):1193–1235, 2001. ISSN 1468-0262.

Miguel A Costa-Gomes og Vincent P Crawford. Cognition and behavior in two-person guessing games: An experimental study. *The American economic review*, 96(5):1737–1768, 2006. ISSN 0002-8282.

Miguel A Costa-Gomes, Vincent P Crawford, og Nagore Iriberri. Comparing models of strategic thinking in van huyck, battalio, and beil's coordination games. *Journal of the European Economic Association*, 7(2-3):365–376, 2009. ISSN 1542-4774.

Vincent P Crawford. Lying for strategic advantage: Rational and boundedly rational misrepresentation of intentions. *The American economic review*, 93(1):133–149, 2003. ISSN 0002-8282.

Vincent P Crawford og Nagore Iriberri. Fatal attraction: Salience, naïveté, and sophistication in experimental "hide-and-seek" games. *The American economic review*, 97(5): 1731–1750, 2007. ISSN 0002-8282.

Vincent P Crawford, Miguel A Costa-Gomes, og Nagore Iriberri. Structural models of nonequilibrium strategic thinking: Theory, evidence, and applications. *Journal of economic literature*, 51(1):5–62, 2013. ISSN 0022-0515.

George Bernard Dantzig. *Linear programming and extensions*. Univ. Press, Princeton, N.J, 1963.

Constantinos Daskalakis. On the complexity of approximating a nash equilibrium. *ACM Transactions on Algorithms (TALG)*, 9(3):1–35, 2013. ISSN 1549-6325.

Peter Dizikes. The case for economics — by the numbers, *MIT News Office*, 3. marts, 2020.

Constantine A Drossos og Andreas N Philippou. A note on minimum distance estimates. *Annals of the Institute of Statistical Mathematics*, 32(1):121–123, 1980. ISSN 1572-9052.

Thomas S. Ferguson. A method of generating best asymptotically normal estimates with application to the estimation of bacterial densities. *The Annals of mathematical statistics*, 29(4):1046–1062, 1958. ISSN 0003-4851.

Thomas S. Ferguson. *Game Theory*, class notes for math 167 fall. *University of California: Los Angeles*, 2000.

Daniel Fragiadakis og Peter Troyan. Improving matching under hard distributional constraints: Improving matching under constraints. *Theoretical economics*, 12(2):863–908, 2017. ISSN 1933-6837.

Drew Fudenberg og Annie Liang. Predicting and understanding initial play. *The American economic review*, 109(12):4112–4141, 2019. ISSN 0002-8282.

Markus Glaser og Martin Weber. Overconfidence and trading volume. *The Geneva risk and insurance review*, 32(1):1–36, 2007. ISSN 1554-9658.

Mark Grinblatt og Matti Keloharju. Sensation seeking, overconfidence, and trading activity. *The Journal of finance (New York)*, 64(2):549–578, 2009. ISSN 0022-1082.

Teck-Hua Ho, Colin Camerer, og Keith Weigelt. Iterated dominance and iterated best response in experimental "p-beauty contests". *The American economic review*, 88(4): 947–969, 1998. ISSN 0002-8282.

HSReplay. *Bringing advanced gaming analytics and tools to players and companies in eSports*. tilgængelig på: https://hsreplay.net/ (tilgået august-december), 2020.

Jiekun Huang og Darren J Kisgen. Gender and corporate finance: Are male executives overconfident relative to female executives? *Journal of financial economics*, 108(3): 822–839, 2013. ISSN 0304-405X.

Daniel Kahneman og Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979. ISSN 0012-9682.

Daniel Kahneman, Jack L Knetsch, og Richard H Thaler. Experimental tests of the endowment effect and the coase theorem. *The Journal of political economy*, 98(6): 1325–1348, 1990. ISSN 1537-534X.

John Maynard Keynes. *The General Theory of Employment, Interest, and Money*. Springer International Publishing, Cham, 1936. ISBN 9783319703435.

J Kiefer og J Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of mathematical statistics*, 23(3):462–466, 1952. ISSN 0003-4851.

D. Kraft. *A software package for sequential quadratic programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988. URL https://books.google.dk/books?id=0D2DtAEACAAJ.

H. W Kuhn og A. W Tucker. Nonlinear programming. *Econometrica*, 19:50–, 1951. ISSN 0012-9682.

Tom Lane. Discrimination in the laboratory: A meta-analysis of economics experiments. *European economic review*, 90:375–402, 2016. ISSN 0014-2921.

David Leonhardt og Kevin Quealy. Puzzle: Are you smarter than 61,139 other new york times readers?, *The New York Times*, 13. august, 2015.

Steven D Levitt og John A List. What do laboratory experiments measuring social preferences reveal about the real world? *The Journal of economic perspectives*, 21(2): 153–174, 2007. ISSN 0895-3309.

Michael L. Littman. *Markov games as a framework for multi-agent reinforcement learning*. Morgan Kaufmann, 1994.

Richard D McKelvey og Thomas R Palfrey. Quantal response equilibria for normal form games. *Games and economic behavior*, 10(1):6–38, 1995. ISSN 0899-8256.

Rosemarie Nagel. Unraveling in guessing games: An experimental study. *The American economic review*, 85(5):1313–1326, 1995. ISSN 0002-8282.

Jorge. Nocedal. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, New York, NY, 1st ed. 1999. edition, 1999. ISBN 1-280-01067-3.

Paul A Samuelson et al. Economics: An introductory analysis, 1948.

Astrid Schou. Gæt-et-tal konkurrence afslører at vi er irrationelle, *Politiken*, 22. september, 2005.

SciPy. *Open-source software for mathematics, science, and engineering*, tilgængeligt på: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html (tilgået december), 2020.

Reinhard Selten. Features of experimentally observed bounded rationality. *European economic review*, 42(3):413–436, 1998. ISSN 0014-2921.

Thomas R Shultz og Karen Cloghesy. Development of recursive awareness of intention. *Developmental psychology*, 17(4):465–471, 1981. ISSN 0012-1649.

Adam Smith. *The theory of moral sentiments. By Adam Smith, Professor of Moral Philosophy in the University of Glasgow*. London, 1759.

Dale O Stahl og Paul W Wilson. On players models of other players: Theory and experimental evidence. *Games and economic behavior*, 10(1):218–254, 1995. ISSN 0899-8256.

Adrian Colin Cameron; P K Trivedi. *Microeconometrics : methods and applications*. Cambridge University Press, New York, NY, 2005. ISBN 9780521848053.

ViciousSyndicate. *Vicious Syndicate has pioneered rigorous data collection procedures and data analytics in Hearthstone*, tilgængelig på: https://www.vicioussyndicate.com/ (tilgået august-december), 2020.

John von Neumann. *ON THE THEORY OF GAMES OF STRATEGY*. Princeton University Press, 1928. ISBN 9780691079370.

John von Neumann og Oskar Morgenstern. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton classic editions. Princeton University Press, 1944. ISBN 0691130612.

J. D. Williams. *The compleat strategyst being a primer on the theory of games of strategy*. The Rand series. McGraw-Hill, New York, 1966.

J. Wolfowitz. Estimation by the minimum distance method in nonparametric stochastic difference equations. *The Annals of mathematical statistics*, 25(2):203–217, 1954. ISSN 0003-4851.

Guy Woodruff og David Premack. Intentional communication in the chimpanzee: The development of deception. *Cognition*, 7(4):333–362, 1979. ISSN 0010-0277.