Joachim Reinhardt

Social Preferences in Asymmetric Commons: A Simulation Study

Bachelorarbeit

Themensteller: Prof. Dr. Carlos Alós-Ferrer

Vorgelegt in der Bachelorprüfung im Studiengang Volkswirtschaftslehre
der Wirtschafts- und Sozialwissenschaftlichen Fakultät der Universität zu Köln

Köln, September 2016

Contents

List of Figures

Part I

Analyzing Asymmetric Commons Dilemmas: Some Important Concepts and Approaches

Chapter 1

The "Tragedy of the Commons"

Coined by the ecologist Garrett Hardin in an 1968 article of the same title [17], the expression "tragedy of the commons" is used to denote situations in which individuals face the predicament of choosing between either preserving a finite resource for the public, but forgoing personal gain and risking exploitation by other individuals; or exploiting the resource for personal gain. Assuming that "rational" agents would maximize their personal utility by extracting the maximum possible amount from the resource, Hardin concluded that no finite common resource could be preserved by a group of individuals with free access to the common good. For Hardin, the collapse of the common resource is thus inevitable; "[r]uin is the destination toward which all free men rush [...] in a society that believes in the freedom of the commons" (p. 1244). He illustrates this with the example of a pasture which a number of herdsmen have free access to. The individual herdsmen are tempted to increase the number of animals in their herd, leading to a faster depletion of the amount of food available in the pasture. While the depletion of the pasture is not a desirable outcome for any of the herders, shepherds who restrain the number of animals in their herd get no direct personal gain from doing so; while herders extending their herd directly benefit from the exploitation of the pasture.[1]

The resources described by Hardin are so-called Common-Pool Resources (CPR). Gardner, Ostrom and Walker [16] identify two main conditions for a resource to be classified as a common-pool resource: yields from the resource need to be subtractable, i.e., "a resource unit withdrawn or harvested by one individual is not fully available to another individual" (p. 336), and there need to be multiple ap-

[1]While the example of the pasture describes a resource from which goods are extracted, Hardin also outlines the opposite case of the pollution of a pristine resource - interestingly, his example of choice is the pollution of public spaces with "mindless music" and advertisements (p.1248).

propriators harvesting the resource who are not easily excluded from doing so. As Gardner, Ostrom and Walker note, these two conditions do not necessarily constitute a situation in which there is a "dilemma". They characterize a "dilemma" situation through the further conditions of suboptimal outcomes produced by the appropriators' actions, and the possibility of changing appropriators' strategies to result in an efficient outcome. In Gardner's, Ostrom's and Walker's framework, there is thus a distinction between a mere CPR "situation", which is not perceived to be problematic, and a CPR "dilemma", which produces suboptimal outcomes that could, in theory, be improved.

As noted by Ostrom, Dietz, Dolšak and Stern [7], Hardin's view of the sustainability of common resources has influenced researchers and policymakers alike, favoring a centralized management of common resources by either private or public entities. As the authors remark, this central management has often had problematic outcomes - communities had often developed decentralized structures and mechanisms to manage common resources, which were missed by many policy makers and researchers. A case study emphasizing the importance of local, decentralized governance of common resources is provided by Lansing [25], who studied irrigation structures in Bali and describes (as he argues, misguided) efforts to centralize control over these irrigation systems. In a subsequent article, Lansing and Kremer [24] outline how decentralized management structures which outperform centralized control could emerge spontaneously.

Ostrom, Walker and Gardner [31] provide an extensive overview of field studies of existing CPR situations in part three of their book, including many systems of rules designed by appropriators themselves which sustained the CPR over an extended amount of time. The question thus arises how effective decentralized management over CPR can be developed without being imposed by any external authority - and how, as Lansing and Kremer [24] show, it might be emerging spontaneously from interactions between individuals.

1.1 A Game-Theoretical Approach: Prisoner's Dilemma and Public Good Games

As Dietz, Dolšak, Ostrom and Stern [7] note, common-pool resource dilemmas are frequently modeled as Prisoners' Dilemma games in game-theoretic analyses, either pair-wise or generalized to an N-person game. Alternatively, the Public Goods game is often used - as shown by Hauert and Szabó [18], the two games

are equivalent in a generalized form and can be transformed into one another.
In the two-player PD game, each player faces a binary choice: cooperation or defection. Following Hauert's and Szabó's description, both players cooperating results in both players receiving a reward R as a payoff; should both players choose to defect, they each receive a punishment P. If one player cooperates and the other defects, the defector receives a high payoff T (for "temptation"), while the cooperator receives a low payoff S, the "sucker's payoff"; with $T > R > P > S$. Irrespective of the other player's move, a player's short-term payoff is maximized by playing "defect" - if the other player cooperates, the player makes T instead of R; in case the other player defects, the player's payoff is P instead of S. If both players apply this reasoning, they end up with a total payoff of $2P$ instead of the higher payoff $2R$ if the players cooperate.
In a Public Goods game, a group of players are endowed with a certain amount of monetary units and choose to invest a certain amount of their endowment into a common resource. The total investment by all players is then multiplied by a certain factor $r > 1$, typically $r = 2$, and split equally between all players. If all players invested their whole endowment, the group could thus double its payoff. However, individual players can maximize their short-term payoff by not investing anything since all players receive a share of the common resource regardless of whether they invested or not, and since all players have to split the rewards of their own investment with all other players. Analogously to the Prisoners' Dilemma, all players would be better off if each player invested their whole endowment, but individual players could exploit others' investments while not investing anything themselves.
An n-player Prisoners' Dilemma game with respective parameters is thus equivalent to an n-player Public Goods game where players can only choose to invest all or nothing of their endowment. A Public Goods game with more than two investment levels could therefore be understood as a more nuanced version of the Prisoners' Dilemma, with multiple "shades" of cooperation and defection.
Gardner, Ostrom and Walker [16] remark that the PD game, and thus also the Public Goods game, is not necessarily a good representation of CPR situations, as these do not necessarily constitute a dilemma. In situations where appropriators do not wish to extract sufficiently large amounts from the CPR, the outcome is not suboptimal and thus no dilemma results, making the PD game not an accurate representation.

1.2 Social Preferences

Underlying approaches to explaining behavior in commons dilemmas such as Hardin's or the application of game-theoretic analyses, such as in the Prisoner's Dilemma, is the assumption that agents behave as rational agents seeking to maximize their personal gain only. However, if the agents cared about the welfare of others in some way, different outcomes could be expected. In economic analyses, agents are modeled to make decisions based on a utility function specified by the modeler. In standard analyses of Prisoners' Dilemma games, the agents' utilities are equal to their payoff. Utility functions could, however, include parameters that make an agent's utility dependent on other agents' payoffs; which are known as social preferences.

These parameters would be set to zero for "selfish" agents. However, the specification of social preferences does not necessarily imply that agents act in an altruistic manner - competitive agents who prefer their payoffs to be higher than others would also be modeled through social preferences.

As noted by Bester and Güth [4], merely including social preferences into utility functions does not explain how such social preferences might be developed. In their paper, they examine under what circumstances altruistic preferences can be evolutionarily stable - i.e., whether a population of mostly altruistic agents "survives" confrontation with selfishly motivated agents among them; or if agents always benefit from acting purely selfish, displacing altruistic agents.

1.3 Agent-Based Modeling Approach

The emergence of complex outcomes from individual units interacting according to simple rules has been studied under the cellular automata approach. An overview of such cellular automata is given by Wolfram [41], who describes patterns generated by simple, identical components. Cellular automata, as noted by Poteete, Janssen and Ostrom [32], could be understood as simple agent-based models, but are highly restricted by their simplicity and limitation to identical units.

Agent-based computational economic models, as defined by Tesfatsion [35], investigate "economic processes modeled as dynamic systems of interacting agents" (p. 835). As Epstein and Axtell [13] note, agent-based models require no assumptions about the rationality of agents and, importantly, allow an arbitrarily high degree of heterogeneity between agents. Epstein and Axtell also emphasize agent-based computational models' ability to capture spatial relations distinctly

from relations between agents, the possibility to reach complex results with highly simple rules, and the possibility of dynamical analyses. For Poteete, Janssen, and Ostrom [32], agent-based models can be described as "information-processing algorithms based on various assumptions about the cognitive ability of the individual agents and the topology of their interactions (networks)" (p. 171). Agents in agent-based models, as Tesfatsion [35] remarks, are not limited to being economic agents but can also represent social, biological, and physical entities.

Epstein and Axtell [13] describe agent-based modeling as a "generative approach" to social science which provides demonstrations that given "sets of microspecifications are sufficient to generate the macrophenomena of interest" (p. 20). An agent-based model could thus be used to demonstrate that cooperation in CPR dilemmas can be generated by agents interacting on a micro level, without being imposed by an external authority. As noted by Flake [14], this does not mean that agents need to resemble real-world individuals closely - it is sufficient to show that the "macrostructure of interest" can be generated from the very limited behavior of such agents.

Tesfatsion [35] emphasizes the complex dynamic properties of economic systems. As he argues, agent-based models are suited to analyze economies as complex adaptive systems - instead of relying on static analyses, agent-based simulations can be used to model dynamic processes. In contrast to traditional economic models, agent-based simulations do not rely on central mechanisms controlling the behavior of agents. Tesfatsion uses the Walrasian auctioneer, a central concept in standard economic theories, as an example. As he argues, agent-based models can be used to test if, and under what conditions, results as predicted by traditional economic theories can arise without the assumption of the Walrasian auctioneer pricing mechanism.

Another important feature of agent-based models is that they can model individuals' ability to learn. Agents can evaluate past outcomes simultaneously and continuously, adjusting future actions accordingly. Brenner [6] gives an extensive summary of how models of different learning processes are informed by psychological research. As outlined by Duffy [8], agent-based models are well-suited to incorporate findings from experimental studies. Since, as he argues, most laboratory experiments in economics and social sciences are carried out using computers, experimental setups can easily be transferred into the domain of agent-based models. A human subject in an experiment can be replaced by a computational agent, allowing extensive experimentation and comparisons between "human-based" and agent-based experiments.

Part II

A Replication Study of M.A. Janssen's and N.D. Rollins' Agent-Based Simulation of the Evolution of Cooperation in Asymmetric Commons

Chapter 2

M.A. Janssen's and N.D. Rollins' "Evolution of Cooperation in Asymmetric Commons Dilemmas"

2.1 Introduction

Marco A. Janssen's and Nathan D. Rollins' 2012 paper on the "Evolution of Cooperation in Asymmetric Commons Dilemmas" [22] explores how the high degree of cooperation observed in real-world irrigation systems for agricultural purposes can be reproduced in an agent-based computational representation. In such irrigation systems, agents' selfish interests usually conflict with the group's overall welfare and agents' access to the resource is often asymmetric. As far as they constitute an common-pool dilemma in the sense of Gardner, Ostrom and Walker [16], irrigation systems provide an illustrative example for the study of common-pool resources since they require coordination in the provision of the irrigation system as well as in the appropriation of the water - see also Ostrom, Gardner, and Walker [31]. In rural contexts, it can be assumed that irrigation systems are usually set up by the same individuals that use the water. This makes individuals interdependent in the situation as a whole, even if they face different incentives in the provision or appropriation phase; this point is emphasized in Janssen's and Rollins' paper. Importantly, as Janssen and Rollins[1] note, there are individual differences in the access to the water provided by an irrigation system - individu-

[1]Unless otherwise noted, "Janssen and Rollins" subsequently refers to [22].

als at the head-end of the (flowing) resource have unlimited access to the water, while tail-end individuals only receive what is left by the upstream users. Janssen and Rollins thus compare head-end individuals to "stationary bandits" or dictators as described by Olson [29]. While head-enders are, in theory, free to take an arbitrarily high share of the water resource, they depend on the cooperation of downstream individuals in the provision of the resource - who will not cooperate unless they receive at least some recompensation. Individuals at the head-end of the resource thus have an incentive to not take more water than downstream users will tolerate.

Based on experimental results confronting participants with an asymmetric dilemma of the commons, Janssen's and Rollins' agent-based model explores how other-regarding preferences - such as altruistic tendencies and aversion to exploitation by others - can evolve in populations of originally selfish agents playing an irrigation game.

2.2 Sample of Related Literature

The agent-based model presented by Janssen and Rollins analyzes cooperation in asymmetric commons dilemmas. Major previous research on asymmetric commons, especially concerning irrigation structures, acknowledged by them includes Ostrom and Gardner [30], Lansing [25], and Hunt [19].

The main issue concerning cooperation in irrigation systems is outlined by Ostrom and Gardner [30] the asymmetry between individuals at the head-end of the resource and individuals located at a more downstream position. They present a model in which upstream individuals rely on downstream individuals in the provision of the irrigation system, but face the temptation to extract disproportionally large shares of water from the system once it is established. The irrigation game they described is, in essence, very similar to the one used by Janssen and Rollins - however, instead of an agent-based computational model, Ostrom and Gardner rely on analyzing the strategic interaction between two representative players. Further general works on commons dilemmas co-authored by Ostrom include Ostrom, Gardner, and Walker [31], Ostrom, Dietz, Dolšak, Stern, Stonich and Weber (Eds.) [7], and the methodological guide by Poteete, Janssen, and Ostrom [32].

Hunt's paper [19] presents an examination of published case studies of large-scale irrigation systems. He finds that irrigation structures need not necessarily be subject to centralized control in order to function, and presents evidence of large

irrigation structures organized in a decentralized manner.

Janssen and Rollins acknowledge a number of previous studies on the level of cooperation in symmetric vs. asymmetric commons dilemmas, including Ahn, Lee, Ruttan, and Walker [1], Beckenkamp, Hennig-Schmidt, and Maier-Rigaud [3], and Tan [34]. Ahn, Lee, Ruttan, and Walker as well as Beckenkamp, Hennig-Schmidt, and Maier-Rigaud consider Prisoners' Dilemma games with asymmetric payoffs. Both studies find that introducing asymmetry in the structure of payoffs significantly decreases cooperation. Tan analyzes linear public good games in which agents' investments into the public good have asymmetric returns. Tan finds that groups with asymmetric returns have lower efficiency than those with symmetric return functions.

As noted by Janssen and Rollins, their model draws insights from Boyd's, Gintis', Bowles' and Richerson's [5] model of the evolution of altruistic punishment. Boyd, Gintis, Bowles and Richerson explore how punishment of non-cooperating agents might develop even if punishment comes at a cost to the punishing agent. Their modeling of the evolutionary selection process between and within groups of agents is adapted by Janssen and Rollins to model their cultural selection process.

In their discussion of how perceived fairness and equality are related to efficiency, Janssen and Rollins reference Wilke's Greed-Efficiency-Fairness (GEF) hypothesis [40], as well as Eek and Biel [10] and Eek, Biel, and Gärling [11]. Wilke described the interplay of motivation by greed, efficiency, and fairness considerations when dealing with common resources, concluding that higher perceived fairness leads to increased cooperation. Eek and Biel present experimental evidence on how considerations of efficiency and perceived fairness affect cooperation. In accordance with Eek, Biel, and Gärling, they find that perceptions of what constitutes "fairness" differ across human individuals, thus affecting the willingness to cooperate in different scenarios. In the light of these results, Janssen and Rollins don't consider how *fair* or *unfair* a particular allocation of resources might be perceived. Instead, they discuss how *equal* or *unequal* the distribution of allocated shares is.

Janssen and Rollins place the development of irrigation systems into the "larger socio-historical context in which prior coordination and collective action problems had likely been solved in the hunter-gathering regimes and the resulting social norms were carried over through the transition into mixed farming" (p. 221). They reference Flannery [15], who describes the transition from a hunter-gathering lifestyle towards agricultural production and herding in Greater Mesopotamia between 7000 and 6000 B.C. As he notes, this transition brought about increased cooperation and exchange between groups.

Van Campenhout, D'Exelle, and Lecoutere [36] present experimental research on resource allocation in irrigation games. Similarly to Janssen and Rollins, van Campenhout, D'Exelle and Lecoutere introduce an asymmetry into the structure of the irrigation game. Instead of different positions in respect to the water source, they assign convex production functions to players. This means that, in contrast to Janssen's and Rollins' model, players achieve efficient outcomes by allocating resources as unequally as possible. They find that a rotation strategy is effectively implemented by some players, taking turns in extracting most of the available resource each round.

Ng, Wang, and Zhao [28] consider iterated n-person Prisoners' Dilemmas with asymmetric payoffs designed to represent real-world water sharing processes. As in Janssen's and Rollins' paper, asymmetries are due to different positions along a flowing water resource. They find that a high level of asymmetry incentivizes head-end players to cooperate since they in turn depend on downstream players' cooperation. While the river as a water resource considered by Ng, Wang and Zhao requires no "provision" in the sense of building it, reciprocity between up- and downstream agents is assumed to be due to their understanding of the river as a complete ecological system on which both the actions of head- and tail-enders have an effect.

Safarzynska [33] proposes the risk of environmental degradation due to overharvesting as a driving force behind the emergence of cooperation between individuals. In a group selection model akin to Boyd, Gintis, Bowles, and Richerson [5] - which was also employed by Janssen and Rollins - she finds cooperative behavior to evolve under certain conditions.

Krebs, Holzhauer and Ernst [23] present an agent-based model of self-organized volunteering efforts in local areas during times of extreme climate conditions, such as heat waves. Representing support in local neighborhoods as a public good, with asymmetries in the structures of neighborhood inhabitants, Krebs, Holzhauer and Ernst try to identify what regions in the German region of Northern Hessia are likely to have a significant number of volunteers.

2.3 Experimental Data

Janssen and Rollins build upon an experimental study that put participants in different positions in an asymmetric commons dilemma irrigation game, the results of which are described in more detail in a subsequently published paper by Janssen, Bousquet, Cardenas, Castillo and Worrapimphong [21]. The design of this asymmetric commons game tries to capture salient features of real-world irrigation systems and serves as the framework for Janssen's and Rollins' agent-based computational model.

The asymmetric commons game is played for ten rounds, each of which consists of an investment and a subsequent extraction phase. At the beginning of each round, all players receive an endowment of 10 tokens, and simultaneously decide how much of this endowment to invest into the common-pool resource (CPR). The amount of CPR that is available to players in the extraction round is a discontinuous function of the total investment by all players - refer to Table one (p. 66) in Janssen, Bousquet, Cardenas, Castillo and Worrapimphong [21][2]. This function generates no resource for investments below 10 tokens, meaning that at least two players need to invest for any resource to be generated. For an investment of 50 tokens, i.e. all players investing all of their endowment, the investment is doubled, with 100 units generated. Players keep the part of their endowment they decided not to invest. Tokens given as an endowment and units generated as CPR have the same value, and participants were able to exchange them to real-world currency after the experiment.

In the share allocation phase, all players make a decision of how much to extract from the CPR. This is done in turns, with every player being assigned a specific position; A, B, C, D, and E; which remains unchanged for all ten rounds of the game. The player at position A can be thought of as the player with the position at the head-end of the irrigation system and gets to extract from the resource before all other players. The player at position B gets to extract from the remaining resource next, and so forth, until all players extracted their share from the CPR or all of the CPR is depleted. Any CPR remaining after the last player's extraction is not transferred to the next round is lost and cannot be extracted by any agent in future rounds of play.

The emerging dilemma can be outlined as follows: while the player at the most upstream position is free to take all of the generated resource, this player also

[2]Table one in Janssen's and Rollins' paper (p. 223) shows the same function with all values divided by 10.

depends on the cooperation of the other agents in the investment phase of all subsequent rounds of play. If the player at position A extracts all of the available CPR, the players at more downstream positions have no incentive to invest in the CPR at all. The same reasoning, to a lesser extent, applies to players B, C, and D, who could take all of the remaining resource, but would benefit from cooperation of downstream players in the next round.

If none of the players in a group invests into the CPR in any of the ten rounds, the total payoff for all players amounts to the 500 tokens players received as endowment. If, on the other hand, all players invest all of their endowment, the total payoff increases to 1000 tokens[3], meaning that a group of players at a whole highly benefits from cooperation (in the form of high investments into the CPR) between players.

The experiments were carried out with participants from rural areas as well as university students in Colombia and Thailand.

Janssen and Rollins present their experimental results in Figure one (p. 222) in their paper, capturing the distribution of investments and extractions between participants at the five different positions. They find that investment into the CPR was invariant to the subjects' position in the irrigation game, with an average investment of about 5 tokens for all five positions. Despite roughly equal contributions, upstream players receive a significantly larger share of the CPR, with players on position A extracting about four, position B about 2.5, and position C about 2 times their investment. Subjects at position E received, on average, less units than they invested.

In Janssen, Bousquet, Cardenas, Castillo and Worrapimphong [21], the authors report a positive correlation between equality of share allocation in one round and higher levels of investment into the CPR in the following round, and thus reach the conclusion that an equal share allocation is the necessary condition for a high total payoff in a group of players.

[3]In fact, players can exploit the step-shape of the resource generating function and invest only 46 tokens per round, resulting in a total payoff of 104 units per round, or 1040 units per game.

2.4 Model

Janssen and Rollins base their conceptual model on the experimental setup described above, replacing human participants with computational agents acting according to a probabilistic choice algorithm. The endowment of 10 tokens in the experiment is normalized to one in the agent-based model, with the discontinuous function generating the CPR scaled down by a factor of 1/10 (see Table one in Janssen's and Rollins' paper, p. 223).

In the investment phase, agents make a choice between ten levels of investment k into the CPR, from 0 to 1 in 0.1 increments. They do so via a probabilistic choice algorithm, which calculates the expected utility derived from each level of k given the amounts of investments and extractions of the other agents in the last round of play, and returns a corresponding probability for each k.

The utility an agent i derives from a specific outcome depends on the agent's wage, defined as

$$w_i = (1 - x_i) + y_i \cdot y_{i-1} \cdots_1 \cdot g \tag{2.1}$$

(equation two in Janssen's and Rollins' paper, p. 224) where g is the function determining the amount of CPR generated as defined in Janssen's and Rollins' table one (p. 223), x_i denotes the agent's investment, and y the extractions from the CPR by agent i and all agents with an upstream position relative to agent i. An agent's utility also depends on two other-regarding parameters[4], α and β, which alter the utility depending on the relation between an agent's wage w_i and the average wage among all agents, $\bar{w}$

$$u_i = w_i - \alpha_i \cdot max(w_i - \bar{w}, 0) + \beta_i \cdot max(\bar{w} - w_i, 0) \tag{2.2}$$

(equation three in Janssen's and Rollins' paper, p. 224).

The probability for choosing each of the ten investment levels k is calculated as

$$P(x_{i,k}) = \frac{e^{\eta \cdot E[u(x_{i,k}|x_{j,k,t-1};y_{j,k,t-1})]}}{\sum_{m=k}^{M} e^{\eta \cdot E[u(x_{m,k}|x_{j,k,t-1};y_{j,k,t-1})]}} \tag{2.3}$$

(equation four in Janssen's and Rollins' paper, p. 224).

Similarly, in the share allocation phase, agents decide on a share level l to extract from the CPR available to them. The probability of choosing each share level l is calculated as

$$P(y_{i,k}) = \frac{e^{\eta \cdot E[u(y_{i,k})]}}{\sum_{m=k}^{M} e^{\eta \cdot E[u(y_{i,m})]}} \tag{2.4}$$

[4]Similarly to the utility function described by Bester and Güth [4]

(equation five in Janssen's and Rollins' paper, p. 224).

The parameters α and β are set so that $\beta \leq \alpha \leq 1$. α decreases an agent's utility for cases where an agent's wage w_i is above the average wage $\bar{w}$, and can thus be understood as an agent's "aversion to exploiting others", as termed by Janssen and Rollins (page 224). For $\alpha < 0$, an agent derives extra utility if this agent's wage is higher than the average wage. β increases the agent's utility for $\bar{w} > w_i$ and could be said to measure the agent's tendency towards altruism, as Janssen and Rollins note.

The parameter η captures an agent's tendency to choose the investment level or extraction share that maximizes the agent's utility. For $\eta = 0$, the agent's choices are invariant to the utility associated with them, and the agent chooses a level with uniformly random probability. For $\eta \to \infty$, only the option(s) with the highest utility is (are) chosen. As defined in Table two in the paper by Janssen and Rollins (p. 225), the default value for η is 0.5.

In Duffy's [8] distinction between reinforcement vs. belief learning algorithms on the agent level, the probabilistic choice model described above fits into the belief learning category. Agents do not, as with a reinforcement learning algorithm, consider the payoffs associated with a certain choice in previous rounds of play. Instead, the agents form a "belief" of what the other agents' next move might be, and calculate their resulting utility in response. This belief is not formed in a sophisticated manner - agents simply assume that the other agents will make the same choices they made in the previous round of play. This is equivalent to a general belief learning algorithm with all previous rounds except for the very last one discounted to zero (see page 980 in Duffy [8]).

In addition to the asymmetric commons dilemma as played in the experimental setup, Janssen and Rollins specify that with frequency $1 - f_a$, a linear public good game (as described in section 1.1 above) is played, with $0 \leq f_a \leq 1$ as the frequency of asymmetric commons games. In the linear public good game, each agent automatically receives 1/5 of the CPR, and the function generating the CPR changes to the linear function

$$g = r \sum_{i=1}^{n} x_i \tag{2.5}$$

(equation six in Janssen's and Rollins' paper, p. 224), with no further specification of the r parameter.

The model includes a cultural selection process which is adapted from Boyd, Gintis, Bowles, and Richerson [5]. After the asymmetric commons or linear public

good game was played for $c = 10$ rounds, the fitness of the group, W_i, and the payoff of individual agents, w_i, are calculated. With probability m, an agent's payoff is then compared to that of a member of a randomly chosen other group, with a comparison between members within the same group taking place with probability $1 - m$. In both cases, agent i copies the traits[5] of the agent j it is being compared to with with probability

$$p = \frac{w_j}{w_j + w_i} \tag{2.6}$$

and vice versa (equation on p. 224 bottom in Janssen's and Rollins' paper).

With a probability of $\epsilon = 0.015$ after each set of rounds, a group i as a whole copies the traits of another randomly chosen group j with probability [6]

$$P = \frac{1 + (W_i \cdot s_i - W_j \cdot s_j)}{2} \tag{2.7}$$

with s_j as the fraction of agents in a group that "survived" by having a wage of $w_i \geq 1$. [7] It appears that in this formula, $W_i \cdot s_i$ and $W_j \cdot s_j$ appear in the wrong order - for $W_j \cdot s_j > W_i \cdot s_i$, the probability of group j being copied is smaller than for $W_j \cdot s_j < W_i \cdot s_i$ despite group j's relative fitness value being higher in the former case.

It is not specified how the fitness W of a group is calculated. In Whitley's [38] tutorial on genetic algorithms, fitness is defined as $f_i/\bar{f}$, with f denoting a specified evaluation of a group's performance. In the present case, the total payoff (or total efficiency) a group has achieved would be the most natural performance measure, meaning a group's fitness is calculated by the group's total payoff divided by the average total payoff of all groups. As Whitley notes, a group's rank among all groups could also be set as the performance measure - in this context, however, the groups' payoffs are easily calculated and provide a more accurate measure.

Janssen and Rollins specify that "at the end of each generation" of $c = 10$ rounds

[5] "Traits" refer to the other-regarding parameters α and β, but the parameter η could also be subject to imitation. In the model description provided by the authors online, it is stated that only α and β are adjusted in the simulation. See https://www.openabm.org/model/2274/version/2/view for this model description (retrieved September 21, 2016).

[6] Vice versa, group j copies the traits of group i with probability $1 - P$, as defined on p. 224 by Janssen and Rollins. Janssen and Rollins use the formulation "group i defeats group j", which logically reverses the relationship, since the *defeated* group copies the other group's traits. In formula 2.7, $W_i \cdot s_i$ and $W_j \cdot s_j$ therefore appear in inverse order.

[7] Even though not specified by Janssen and Rollins, this likely refers to the agent having a payoff at least equal to their total initial endowment in the ten rounds, i.e. a total payoff of $w_i = 10$.

played, "a small amount of white noise $n(0, \sigma)$ will affect the probabilities that define agent strategies" (p. 225), with a default value of $\sigma = 0.01$. The application of a noise term is necessary for any differences between groups and agents to emerge - as all agents start with the same default parameter values, there is no other mechanism for the development of different strategies. With the noise term being added, agents' strategies change at random - which allows a differentiation of payoffs and thus the selection of strategies with higher fitness.

2.5 Results

Janssen and Rollins present their model's main results in a number of graphics. In order to determine the results for parameters in the range of $-1 \leq \beta \leq \alpha \leq 1$, the authors ran the model with identical values for parameters α and β for all agents. The resulting average level of generated CPR per round as well as the average gini coefficients for investments and extractions between agents are plotted in Figures two, three and four (pp. 225f) in the original paper for different combinations of α and β. Since the values of α and β are fixed to specific values here, the model used to obtain these results necessarily excluded the imitation of other agents and groups. It is not entirely clear if the results were obtained by modeling only asymmetric commons dilemmas or if linear public goods games were also included with some frequency $1 - f_a$. The authors stating that "Fig. 2 illustrates the difficulty of agents evolving to values of α and β that generates positive values of the asymmetric resource" (pp. 225ff) indicates that the former is the case. This issue will be explored further in section 3.2 when discussing the results of the replication model.

Janssen's and Rollins' figure two (p. 225) shows that the amount of CPR generated is sensitive to both parameters. No significant amounts of CPR are generated for $\beta < 0$, and the generated amounts strictly increase in β. The highest amount of generated CPR is obtained for $\alpha = \beta = 1$, with an absolute value of just below 7.

The gini coefficient of investments graphed in figure three (p. 226) shows investments into the CPR to be most unequal for $\alpha = \beta = -1$, decreasing in both parameters, with a fairly equal distribution of investments for α and β close to one.

The gini coefficient of extractions from the CPR as depicted in figure four (p. 226) shows the inequality of extractions roughly increasing in α and β, with a slight drop for values of α and β close to one. The inequality for values of $\beta < 0$ is very low, corresponding to the fact that for these parameter values, almost no

CPR which could be extracted was generated in the first place (as shown in figure two, p. 225). With a maximum value of between 0.7 and 0.8, the inequality of extractions from the CPR is significantly higher than the inequality measure for investments into the CPR, which peaks at a gini coefficient of about 0.55.

Janssen and Rollins give an intuition about how α and β relate to decisions made by human subjects in their experimental data by comparing subjects' decisions to results given by the choice algorithm in equivalent situations. This way, parameter values for α and β "closest", i.e. minimizing the error between observation and simulation, to each human subject's actual decisions were obtained. Janssen's and Rollins' graph in figure five (p. 226) shows most participants could be assigned an α value close to one, while values for β have a high variance and range between 1 and about -1.75.

Janssen and Rollins report that when running the model including the cultural selection model for agents starting as "selfish", i.e. with parameters $\alpha = \beta = 0$, no significant investments into the CPR are observed in most of the cases. The authors find the frequency of asymmetric commons (vs. linear public good games) f_a to be the crucial setting affecting the level of investments into the CPR. The higher the frequency of linear good games being played, the higher the evolved values of the parameters α and β, and the higher the average level of the CPR (see figures six and seven in Janssen's and Rollins' paper, p. 227). Janssen and Rollins find the amount of generated CPR to be either close to zero, or, if a sufficient amount of linear good games is being played, around 6, with little values above or in between the two main outcomes (see Janssen's and Rollins' figure eight, p. 228).

On the level of individual agent's extraction decision from the CPR, Janssen's and Rollins' figure nine (p. 228) shows the agents at the head-end position (position A) to take about half of the generated CPR if mostly linear good games are played, and extracting 40% or less for $f_a \geq 0.5$. In these cases, however, the overall level of generated CPR is fairly low (see discussion of figures seven and eight, pp. 227f, above), meaning that the relatively equal distribution of extractions and high levels of generated resource observed in the experiments are unlikely to be found in the computational model, as the authors note.

Janssen and Rollins also consider the case of setting f_a to zero at the start of the simulation, and gradually increasing its value after a certain number of rounds being played. With such a delay of 200 time steps or more, Janssen and Rollins find significant amounts of CPR generated, with less unequal extractions from the CPR between the agents (figures ten and eleven, pp. 228f). In this setting, the

results bear more resemblance to the experimental results than those without a gradual increase in f_a.

As Janssen and Rollins conclude, their results indicate that high levels of cooperation (as observed in their experimental results) are improbable to result from selfish agents playing asymmetric common dilemma games only, if they interact in the way specified by their model. For significant levels of cooperation to be found, linear good games had to be introduced into the model with some frequency; or the model had to start with only linear good games being played, only gradually increasing the frequency of asymmetric common dilemmas.

Chapter 3

Replication of Janssen's and Rollins' Model

For the sake of clarity, the term "original model" will be used to refer to the implementation as presented by Janssen and Rollins [22]. "Replication model" will refer to the replication effort by this author, presented below. Both implementations are based on the conceptual description of the model in Janssen's and Rollins' paper. This will be termed the "conceptual model", which both original and replication model share. This distinction between the conceptual model, which is an abstraction of the real-world phenomenon it represents and described in real language; and the implemented model, which in turn is a representation of the conceptual model and is written in a programming language, follows Edmonds and Hales [9].[1]

3.1 Replication Model

In a 2007 paper on agent-based model replication, Wilenksy and Rand [39] state that "[a]n original model and an associated replicated model can differ across at least six dimensions: (1) time, (2) hardware, (3) languages, (4) toolkits, (5) algorithms and (6) authors" (section 2.7ff).
In case of the replication presented here, all dimensions except for the algorithm were varied. The first dimension, time, refers to the simple rewriting of the model code at a later time with referral to the original conceptual model, but independently of the code of the first implementation - together with dimensions (2) hardware and (6) authors, this dimension varied from the original model for evi-

[1] "Original paper", "Janssen and Rollins", etc. all refer to [22]

dent reasons. The original model was implemented in the NetLogo programming language. The replication model is written in Python 3.5, thus differing in the (3) language used. Wilensky and Rand consider NetLogo to be both a programming language and a toolkit, which is a program library meant to facilitate the implementation. For the replication model, the python framework mesa was used as the main toolkit - see Masad and Kazil [26] for a description.
Additionally, the numpy (as described by van der Walt, Colbert, and Varoquaux [37]) and pandas (as described by McKinney [27]) packages were used for parts of the calculations; as well as the matplotlib package for visualization, as described by Hunter [20].
As described above, Janssen's and Rollins' model consists of an irrigation game played for a specified number of rounds within groups of agents, and a subsequent cultural group selection process between groups. The irrigation game consists of an asymmetric commons dilemma with a certain frequency, and a linear public goods game in the other cases.
In the replication model, only the irrigation game is considered, both for asymmetric common and linear public good games. A comparison with the original model is possible since Janssen and Rollins published results for the full parameter range of the irrigation game part of the model (see figures two, three, and four in Janssen's and Rollins' paper, pp. 225f).
In order to determine if a successful replication was achieved or not, a replication standard has to be set. Axtell, Axelrod, Epstein and Cohen [2] term a successful replication as being "equivalent" to the original model - i.e, the two models "produce equivalent results in equivalent conditions" (p. 124), and determine three levels of equivalence: "numerical identity", "distributional equivalence", and "relational equivalence". For numerical identity, two models are required to give the exact same numerical output when given the same starting conditions and parameters - which, as the authors note, is not a useful criterion if the models in question include a stochastic element. For two models to be distributionally equivalent, their outputs need to be indistinguishable as far as the statistical distribution of results is concerned. Relational equivalence refers to two models "produc[ing] the same internal relationship among their results" ([2], p. 135).
Since the Janssen and Rollins model both has a probabilistic element and exact numerical values of their results are not provided, the numerical identity and distributional equivalence standards are not feasible in this case. Thus, the standard for a successful replication will be reached if the replication model produces results with the same internal relationship. Janssen and Rollins provide result

charts depicting the level of the common public resource produced as well as the gini coefficent of investments and extractions for a range of parameter values (refer to figures two, three, and four in their paper, pp. 225f). These charts will be used to determine whether a successful replication was achieved or not. A relationally equivalent model would have a result graph with the same shape as the original, with similar absolute values.

Although Janssen and Rollins provide the full code to their model online, this resource was not used for the replication. The replication model is based solely on the verbal and mathematical description of the conceptual model in their paper. While this is largely due to the author's lack of knowledge of the NetLogo programming language used by Janssen and Rollins, it provides a stronger test of the conceptual description in the original paper, as well as a test of whether the original implementation of the model is equivalent to the conceptual model. If the replication is successful, it can be said with confidence that the original authors' description captures all essential features of the model. Likewise, if the replication model produces results equivalent to the original implementation without referral to the original code, it gives some additional degree of verification that the original implementation is a correct representation of the conceptual model.

3.2 Results

The replication model was run with the default parameter values as outlined in Table two in Janssen's and Rollins' paper (p. 225), specifically for values of α and β with $-1 \leq \beta \leq \alpha \leq 1$ and $\eta = 0.5$ for 50 iterations as in the original model. First, only the asymmetric commons dilemma case was considered.

The implementation of the replication model is described by the pseudo-code in Fig. 3.1. The full python code is provided in appendix A.

The results are shown in Fig. 3.2, following the presentation in Janssen's and Rollins' Figures two, three, and four (pp. 225f).

Clearly, these results are not equivalent to the original results. The level of common public resource (CPR) produced as well as the gini coefficients for investments and extractions between the agents show no pronounced sensitivity to the parameters α and β, which is the case in the results presented by Janssen and Rollins.

The possibility that this difference is due to the exclusion of the linear public goods game was explored. Janssen and Rollins specify a linear public goods game to be played with some frequency $1 - f_a$ instead of the asymmetric commons

Create n agents with parameters α, β and η

- Loop through a specified number of iterations
  - Loop through $c = 10$ rounds
    - Let each agent make a probabilistic investment choice as specified in equation (2.3)
    - Calculate the total CPR resulting from the agents' investment as specified in Table one (p. 223) in Janssen's and Rollins' paper
    - Let the agents make a probabilistic extraction choice as specified in equation (2.4); starting with the agent at position 0, then positions 1, 2, 3 and 4, respectively

Figure 3.1: Pseudo-code for the replication model

dilemma. The exact value of f_a is not specified in the paper, but if the results of the linear public goods game prove to be sensitive to α and β, the results of a mixture between asymmetric commons and public good games would be closer to the results reported by Janssen and Rollins. Fig. 3.3 shows that this is not the case [2]. For the python code, refer to appendix A.

[2]For the linear public goods game, the amount of CPR generated depends on the parameter r, the value of which is not specified by the authors. However, the results of the public good game do not qualitatively change with different values for r. For the results shown, $r = 2$ was used.

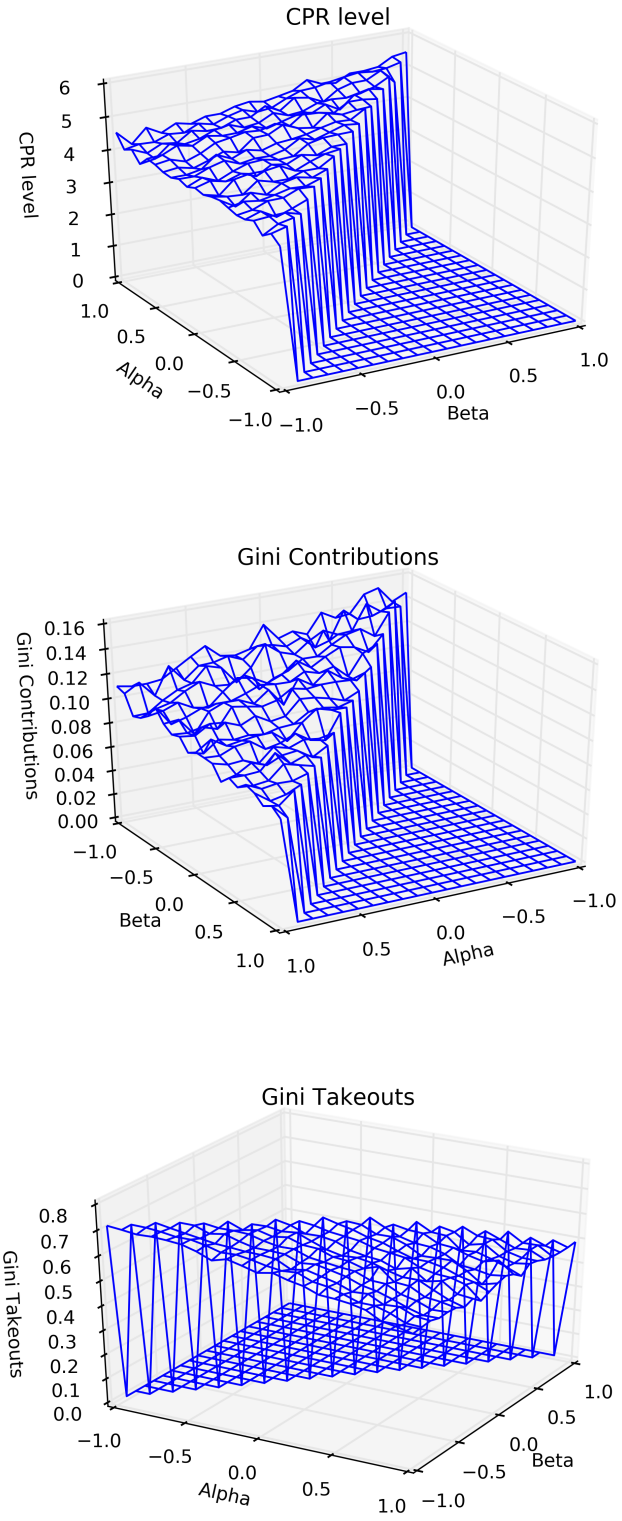

Figure 3.2: Replication model results for the asymmetric commons dilemma

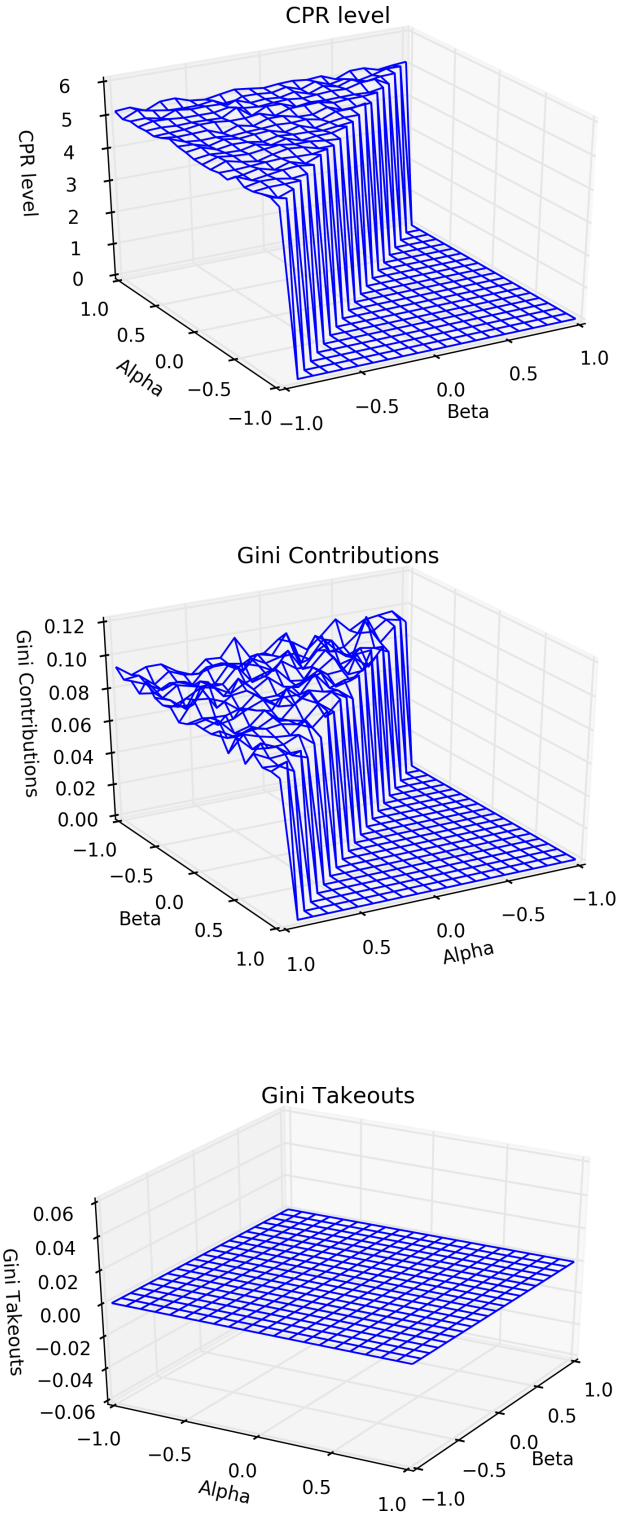

Figure 3.3: Replication model results for the linear public goods game

3.2.1 Exploring the Space of Possible Outcomes for the Probabilistic Choice Algorithm

In the conceptual model described by Janssen and Rollins, agents' investment decisions are based on the probabilistic choice algorithm (equation four in the original paper, p. 224):

$$P(x_{i,k}) = \frac{e^{\eta \cdot E[u(x_{i,k}|x_{j,k,t-1};y_{j,k,t-1})]}}{\sum_{m=k}^{M} e^{\eta \cdot E[u(x_{m,k}|x_{j,k,t-1};y_{j,k,t-1})]}} \tag{3.1}$$

The agents' utility is calculated as

$$u_i = w_i - \alpha_i \cdot max(w_i - \bar{w}, 0) + \beta_i \cdot max(\bar{w} - w_i, 0) \tag{3.2}$$

(equation three in the original paper, p. 224), where

$$w_i = (1 - x_i) + y_i \cdot y_{i-1} \cdots_1 \cdot g \tag{3.3}$$

(equation two in the original paper, p. 224) with y as the share extracted from the resource g in the allocation phase. The average wage $\bar{w}$ can be defined as

$$\bar{w} = \frac{1}{5} \cdot [5 - \sum_{i=1}^{5} x_i + g(\sum_{i=1}^{5} x_i)] \tag{3.4}$$

with $g(x)$ as the CPR generated for a given amount of the sum of contributions x of each player in the investment phase (refer to Table one in the original paper, p. 223). Plugging in equations (3.3) and (3.4) into (3.2) yields, for agent i,

$$u_i = (1-x_i)+y_i \cdot \prod^{j} y_j \cdot g(x_i+\sum^{j} x_j) - \alpha_i \cdot max(w_i-\bar{w},0) + \beta_i \cdot max(\bar{w}-w_i,0) \tag{3.5}$$

(3.3), (3.4) and (3.5) can be used to rewrite (3.1) as

$$P(x_{i,k}) = \frac{e^{\eta \cdot (1-x_i)+\prod^{j} y_j \cdot g(x_i+\sum^{j} x_j) - \alpha_i \cdot max(w_i-\bar{w},0)+\beta_i \cdot max(\bar{w}-w_i,0)}}{\sum_{m=k}^{M} e^{\eta \cdot (1-x_m)+\prod^{j} y_j \cdot g(x_i+\sum^{j} x_j) - \alpha_i \cdot max(w_i-\bar{w},0)+\beta_i \cdot max(\bar{w}-w_i,0)}} \tag{3.6}$$

With equation (3.6), probabilities for the 10 investment levels can be calculated for any given $\sum^{j} x_j$ and $\prod^{j} y_j$. Multiplying the probabilities with the investment level, an expected value $E(x_i \mid \sum^{j} x_j, \prod^{j} y_j)$ can be obtained.
Possible values for $\sum^{j} x_j$ range between 0 and 4 in 0.1 increments. Possible values

for $\prod^j y_j$ range between 0 and 1 in arbitrarily small increments, but using increments of 0.01 likely gives a good approximation.
This allows to calculate $E(x_i \mid \sum^j x_j, \prod^j y_j)$ for all possible combinations of $\sum^j x_j$ and $\prod^j y_j$, regardless of whether or not they could actually occur in the irrigation game model. This means no expected values for x_i outside of these values can possibly occur in the model.
This can be done using the algorithm

- Loop through all possible values for $\sum^j x_j$ in 0.1 increments
  - Loop through all possible values for $\prod^j y_j$ in 0.01 increments
    - calculate $E(x_i \mid \sum^j x_j, \prod^j y_j)$

For the python code, refer to appendix B.
Graphing the results for different values of η, α, and β

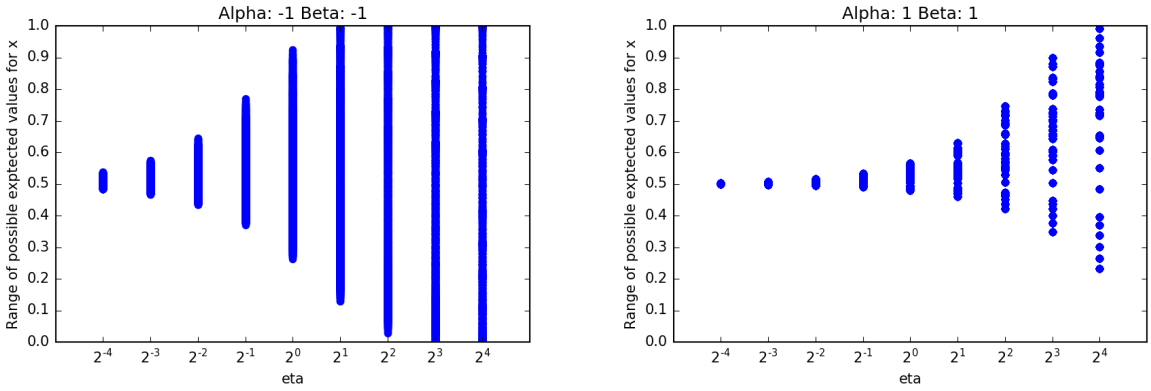

Figure 3.4: Results for possible expected values of contributions

Figure 3.4 shows that for $\alpha = \beta = -1$ and $\eta = 0.5$, the expected contribution value never drops below about 0.35. The expected total contribution value for all five players is thus never below 1.75. For a high number of iterations as used in both the original and replication model, the lower bound of contributions should not be significantly lower than the lower bound of expected contributions. Respectively, for $\alpha = \beta = 1$ and $\eta = 0.5$, the expected contribution value never exceeds 0.55, resulting in an expected total contribution of 2.75.
Since $g(1.75) = 2$ and $g(2.75) = 6$, [3] the values depicted in Janssen and Rollins' result graph (see Figure two in the original paper, p. 225) of below 0.25 for $\alpha = \beta = -1$ and about 7 for $\alpha = \beta = 1$ do not result from the probabilistic choice model (eq. 3.1) if $\eta = 0.5$ is used. This suggests why the results of the original and the replication model are not equivalent.

[3] refer to Table one in Janssen and Rollins' paper (p. 223)

3.2.2 Results for Larger Values of Eta

Experimenting with different values for η, a shape of the result graph more similar to the original results was reached for $\eta = 15$, i.e., 30 times the default value for η used in the paper (Fig. 3.5). For the python code, refer to appendix A.
For this result to occur, an initial value of 0 for the expected level of cooperation of the other agents had to be set[4]. This suggests that the results presented in the original paper were obtained with a different value for η than the default of $\eta = 0.5$. The mathematical result described in section 3.2.1 suggests that the results presented in the original paper are unlikely to result from the described algorithm if $\eta = 0.5$ is being used.

Using fixed values for the parameters α and β, the generated amount of CPR for different values for η can be explored. For $\alpha = \beta = -1$, the original result (Figure two in the original paper, p. 225) shows a CPR level of 0. For $\alpha = \beta = 1$, the original CPR value is just below 7. In between these two extremes, Janssen's and Rollins' result chart shows a CPR level of 2 for $\alpha = 1$ and $\beta = -0.2$.

Fig. 3.6 shows, on a base 2 logarithmic scale, $\eta = 15$ to be somewhat fitting - although there is no exact match except for the $\alpha = \beta = -1$ case. $\eta = 2^{-1} = 0.5$ is not a good approximation in any of the three cases.

3.2.3 Conclusion

Comparing the results in Fig. 3.5 to figures two, three and four in the paper by Janssen and Rollins (pp. 225f) shows a high degree of similarity in the shapes of the result graphs.
As for the gini coefficients of investments and extractions between agents (figures three and four in the original paper, p. 226; middle and bottom image in Fig. 3.5), there seems to be little difference between original and replication model. Except for a more smooth surface in the original graph[5], the graph shapes of original and replication are essentially identical. There also seems to be little difference in the absolute values - although it is not easy to discern exact values in figures three and four (p. 226) in the original paper for all parameter values, original and replication results have very similar minimum and maximum values.
In case of the generated CPR level, the two graphs (Fig. 3.5 top and figure two in the original paper, p. 225) also are similarly shaped. Both graphs are strictly

[4]The initial value for the expected cooperation of the other agents is not specified by Janssen and Rollins. For the case of $\eta = 0.5$, using different values for the expected cooperation does not change the results significantly.

[5]which is also reached in the replication model if more than 50 iterations are run

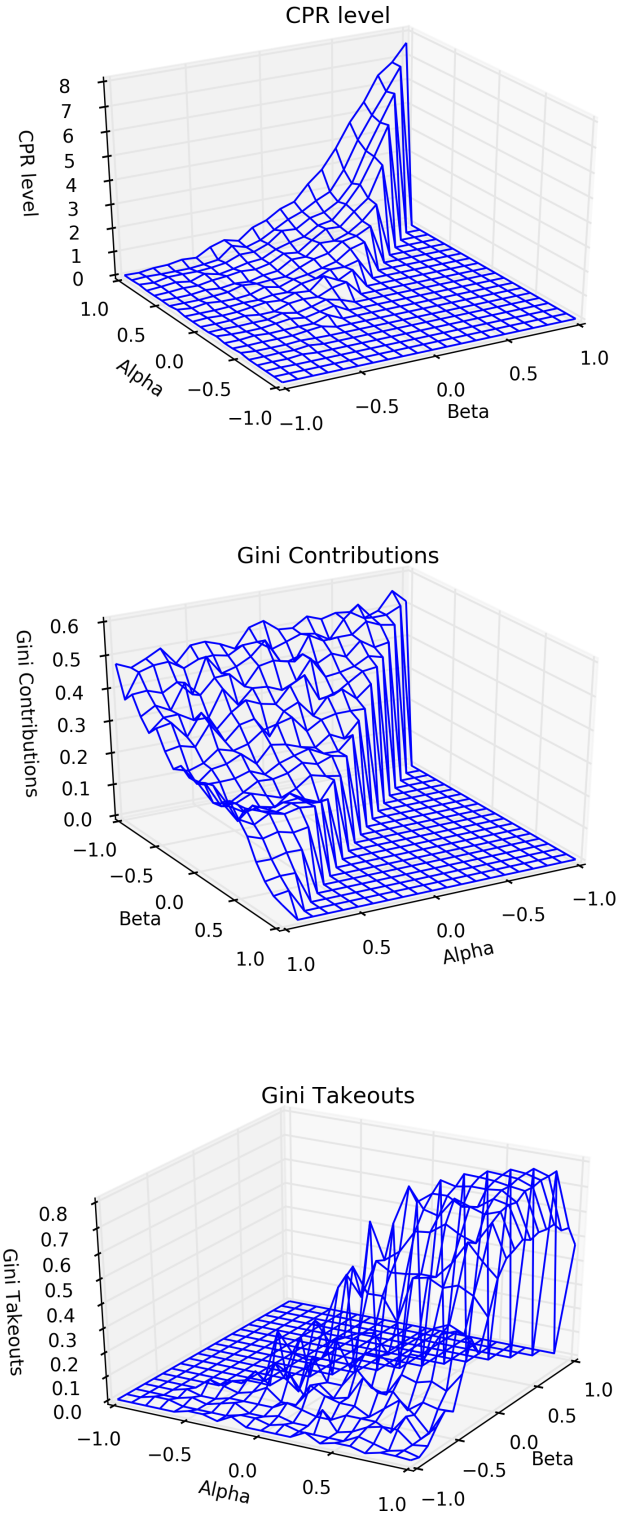

Figure 3.5: Replication results for eta = 15

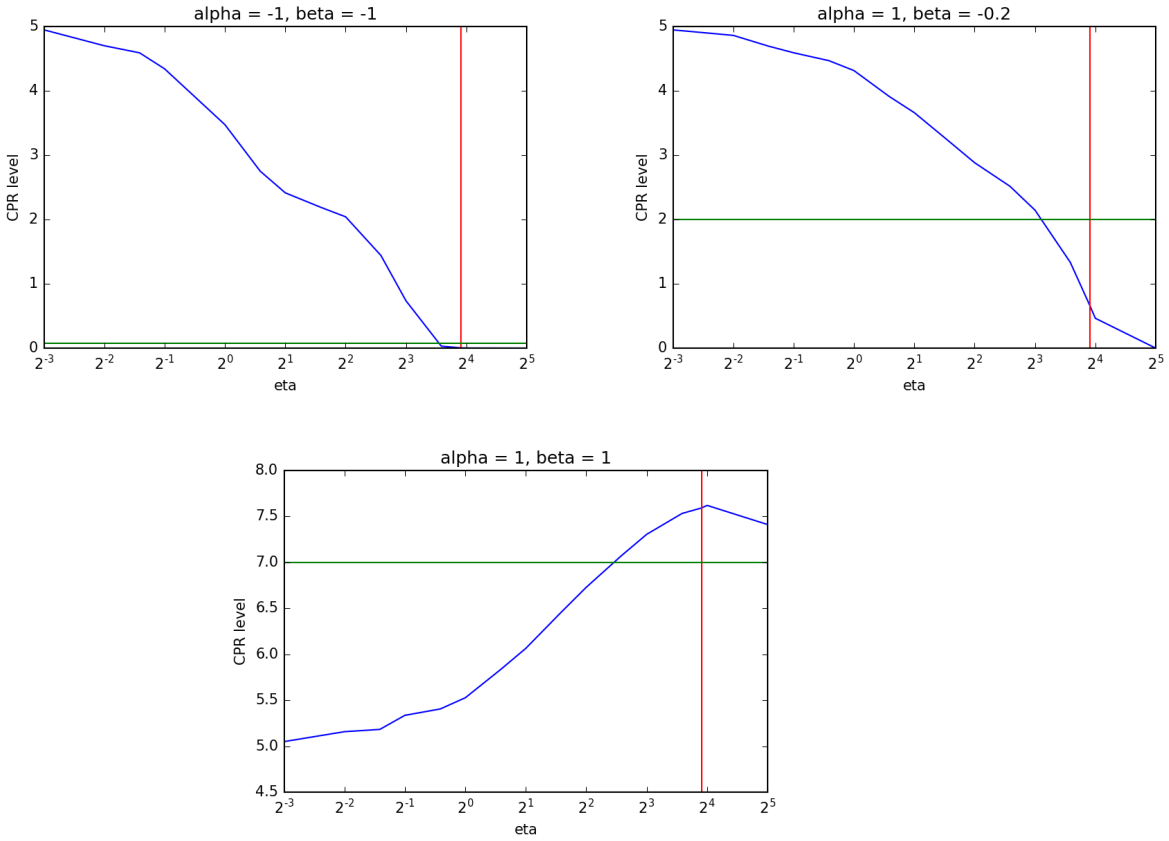

Figure 3.6: Average CPR levels for different values of eta, vertical red line: eta = 15, horizontal green line: original result. Run for 500 iterations for each value of eta.

increasing in β, from a CPR level value of zero for $\beta = -1$. As shown in Fig. 3.6, the CPR values of the original model are not matched exactly by the replication model - the maximum value, at $\alpha = \beta = 1$, is slightly above 7.5, instead of 7.0 in the original model. For the "middle point" of $\alpha = 1$ and $\beta = -0.2$, the replication model gives a result of about 0.5 instead of 2 in the original result. The generated CPR levels of the replication model are generally lower than the original results for most values except $\beta \approx 1$.[6]

While the results are not numerically identical, they can be said to satisfy the criterion of relational equivalence: the outputs of both original and replication model are sensitive to the parameters α and β, and move in the same direction when these parameters are varied, with similar absolute values.

As elaborated in section 3.2.2, however, this relational equivalence of replication and original model is reached only for $\eta = 15$, not for the default value of $\eta = 0.5$.

[6]Refer to appendix A

3.3 Further Explorations

Janssen and Rollins present experimental data for contributions and extractions for different agent positions in the experimental setup in figure one of their paper (p. 222). Even though Janssen and Rollins provide no directly equivalent figure for their computational model, figures six and nine (pp. 227 and 228, respectively) can be jointly used to obtain shares of extraction from the CPR for different parameters.

As can be seen in Janssen's and Rollins' figure nine (p. 228), for a frequency of asymmetric games f_a of 2/10, the agent at the head-end position (position A) extracted about half of the total CPR. This agent's share drops to just about 30% for $f_a = 0.8$ and increases back to 50% when only asymmetric games are played (i.e., $f_a = 1$). The corresponding average values for α and β that evolved are given in Janssen's and Rollins's figure six (p. 227). For $f_a = 0.2$, the respective values are $\alpha \approx 0.8$ and $\beta \approx 0.5$; for $f_a = 0.8$, $\alpha \approx 0.2$ and $\beta \approx -0.5$; and for $f_a = 1$, values of $\alpha \approx 0$ and $\beta \approx -0.6$ are shown.

The aligned replication model with $\eta = 15$ can be used to see if similar results are obtained. For this part of the replication, the model was run for 5000 iterations in each case. The results are presented in Fig. 3.7. The model was run with asymmetric games being played only, i.e. $f_a = 1$.

In the experimental results presented by Janssen and Rollins, the investment is invariant to the participant's position, with values of about 5 tokens for all players (0.5 units in the model). The experimental data also show that for all except the most downstream position, players' extractions are on average higher than their contributions.

This is clearly not the case for the replication model with $\alpha = 0.85$ and $\beta = 0.4$, where only the agents at position 0 and 1 receive any extractions, almost all of which is extracted by the agent at the most upstream position. Fig. 3.7 top left shows that for $\alpha = 0.85$ and $\beta = 0.4$, the investments are declining the higher an agent's relative position is.[7]

Increasing α to 1 and β to 0.75, the distribution of investments gets more even, and with values of about 0.5 closer to the experimental data (Fig. 3.7 top right). For this setting, all players receive some share of the CPR, with a positive net extraction value for agents on position 0 and 1. The distribution closest to the experimental results is reached for $\alpha = \beta = 1$ (Fig 3.7 bottom). The three most upstream players have a positive net extraction, with stable investment values on

[7] I.e., the more "downstream" an agent's position is.

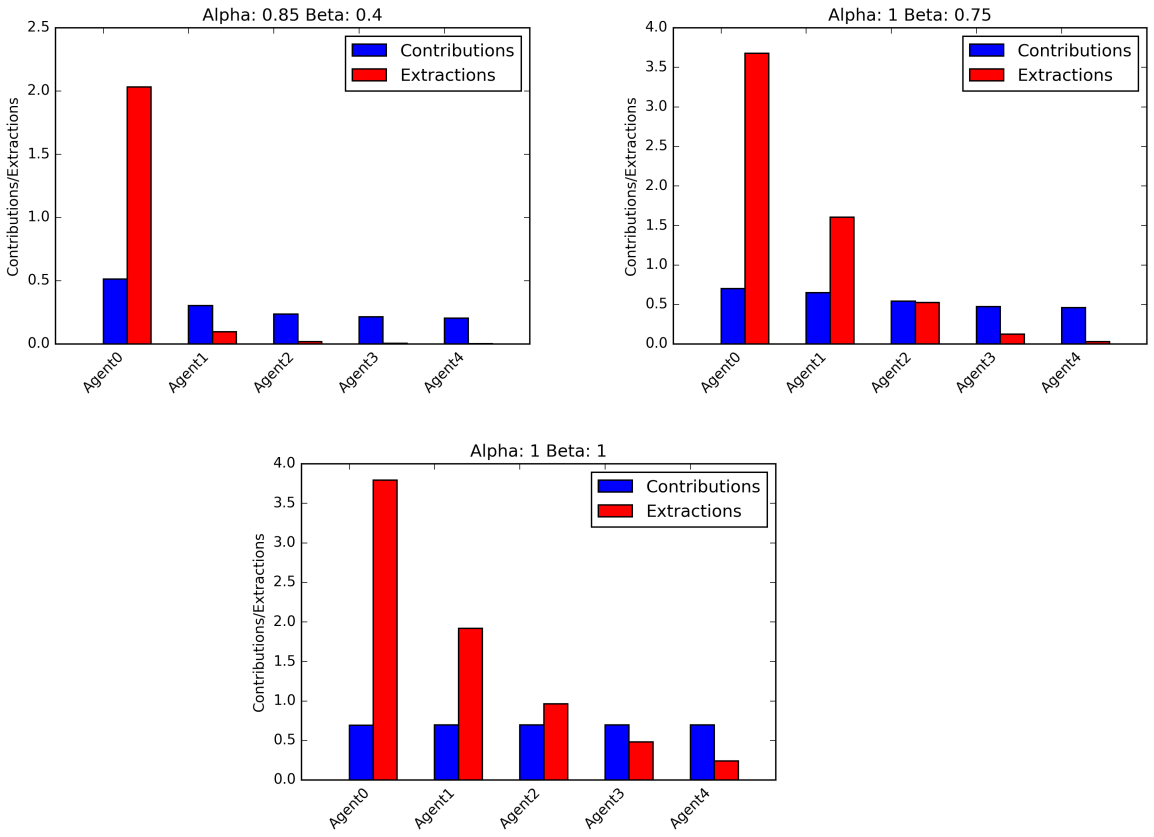

Figure 3.7: Average distribution of investments and extractions per round between the agents. Since monetary units in the computational model are normalized to one, one unit in this graph is equal to ten tokens in Janssen's and Rollins' figure one (p. 222).

all five positions of above 0.5.

In none of the three parameter settings, the rather even distribution of extractions and positive net extractions for four of the five players in the experimental results are reached in the results of the replication model. Considering the close similarity of gini coefficients for extractions and investments in original and replication model (see Fig. 3.5 middle and bottom as well as figures three and four in the original paper, p. 226), this is likely the case for the original model, too. However, as noted by the original authors, it is not their intent to "model the actual contemporary play of the game, but the cultural-evolutionary processes that may lead artificial agents" to behave similarly to the experimental results (p. 223).

3.4 Attempting to Replicate the Cultural Selection Model

Despite not being in the original scope for the replication, exploring how the replication model behaves when extended to include the cultural selection process could provide additional insights into the workings of the model. The full model is described by the following pseudo-code, with the python code provided in Appendix A.

Set default parameters $\alpha, \beta, \eta, c, m, \epsilon, \sigma$ and f_a.
Create a specified number of groups (default: 100) with $n = 5$ agents with default parameters α, β and η.
Loop through a specified number of rounds (default: 3000).
Loop through each group i

- draw $\delta \in B(1, f_a)$
- If $\delta = 1$: Loop through $c = 10$ steps of asymmetric commons dilemma games
    - Let each agent make a probabilistic investment choice as specified in equation (2.3)
    - Calculate the total CPR resulting from the agents' investment as specified in Table one in Janssen's and Rollins' paper (p. 223)
    - Let the agents make a probabilistic extraction choice as specified in equation (2.4); starting with the agent at position 0, then positions 1, 2, 3 and 4
- If $\delta = 0$: Loop through $c = 10$ steps linear public good games
    - Let each agent make a probabilistic investment choice as specified in equation (2.3)
    - Calculate the total CPR resulting from the agents' investment as specified in Table one in Janssen's and Rollins' paper (p. 223) and distribute equally among agents

- Add a noise value to each agent's α and β parameter separately, randomly drawn from a normal distribution $n(0, \sigma)$ for each agent.
  If $\alpha > 1$ for any agent, set $\alpha = 1$ for that agent. If $\beta > \alpha$ for any agent, set $\beta = \alpha$ for that agent. If $\alpha < -1$ or $\beta < -1$ for any agent, set $\alpha = -1$ or $\beta = -1$, respectively, for that agent.

- For each agent, draw $\gamma \in B(1, m)$

  - If $\gamma = 1$:

    - Randomly choose another group j
    - Randomly match an agent from group j to each agent in group i
    - Replace the group i agent's α, β with the α, β values of the agent from group j with probability p as defined in equation 2.6

  - If $\gamma = 0$:

    - randomly match an agent j from the same group to each agent i
    - Replace agent i's α, β with the α, β values of the agent j with probability p as defined in equation 2.6

- Draw $\theta \in B(1, \epsilon)$

  - If $\theta = 1$:

    - Randomly choose another group j
    - Calculate imitation probability P according to equation 2.7
    - Draw $\lambda \in B(1, P)$
    - If $\lambda = 1$:

      - Match agents from group i to agents from group j at the same relative position
      - Replace α, β values of agents from group i by α, β values of agents from group j

    - If $\lambda = 0$:

      - Match agents from group i to agents from group j at the same relative position
      - Replace α, β values of agents from group j by α, β values of agents from group i

The model as described above was run for 3000 rounds of $c = 10$ steps for 15 iterations. Default parameter values were $\alpha = \beta = 0$, thus starting with selfish agents, and $\eta = 15$, following the description presented above.
The results are compared to figures six and seven in Janssen's and Rollins' paper (p. 227), depicting the average results for α and β and generated CPR, respectively, for different frequencies f_a of asymmetric games played. Results are presented in Fig 3.8.

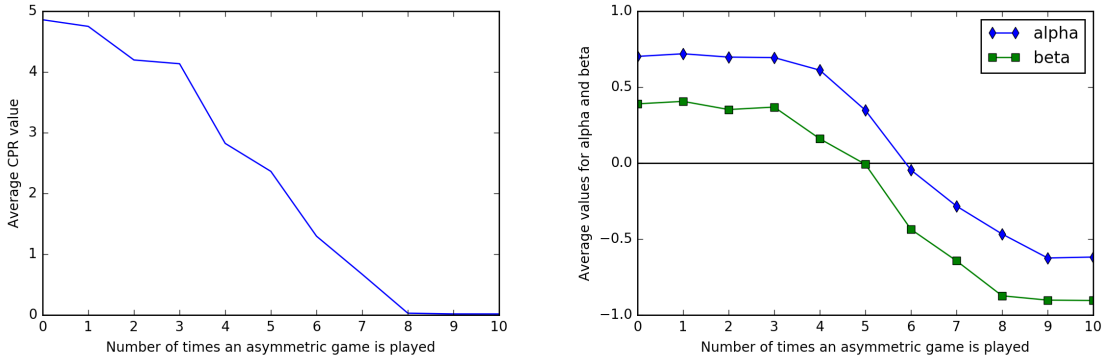

Figure 3.8: Average value of evolved CPR, as well as alpha and beta, for different frequencies of asymmetric games out of 10 rounds

Fig. 3.8 left shows that the evolved average value of CPR generated is sensitive to the frequency of asymmetric games being played, with the average amount of generated CPR decreasing in f_a. Compared to Janssen's and Rollins' figure seven (p. 227), the values of generated CPR are generally lower. Whereas in the original model, the values peak at about 6.5 for $f_a = 0$, with a generated CPR of just below 4 for $f_a = 0.5$; results for the replication model are just below 5 for $f_a = 0$ and about 2.5 for $f_a = 0.5$. In the replication model, no CPR is generated for eight or more asymmetric dilemmas out of ten games. In the original model, the value of the common pool drops to zero for $f_a = 10/10$; with low, but positive CPR values for $f_a = 0.8$ and $f_a = 0.9$. While the results from original and replication model are clearly not numerically identical; their internal relationship with cooperation decreasing in the frequency of asymmetric games is similar. In both cases, very little or no resource generation evolves for $f_a > 0.7$.
As the amount of investment into the CPR is a function of the agents' social preferences, results for the average values of α and β that evolved in the original and replication model have a similar relationship to the CPR levels. Values for α and β decrease in f_a in both models, with the evolved α and β being generally lower in the replication than in the original model. Evolved βs in both models

reach zero at $f_a = 0.5$, with negative values for $f_a > 0.5$. Average values for α are positive for all f_a in the original model; but not so in the replication model, where they drop to around -0.6 for $f_a = 1.0$.

Despite significant differences in the numerical values, a similar evolutionary dynamic appears to take place in original and replication model. In both cases, cooperative behavior evolves only where linear public good games are played with some frequency. The higher the frequency of asymmetric games being played, the less likely any form of cooperation between agents is to evolve.

Chapter 4

Concluding Remarks

An important feature of agent-based models is that complex outcomes emerge from relatively simple agents - in ways that the researcher did not necessarily anticipate. As Epstein [12] notes, an "experimental attitude" (p. 51) is required. In a way, the conceptual description of an agent-based model could be compared to the description of a more traditional experiment, while the concrete implementation in a particular programming language produces the actual experimental results. In both cases, "traditional" and "agent-based" experiments, independent replications play a vital role in ensuring that the experiment is valid.

Many researchers utilizing agent-based models encountered by this author rely on a verbal description of their conceptual model. However, these descriptions are necessarily somewhat fuzzy; as discussed above, the verbal description of a conceptual model does not necessarily align with its implementation even by the original researcher. A completely accurate verbal description of a computational model would have to be written with almost excessive precision, which is not necessarily the best way of presenting a model in a research paper. The only "failure-proof" description of the workings of a model is the code of its implementation. This poses the problem that since a multitude of programming languages exist, most of them are not known by any given individual researcher.

Pseudo-Code, as utilized in describing the replication models above, could alleviate the problem as it allows more accuracy than a mere verbal description, yet also does not require knowledge of any particular programming language. It appears, however, not to be in universal use in research papers on agent-based models.

As noted by Tesfatsion [35], "programming skills remain [...] problematic for economists because few graduate economic programs currently have computer programming requirements" (p. 865). From the personal experience of the author as an undergraduate student of economics, this remains true ten years after Tes-

fatsion's remark. Certainly, few of the economic models taught in undergraduate economic courses appear to draw insights from computational models. As Tesfatsion remarks, "programming frees us to adapt the tool to the problem rather than the problem to the tool" - for him, "it is time" that computational models get to play a larger role in economic modeling (p. 866).

Chapter 5

Summary

Irrigation systems provide a real-world example of Common Public Resources (CPR) with asymmetric access to the resource. Acknowledging this, Janssen, Bousquet, Cardenas, Castillo and Worrapimphong [21] devised an abstracted "irrigation game" which captures salient features of irrigation systems and collected experimental data on how human participants behave when playing this game. The agent-based model of Janssen and Rollins [22] uses this "irrigation game" as the basic structure, with computational agents interacting in the same fashion as participants in the experimental setup. Janssen and Rollins then applied an evolutionary selection process to determine how agents' preferences could evolve to produce cooperative behavior. They find that in their model, cooperative behavior does not evolve when agents are engaged in solely asymmetric interactions. A mix between symmetric and asymmetric CPR dilemmas played successively by the same group of agents, however, causes agents to evolve towards cooperative behavior with high probability.

The present work constitutes an effort to independently replicate Janssen's and Rollins' findings. While the basic structure of the model and the algorithms determining agent behavior were adopted unmodified; the programming language as well as the framework in use were altered, and the replication model was designed without referral to the original code.

For the default parameter values as described by Janssen and Rollins, no successful replication is reached. It is shown that Janssen's and Rollins' findings are unlikely to result from the default parameters specified in their paper as it is understood by this author. Indeed, when one of the default parameters is changed, the replication model produces results which are qualitatively similar to the original findings and satisfy the replication standard of relational equivalence. For the altered default parameters, the replication model also shows evolving cooperative

behavior when the evolutionary selection process is applied; as in the original model, this finding only holds for successive play of symmetric and asymmetric CPR games.

Bibliography

[1] Toh-Kyeong Ahn, Myungsuk Lee, Lore Ruttan, and James Walker. Asymmetric payoffs in simultaneous and sequential prisoner's dilemma games. *Public Choice*, 132(3-4):353–366, 2007.

[2] Robert Axtell, Robert Axelrod, Joshua M. Epstein, and Michael D. Cohen. Aligning simulation models: A case study and results. *Computational & Mathematical Organization Theory*, 1(2):123–141, 1996.

[3] Martin Beckenkamp, Heike Hennig-Schmidt, and Frank P Maier-Rigaud. Cooperation in symmetric and asymmetric prisoner's dilemma games. *Preprints of the Max Planck Institute for Research on Collective Goods*, (2006/25), 2007.

[4] Helmut Bester and Werner Güth. Is altruism evolutionarily stable? *Journal of Economic Behavior & Organization*, 34(2):193–209, 1998.

[5] Robert Boyd, Herbert Gintis, Samuel Bowles, and Peter J Richerson. The evolution of altruistic punishment. *Proceedings of the National Academy of Science of the United States of America*, 100(6):3531–3535, 2003.

[6] Thomas Brenner. Agent learning representation: Advice on modelling economic learning. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 18, pages 895–947. Elsevier, 2006.

[7] Thomas Dietz, Nives Dolšak, Elinor Ostrom, and Paul C Stern. The drama of the commons. In Thomas Dietz, Nives Dolšak, Elinor Ostrom, Paul C Stern, Susan Stonich, and Elke U. Weber, editors, *The Drama of the Commons*, chapter 1, pages 3–35. National Academy Press, Washington, DC, 2002.

[8] John Duffy. Agent-based models and human subject experiments. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 19, pages 949–1011. Elsevier, 2006.

[9] Bruce Edmonds and David Hales. Replication, replication and replication: Some hard lessons from model alignment. *Journal of Artificial Societies and Social Simulation*, 6(4), 2003.

[10] Daniel Eek and Anders Biel. The interplay between greed, efficiency, and fairness in public-goods dilemmas. *Social Justice Research*, 16(3):195–215, 2003.

[11] Daniel Eek, Anders Biel, and Tommy Gärling. Cooperation in asymmetric social dilemmas when equality is perceived as unfair. *Journal of Applied Social Psychology*, 31(3):649–666, 2001.

[12] Joshua M Epstein. Agent-based computational models and generative social science. *Complexity*, 4(5):41–60, 1999.

[13] Joshua M Epstein and Robert Axtell. *Growing Artificial Societies: Social Science from the Bottom up.* Brookings Institution Press, 1996.

[14] Gary William Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation.* MIT press, 1998.

[15] Kent V Flannery. The ecology of early food production in mesopotamia. *Science*, 147(3663):1247–1256, 1965.

[16] Roy Gardner, Elinor Ostrom, and James M Walker. The nature of common-pool resource problems. *Rationality and Society*, 2(3):335–358, 1990.

[17] Garrett Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, 1968.

[18] Christoph Hauert and György Szabó. Prisoner's dilemma and public goods games in different geometries: Compulsory versus voluntary interactions. *Complexity*, 8(4):31–38, 2003.

[19] Robert C Hunt. Size and structure of authority in canal irrigation systems. *Journal of Anthropological Research*, 44(4):335–355, 1988.

[20] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[21] Marco A Janssen, François Bousquet, Juan-Camilo Cardenas, Daniel Castillo, and Kobchai Worrapimphong. Field experiments on irrigation dilemmas. *Agricultural Systems*, 109:65–75, 2012.

[22] Marco A Janssen and Nathan D Rollins. Evolution of cooperation in asymmetric commons dilemmas. *Journal of Economic Behavior & Organization*, 81(1):220–229, 2012.

[23] Friedrich Krebs, Sascha Holzhauer, and Andreas Ernst. Modelling the role of neighbourhood support in regional climate change adaptation. *Applied Spatial Analysis and Policy*, 6(4):305–331, 2013.

[24] J Stephen Lansing and James N Kremer. Emergent properties of balinese water temple networks: Coadaptation on a rugged fitness landscape. *American Anthropologist*, 95(1):97–114, 1993.

[25] John Stephen Lansing. *Priests and Programmers: Technologies of Power in the Engineered Landscape of Bali*. Princeton University Press, 1991.

[26] David Masad and Jacqueline Kazil. Mesa: An Agent-Based Modeling Framework. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 53 – 60, 2015.

[27] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[28] Cho Nam Ng, Raymond Yu Wang, and Tianjie Zhao. Joint effects of asymmetric payoff and reciprocity mechanisms on collective cooperation in water sharing interactions: A game theoretic perspective. *PloS One*, 8(8):1–11, 2013.

[29] Mancur Olson. Dictatorship, democracy, and development. *American Political Science Review*, 87(03):567–576, 1993.

[30] Elinor Ostrom and Roy Gardner. Coping with asymmetries in the commons: Self-governing irrigation systems can work. *The Journal of Economic Perspectives*, 7(4):93–112, 1993.

[31] Elinor Ostrom, Roy Gardner, and James Walker. *Rules, Games, and Common-Pool Resources*. University of Michigan Press, 1994.

[32] Amy R Poteete, Marco A Janssen, and Elinor Ostrom. *Working Together: Collective Action, the Commons, and Multiple Methods in Practice*. Princeton University Press, 2010.

[33] Karolina Safarzynska. The coevolution of culture and environment. *Journal of Theoretical Biology*, 322:46–57, 2013.

[34] Fangfang Tan. Punishment in a linear public good game with productivity heterogeneity. *De Economist*, 156(3):269–293, 2008.

[35] Leigh Tesfatsion. Agent-based computational economics: A constructive approach to economic theory. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 16, pages 831–880. Elsevier, 2006.

[36] Bjorn Van Campenhout, Ben D'Exelle, and Els Lecoutere. Equity–efficiency optimizing resource allocation: The role of time preferences in a repeated irrigation game. *Oxford Bulletin of Economics and Statistics*, 77(2):234–253, 2015.

[37] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13(2):22–30, 2011.

[38] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[39] Uri Wilensky and William Rand. Making models match: Replicating an agent-based model. *Journal of Artificial Societies and Social Simulation*, 10(4), 2007.

[40] Henk AM Wilke. Greed, efficiency and fairness in resource management situations. *European Review of Social Psychology*, 2(1):165–187, 1991.

[41] Stephen Wolfram. Cellular automata as models of complexity. *Nature*, 311(5985):419–424, 1984.

Appendix A

Code for the Replication Model

Note: All Code in the appendix written in Python 3.5.
For easy replication of the figures presented above, the respective variable at the start of the code can be set to "1". Calculations for Fig. 3.8 are highly computationally intensive and took up to a day on a standard laptop. For Fig. 3.8, the average of fifteen runs was taken.
Note that due to the model including a stochastic element, results from different runs are never exactly identical.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 3 12:38:36 2016

@author: Joe
"""

import random
import math
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mesa import Model, Agent
from itertools import product
from mpl_toolkits.mplot3d import *


"""
Reproduction: Set the corresponding variable to "1" to reproduce
    results for
one of the figures in the Thesis. Please set all other of the variables
    to "0".
"""
```

```
figure3_2 = 1
figure3_3 = 0
figure3_5 = 0
figure3_6 = 0
figure3_7 = 0
figure3_8 = 0 #Note: Computationally intensive

if figure3_2 == 1:
    print("Reproducing Fig. 3.2")
elif figure3_3 == 1:
    print("Reproducing Fig. 3.3")
elif figure3_5 == 1:
    print("Reproducing Fig. 3.5")
elif figure3_6 == 1:
    print("Reproducing Fig. 3.6")
elif figure3_7 == 1:
    print("Reproducing Fig. 3.7")
elif figure3_8 == 1:
    print("Reproducing Fig. 3.8")

figurecheck = figure3_2 + figure3_3 + figure3_5 + figure3_6 + figure3_7
    + figure3_8
if figurecheck > 1:
    sys.exit("Please select only one figure")

"""
Begin of Actual Model
"""

#Agent Class, generated by the Model class
class Resource_Agent(Agent):

#Initial Conditions
    def __init__(self, unique_id, alpha, beta, eta):
        self.unique_id = unique_id
        self.score = 0 #accumulating score
        self.subscore = 0 #score of this round only
        self.endowment = 1
        self.contribution = 0#(random.randrange(0, 11, 1)) / 10
        self.takeout_share = (random.randrange(0, 11, 1)) / 10
        self.alpha = alpha
        self.beta = beta
        self.eta = eta
        self.past_contributions = [self.contribution]
        self.past_takeout_shares = [self.takeout_share]
        self.total_contribution = 0
        self.total_amount_taken_out = 0

#Resource Provision Phase
    def first_round(self, model):
```

```
        self.endowment = 1
        #self.subscore = 0
        self.contribution = ((np.random.choice(11, p =
            self.investment_p(model))) / 10)
        self.score += (self.endowment - self.contribution)
        self.subscore += (self.endowment - self.contribution)
        self.past_contributions.append(self.contribution)
        self.total_contribution += self.contribution
        model.sum_of_contributions += self.contribution

#Resource Allocation Phase
    def second_round(self, model):
        takeout_share = ((np.random.choice(11, p =
            self.extraction_p(model))) / 10)
        takeout = round(model.resource * takeout_share, 1)
        actual_takeout_share = round((takeout / (model.resource +
            0.000000001)), 1)
        self.takeout_share = actual_takeout_share
        if takeout_share <= 0:
            takeout_share = 0
        self.past_takeout_shares.append(actual_takeout_share)
        self.score += takeout
        self.subscore += takeout
        model.resource -= takeout
        if model.resource < 0:
            model.resource = 0
        self.total_amount_taken_out += takeout

#Variant of the Allocation phase for the linear public good game. The
    resource
#provision phase is the same in both types of games.
    def sym_second_round(self, model):
        takeout = model.equal_share
        self.takeout_share = 0.2
        self.past_takeout_shares.append(0.2)
        self.score += takeout
        self.subscore += takeout
        model.resource -= takeout
        if model.resource < 0:
            model.resource = 0
        self.total_amount_taken_out += takeout


#General functions, needed for the probabilistic choice calculations
    def get_id(self):
        return self.unique_id

    def determine_upstream_players(self, model):
        upstream_players = []
        for agent in model.schedule.agents:
```

```
            if agent.unique_id < self.get_id():
                upstream_players.append(agent)
        return upstream_players

    def determine_downstream_players(self, model):
        downstream_players = []
        for agent in model.schedule.agents:
            if agent.unique_id > self.get_id():
                downstream_players.append(agent)
        return downstream_players

    def last_round_sum_of_other_contributions(self, model):
        sum_of_contributions = 0
        for i in self.determine_upstream_players(model):
            sum_of_contributions +=
                i.past_contributions[model.steps_this_round]
        for j in self.determine_downstream_players(model):
            sum_of_contributions +=
                j.past_contributions[model.steps_this_round]
        return sum_of_contributions

    def last_round_upstream_agents_combined_takeout_shares(self, model):
        remaining_share = 1
        for i in self.determine_upstream_players(model):
            remaining_share = remaining_share * (1 -
                i.past_takeout_shares[model.steps_this_round])
        return remaining_share

    def this_round_upstream_players_combined_takeout_shares(self,
        model):
        remaining_share = 1
        for i in self.determine_upstream_players(model):
            remaining_share = remaining_share * (1 - i.takeout_share)
        return remaining_share

#Probabilistic Choice functions
    def investment_p(self, model):
        xj = self.last_round_sum_of_other_contributions(model)
        yj =
            self.last_round_upstream_agents_combined_takeout_shares(model)
        u_list = []
        for x in range(0, 11, 1):
            wi = 1 - x/10 + yj * model.created_resource(x/10 + xj)
            wavg = 1 - 0.2 * x/10 - 0.2 * xj + 0.2 *
                model.created_resource(x/10 + xj)
            u_of_x = wi - self.alpha * max(wi - wavg, 0) + self.beta *
                max(wavg - wi, 0)
            u_list.append(u_of_x)
            p_list = []
            denominator = 0
```

```
            for u in u_list:
                denominator += math.exp(self.eta * u)
            for u in u_list:
                    numerator = math.exp(self.eta * u)
                    p_list.append(numerator / denominator)
        return p_list

    def extraction_p(self, model):
        x = self.contribution
        xij = 0
        for i in model.schedule.agents:
            xij += i.contribution
        yj =
            self.this_round_upstream_players_combined_takeout_shares(model)
        u_list = []
        for y in range(0, 11, 1):
            wi = 1 - x + y/10 * yj * model.created_resource(xij)
            wavg = 1 - 0.2 * xij + 0.2 * model.created_resource(xij)
            u_of_x = wi - self.alpha * max(wi - wavg, 0) + self.beta *
                max(wavg - wi, 0)
            u_list.append(u_of_x)
            p_list = []
            denominator = 0
            for u in u_list:
                denominator += math.exp(self.eta * u)
            if denominator == 0:
                for u in u_list:
                    p_list.append(1/11)
            else:
                for u in u_list:
                    numerator = math.exp(self.eta * u)
                    p_list.append(numerator / denominator)
        return p_list

#Model Class
class Resource_Model(Model):

#Initial Conditions
    def __init__(self, N, alpha, beta, eta, fa, r, group_id = 1):
        self.num_agents = N
        self.alpha = alpha
        self.beta = beta
        self.eta = eta
        self.schedule = BaseScheduler(self)
        self.create_agents()
        self.resource = 0
        self.sum_of_contributions = 0
        # Variables needed for mesa's BatchRunner to work later
        self.total_produced_resource = 0
        self.running = True
```

```
        self.fa = fa # frequency of asymmetric games being played. Set
            to 1 for
        #only asymmetric games, and to 0 for only linear games being
            played.
        self.r = r #parameter in the resource generation function for the
        #linear games.
        self.equal_share = 0 #Determining the share each player receives
            in the
        #linear symmetric game.
        self.group_id = group_id
        self.this_round_produced_resource = 0
        self.steps_this_round = 0

    def create_agents(self):
        for i in range(self.num_agents):
            a = Resource_Agent(i, self.alpha, self.beta, self.eta)
            self.schedule.add(a)

    def step(self, asym):
        #asym = np.random.choice(2, p = [self.fa, (1 -
            self.fa)])#Determines
        #whether an asymmetric or linear game is being played.
        if asym == 0:
            self.schedule.step() #step function defined in mesa's
                BaseScheduler Class.
            #An asymmetric commons game is being played for this round.
        else:
            self.schedule.symstep() #Manually added function in mesa's
                BaseScheduler Class.
            #A linear public good game is being played in this round.
        self.total_produced_resource +=
            self.created_resource(self.sum_of_contributions)
        self.this_round_produced_resource +=
            self.created_resource(self.sum_of_contributions)
        self.sum_of_contributions = 0
        self.steps_this_round += 1

    def run_model(self, steps):
        asym = np.random.choice(2, p = [self.fa, (1 - self.fa)])
        self.this_round_produced_resource = 0
        self.steps_this_round = 0
        for a in self.schedule.agents:
            a.contribution = 0
            a.takeout_share = random.randrange(0, 11, 1) / 10
            a.past_contributions = [a.contribution]
            a.past_takeout_shares = [a.takeout_share]
            a.subscore = 0
        for i in range(steps):
            self.step(asym)
```

```
    def determine_generated_cpr(self): #Needed to collect CPR levels
        for runs with multiple iterations
        generated_cpr = self.total_produced_resource
        return generated_cpr


    def created_resource(self, sum_of_contributions): #resource function
        if sum_of_contributions >= 0 and sum_of_contributions < 1:
            produced_resource = 0
        elif sum_of_contributions >=1 and sum_of_contributions < 1.5:
            produced_resource = 0.5
        elif sum_of_contributions >=1.5 and sum_of_contributions < 2:
            produced_resource = 2
        elif sum_of_contributions >= 2 and sum_of_contributions < 2.5:
            produced_resource = 4
        elif sum_of_contributions >= 2.5 and sum_of_contributions < 3:
            produced_resource = 6
        elif sum_of_contributions >=3 and sum_of_contributions < 3.5:
            produced_resource = 7.5
        elif sum_of_contributions >=3.5 and sum_of_contributions < 4:
            produced_resource = 8.5
        elif sum_of_contributions >=4 and sum_of_contributions < 4.5:
            produced_resource = 9.5
        elif sum_of_contributions >=4.5 and sum_of_contributions <= 5:
            produced_resource = 10
        return produced_resource

#Needed to determine distribution of extractions between agent positions
    def determine_agent_total_takeout(self):
        takeout_list = []
        for i in self.schedule.agents:
            takeout = i.total_amount_taken_out
            takeout_list.append(takeout)
        return takeout_list

#Needed to determine distribution of investments between agent positions
    def determine_agent_total_contribution(self):
        contribution_list = []
        for i in self.schedule.agents:
            contribution = i.total_contribution
            contribution_list.append(contribution)
        return contribution_list


#BaseScheduler + BatchRunner: classes imported from mesa, modified as
    described
class BaseScheduler(object):
    model = None
    steps = 0
    time = 0
```

```
    agents = []

    def __init__(self, model):
        self.model = model
        self.steps = 0
        self.time = 0
        self.agents = []

    def add(self, agent):
        self.agents.append(agent)

    def remove(self, agent):
        while agent in self.agents:
            self.agents.remove(agent)

    def step(self):
    #Modified part. Adding a first (provision) and second(allocation)
        round
        for agent in self.agents:
            agent.first_round(self.model)
        self.model.resource =
            self.model.created_resource(self.model.sum_of_contributions)
        for agent in self.agents:
            agent.second_round(self.model)
        self.steps += 1
        self.time += 1

    #Added function for the linear goods game
    def symstep(self):
        for agent in self.agents:
            agent.first_round(self.model)
        self.model.resource = self.model.r *
            (self.model.sum_of_contributions)
        self.model.equal_share = (self.model.resource / 5)
        for agent in self.agents:
            agent.sym_second_round(self.model)
        self.steps += 1
        self.time += 1


    def get_agent_count(self):
        return len(self.agents)

class BatchRunner(object):


    model_cls = None
    parameter_values = {}
    iterations = 1
```

```
model_reporters = {}
agent_reporters = {}

model_vars = {}
agent_vars = {}

def __init__(self, model_cls, parameter_values, iterations=1,
             max_steps=1000, model_reporters=None,
                 agent_reporters=None):
    self.model_cls = model_cls
    self.parameter_values = {param: self.make_iterable(vals)
                             for param, vals in
                                 parameter_values.items()}
    self.iterations = iterations
    self.max_steps = max_steps

    self.model_reporters = model_reporters
    self.agent_reporters = agent_reporters

    if self.model_reporters:
        self.model_vars = {}

    if self.agent_reporters:
        self.agent_vars = {}

def run_all(self): # modified to give some progress info while the
    model is running
    params = self.parameter_values.keys()
    param_ranges = self.parameter_values.values()
    run_count = 0
    s = 1
    g = len(alpha_values) * len(beta_values)
    if figure3_6 == 1 or figure3_7 == 1:
        print("running...")
    elif figure3_8 ==1:
        pass
    else:
        print("running iteration " + str(s) + " of " +
            str(iterations) + "...")
    for param_values in list(product(*param_ranges)):
        kwargs = dict(zip(params, param_values))
        for _ in range(self.iterations):
            model = self.model_cls(**kwargs)
            self.run_model(model)
            # Collect and store results:
            if self.model_reporters:
                key = tuple(list(param_values) + [run_count])
                self.model_vars[key] = self.collect_model_vars(model)
            if self.agent_reporters:
                for agent_id, reports in
```

```
                    self.collect_agent_vars.items():
                    key = tuple(list(param_values) + [run_count,
                        agent_id])
                    self.agent_vars[key] = reports
            run_count += 1
            if figure3_6 != 1 and figure3_7 != 1 and figure3_8 != 1:
                if run_count % g == 0:
                    s += 1
                    if s <= iterations:
                        print("running iteration " + str(s) + " of " +
                            str(iterations) + str("..."))
                    else:
                        print("")
                        print("Done!")

def run_model(self, model):
    while model.running and model.schedule.steps < self.max_steps:
        asym = np.random.choice(2, p = [model.fa, (1 - model.fa)])
        model.step(asym)

def collect_model_vars(self, model):
    model_vars = {}
    for var, reporter in self.model_reporters.items():
        model_vars[var] = reporter(model)
    return model_vars

def collect_agent_vars(self, model):
    agent_vars = {}
    for agent in model.schedule.agents:
        agent_record = {}
        for var, reporter in self.agent_reporters.items():
            agent_record[var] = reporter(agent)
        agent_vars[agent.unique_id] = agent_record
    return agent_vars

def get_model_vars_dataframe(self):
    index_col_names = list(self.parameter_values.keys())
    index_col_names.append("Run")
    records = []
    for key, val in self.model_vars.items():
        record = dict(zip(index_col_names, key))
        for k, v in val.items():
            record[k] = v
        records.append(record)
    return pd.DataFrame(records)

def get_agent_vars_dataframe(self):
    index_col_names = list(self.parameter_values.keys())
    index_col_names += ["Run", "AgentID"]
    records = []
```

```
        for key, val in self.agent_vars.items():
            record = dict(zip(index_col_names, key))
            for k, v in val.items():
                record[k] = v
            records.append(record)
        return pd.DataFrame(records)

    @staticmethod
    def make_iterable(val):
        if hasattr(val, "__iter__") and type(val) is not str:
            return val
        else:
            return [val]


#Functions for determining amounts of generated CPR, as well as ginis
    for investments and extractions,
# and distribution of investments and extractions between agent
    positions
def determine_final_resource(model):
    final_resource = model.determine_generated_cpr()
    return final_resource

def determine_takeouts(model):
    takeouts = model.determine_agent_total_takeout()
    return takeouts

def determine_contributions(model):
    contributions = model.determine_agent_total_contribution()
    return contributions

def compute_gini_takeouts(model):
    denominator = 0
    divisor = 0
    agent_takeout_sum = [agent.total_amount_taken_out for agent in
        model.schedule.agents]
    for xi in agent_takeout_sum:
        for xj in agent_takeout_sum:
            divisor += xj
            a = xi - xj
            if a < 0:
                a = -1 * a
            denominator += a
    gini = denominator / ((divisor * 2) + 0.000000000000000001)
    return gini

def compute_gini_contributions(model):
    denominator = 0
    divisor = 0
    agent_contribution_sum = [agent.total_contribution for agent in
```

```
        model.schedule.agents]
    for xi in agent_contribution_sum:
        for xj in agent_contribution_sum:
            divisor += xj
            a = xi - xj
            if a < 0:
                a = -1 * a
            denominator += a
    gini = denominator / ((divisor * 2) + 0.000000000000000001)
    return gini

#Function for running the cultural selection model
def imitation_model(alpha, beta, eta, iterations, groups, m, epsilon,
    fa):
    group_list = []
    #creating groups of agents
    for j in range(groups):
        group_list.append(Resource_Model(agents, alpha, beta, eta, fa,
            r, j))

    #application of the white noise term
    its = 0
    rolling_output = []
    for g in range(iterations):
        its += 1
        rolling_avg_alpha = 0
        rolling_avg_beta = 0
        rolling_avg_cpr = 0
        for h in range(groups):
            group_list[h].run_model(runs)
            rolling_avg_cpr += group_list[h].this_round_produced_resource
            for j in group_list[h].schedule.agents:
                rolling_avg_alpha += (j.alpha / agents)
                rolling_avg_beta += (j.beta / agents)
        rolling_avg_cpr = (rolling_avg_cpr / (runs * groups))
        rolling_avg_beta = (rolling_avg_beta / groups)
        rolling_avg_alpha = (rolling_avg_alpha / groups)
        subroll = [its, rolling_avg_cpr, rolling_avg_alpha,
            rolling_avg_beta]
        rolling_output.append(subroll)

        for h in range(groups):
            for c in group_list[h].schedule.agents:
                noise = np.random.normal(0, 0.01)
                c.alpha += noise
                #ensuring alpha <= 1
                if c.alpha > 1:
                    c.alpha = 1
                if c.alpha <-1:
                    c.alpha = -1
```

```
        noise = np.random.normal(0, 0.01)
        c.beta += noise
        #ensuring beta <= alpha
        if c.beta > c.alpha:
            c.beta = c.alpha
        if c.beta < -1:
            c.beta = -1

#Copying traits of another agent from same(p=1-m) or another
    group(p=m)
#after each set of rounds, with specified probability depending
    on agents'
#past payoff
for h in range(groups):
    for x in group_list[h].schedule.agents:
        #Determining if agent from the same group or another is
            faced
        agent_imitation = np.random.choice(2, p = [(1-m), m])
        w_i = x.score
        jchoice = np.random.choice(agents)
        #Facing random agent from the same group
        if agent_imitation == 0:
            j = group_list[h].schedule.agents[jchoice]
            w_j = j.score
            #probability of imitation
            imit_p = (w_j / (w_i + w_j + 0.0000000000000001))
            imit = np.random.choice(2, p = [imit_p, (1 - imit_p)])
            if imit == 0:
                x.alpha = j.alpha
                x.beta = j.beta
        #facing an agent from another group
        else:
            gchoice = np.random.choice(groups)
            j = group_list[gchoice].schedule.agents[jchoice]
            w_j = j.score
            #probability of imitation
            imit_p = (w_j / (w_i + w_j + 0.00000000000000001))
            imit = np.random.choice(2, p = [imit_p, (1 - imit_p)])
            if imit == 0:
                x.alpha = j.alpha
                x.beta = j.beta

#Intergroup interaction
average_fitness = 0
for h in range(groups):
    average_fitness += group_list[h].total_produced_resource
average_fitness = (average_fitness/groups)

for h in range(groups):
    #determining if intergroup interaction takes place
```

```
            group_imitation = np.random.choice(2, p=[epsilon, (1 -
                epsilon)])
            #if intergroup interaction takes place:
            if group_imitation == 0:
                #Determining W_i
                fitness_i = (group_list[h].total_produced_resource /
                    (average_fitness + 0.000000000001))
                s_i = 0 #Determining s_i
                for a in group_list[h].schedule.agents:
                    if a.subscore >= 10:
                        s_i +=1
                s_i = (s_i / agents)
                gchoice = np.random.choice(groups) #Choosing another
                    group at random
                #Determining W_j
                fitness_j = (group_list[gchoice].total_produced_resource
                    / (average_fitness + 0.0000000000000001))
                s_j = 0 #Determining s_j
                for b in group_list[gchoice].schedule.agents:
                    if b.subscore >= 10:
                        s_j += 1
                s_j = (s_j / agents)
                imit_p = ((1 + ((fitness_j * s_j) - (fitness_i * s_i)))
                    / 2)
                if imit_p < 0:
                    imit_p = 0
                elif imit_p > 1:
                    imit_p = 1
                #Determining which group copies traits from the other
                    group
                imit = np.random.choice(2, p = [imit_p, (1 - imit_p)])
                #group i imitating group j
                if imit == 0:
                    z = 0
                    for x in group_list[h].schedule.agents:
                        j = group_list[gchoice].schedule.agents[z]
                        x.alpha = j.alpha
                        x.beta = j.beta
                        z += 1
                #group j imitating group i
                else:
                    z = 0
                    for x in group_list[gchoice].schedule.agents:
                        j = group_list[h].schedule.agents[z]
                        x.alpha = j.alpha
                        x.beta = j.beta
                        z += 1
    return rolling_output
```

```
#Model Parameter settings.
runs = 10 #corresponding to c
agents = 5
#alpha = 1 #Used for the run with variable eta
#beta = 1 #Used for the run with variable eta
#eta = 15
#fa = 0 #frequency of asymmetric games played.
r = 2 #parameter in the linear resource generating function.
iterations = 10 #iterations for each parameter combination
eta_values = [0, 0.125, 0.25, 0.375, 0.5, 0.75, 1, 1.5, 2, 3, 4, 6, 8,
    12, 15, 16, 32]
#Used for the run with variable eta
alpha_values = [] #the range of values for alpha with which the
    iterated model is run
for i in range(-10, 11):
    x = i/10
    alpha_values.append(x)
beta_values = [] #the range of values for beta with which the iterated
    model is run
for i in range(-10, 11):
    x = i/10
    beta_values.append(x)
#visualization settings
visu3d = 0

if figure3_2 == 1:
    visu3d = 1
    runs = 10
    agents = 5
    eta = 0.5
    fa = 1
    iterations = 50
    #parameters for the BatchRunner, i.e. the iterated model
    param_values = {"N": agents, "alpha": alpha_values, "beta":
        beta_values, "eta": eta,
               "fa": fa, "r": r}
    model_reporter = {"Generated_CPR": determine_final_resource,
                 "Gini_Contributions": compute_gini_contributions,
                 "Gini_Takeouts": compute_gini_takeouts}
    batch = BatchRunner(Resource_Model, param_values, iterations, runs,
        model_reporter)

if figure3_3 == 1:
    visu3d = 1
    runs = 10
    agents = 5
    eta = 0.5
    fa = 0
```

```
    iterations = 50
    #parameters for the BatchRunner, i.e. the iterated model
    param_values = {"N": agents, "alpha": alpha_values, "beta":
        beta_values, "eta": eta,
               "fa": fa, "r": r}
    model_reporter = {"Generated_CPR": determine_final_resource,
                 "Gini_Contributions": compute_gini_contributions,
                 "Gini_Takeouts": compute_gini_takeouts}
    batch = BatchRunner(Resource_Model, param_values, iterations, runs,
        model_reporter)


if figure3_5 == 1:
    visu3d = 1
    runs = 10
    agents = 5
    eta = 15
    fa = 1
    iterations = 50
    #parameters for the BatchRunner, i.e. the iterated model
    param_values = {"N": agents, "alpha": alpha_values, "beta":
        beta_values, "eta": eta,
               "fa": fa, "r": r}
    model_reporter = {"Generated_CPR": determine_final_resource,
                 "Gini_Contributions": compute_gini_contributions,
                 "Gini_Takeouts": compute_gini_takeouts}
    batch = BatchRunner(Resource_Model, param_values, iterations, runs,
        model_reporter)

if figure3_6 == 1:
    runs = 10
    agents = 5
    fa = 1
    iterations = 500
    #parameters for the BatchRunner, i.e. the iterated model
    param_values = {"N": agents, "alpha": -1, "beta": -1, "eta":
        eta_values,
               "fa": fa, "r": r}
    model_reporter = {"Generated_CPR": determine_final_resource,
                 "Gini_Contributions": compute_gini_contributions,
                 "Gini_Takeouts": compute_gini_takeouts}
    batch = BatchRunner(Resource_Model, param_values, iterations, runs,
        model_reporter)
    batch.run_all()
    out = batch.get_model_vars_dataframe() # Stores the collected data
    grouped = out.groupby("eta").mean()
    y = grouped.Generated_CPR
    ylist = np.array(y)
    ylist = ylist / 10
    fig = plt.figure()
```

```
ax1 = fig.add_subplot(111)
ax1.set_xlabel("eta")
ax1.set_ylabel("CPR level")
ax1.set_title("alpha = -1, beta = -1")
ax1.plot(eta_values, ylist)
ax1.axvline(15, color = "red")
ax1.axhline(0.075, color = "green")
ax1.set_xscale('log', basex = 2)
fig.show()


param_values2 = {"N": agents, "alpha": 1, "beta": -0.2, "eta":
    eta_values,
           "fa": fa, "r": r}
batch2 = BatchRunner(Resource_Model, param_values2, iterations,
    runs, model_reporter)
batch2.run_all()
out2 = batch2.get_model_vars_dataframe() # Stores the collected data
grouped2 = out2.groupby("eta").mean()
y2 = grouped2.Generated_CPR
ylist2 = np.array(y2)
ylist2 = ylist2 / 10
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
ax2.set_xlabel("eta")
ax2.set_ylabel("CPR level")
ax2.set_title("alpha = 1, beta = -0.2")
ax2.plot(eta_values, ylist2)
ax2.axvline(15, color = "red")
ax2.axhline(2, color = "green")
ax2.set_xscale('log', basex = 2)
fig2.show()

param_values3 = {"N": agents, "alpha": 1, "beta": 1, "eta":
    eta_values,
           "fa": fa, "r": r}
batch3 = BatchRunner(Resource_Model, param_values3, iterations,
    runs, model_reporter)
batch3.run_all()
out3 = batch3.get_model_vars_dataframe() # Stores the collected data
grouped3 = out3.groupby("eta").mean()
y3 = grouped3.Generated_CPR
ylist3 = np.array(y3)
ylist3 = ylist3 / 10
fig3 = plt.figure()
ax3 = fig3.add_subplot(111)
ax3.set_xlabel("eta")
ax3.set_ylabel("CPR level")
ax3.set_title("alpha = 1, beta = 1")
ax3.plot(eta_values, ylist3)
```

```
    ax3.axvline(15, color = "red")
    ax3.axhline(7, color = "green")
    ax3.set_xscale('log', basex = 2)
    fig3.show()

if figure3_7 == 1:
    runs = 10
    agents = 5
    eta = 15
    fa = 1
    iterations = 5000
    model_reporter = {"Takeouts": determine_takeouts, "Contributions":
        determine_contributions}
    scale = iterations * runs
    ind = np.arange(agents)
    width = 0.28

    param_values = {"N": agents, "alpha": 0.85, "beta": 0.4, "eta": eta,
              "fa": fa, "r": r}
    batch = BatchRunner(Resource_Model, param_values, iterations, runs,
        model_reporter)
    batch.run_all()
    out = batch.get_model_vars_dataframe()
    contrib = out.Contributions[0]
    for j in range(1, iterations):
        subcon = out.Contributions[j]
        contrib = [contrib[i]+subcon[i] for i in range(len(contrib))]
    contrib = [(contrib[i]/scale) for i in range(len(contrib))]
    extrac = out.Takeouts[0]
    for j in range(1, iterations):
        subex = out.Takeouts[j]
        extrac = [extrac[i]+subex[i] for i in range(len(extrac))]
    extrac = [(extrac[i]/scale) for i in range(len(extrac))]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind + width, contrib, width, color = "blue")
    rects2 = ax.bar(ind + width * 2, extrac, width, color = "red")
    ax.set_xlim(-width, len(ind) + width)
    ax.set_ylabel("Contributions/Extractions")
    ax.set_title("Alpha: " + str(0.85) + " Beta: " + str(0.4))
    xTickMarks = ["Agent" + str(i) for i in range(agents)]
    ax.set_xticks(ind + width)
    xtickNames = ax.set_xticklabels(xTickMarks)
    plt.setp(xtickNames, rotation = 45, fontsize = 10)
    ax.legend((rects1[0], rects2[0]), ("Contributions", "Extractions"))
    fig.show()

    param_values2 = {"N": agents, "alpha": 1, "beta": 0.75, "eta": eta,
              "fa": fa, "r": r}
    batch2 = BatchRunner(Resource_Model, param_values2, iterations,
```

```
    runs, model_reporter)
batch2.run_all()
out2 = batch2.get_model_vars_dataframe()
contrib2 = out2.Contributions[0]
for j in range(1, iterations):
    subcon2 = out2.Contributions[j]
    contrib2 = [contrib2[i]+subcon2[i] for i in range(len(contrib2))]
contrib2 = [(contrib2[i]/scale) for i in range(len(contrib2))]
extrac2 = out.Takeouts[0]
for j in range(1, iterations):
    subex2 = out2.Takeouts[j]
    extrac2 = [extrac2[i]+subex2[i] for i in range(len(extrac2))]
extrac2 = [(extrac2[i]/scale) for i in range(len(extrac))]
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
rects3 = ax2.bar(ind + width, contrib2, width, color = "blue")
rects4 = ax2.bar(ind + width * 2, extrac2, width, color = "red")
ax2.set_xlim(-width, len(ind) + width)
ax2.set_ylabel("Contributions/Extractions")
ax2.set_title("Alpha: " + str(1) + " Beta: " + str(0.75))
xTickMarks = ["Agent" + str(i) for i in range(agents)]
ax2.set_xticks(ind + width)
xtickNames = ax2.set_xticklabels(xTickMarks)
plt.setp(xtickNames, rotation = 45, fontsize = 10)
ax2.legend((rects3[0], rects4[0]), ("Contributions", "Extractions"))
fig2.show()

param_values3 = {"N": agents, "alpha": 1, "beta": 1, "eta": eta,
           "fa": fa, "r": r}
batch3 = BatchRunner(Resource_Model, param_values3, iterations,
    runs, model_reporter)
batch3.run_all()
out3 = batch3.get_model_vars_dataframe()
contrib3 = out3.Contributions[0]
for j in range(1, iterations):
    subcon3 = out3.Contributions[j]
    contrib3 = [contrib3[i]+subcon3[i] for i in range(len(contrib3))]
contrib3 = [(contrib3[i]/scale) for i in range(len(contrib3))]
extrac3 = out.Takeouts[0]
for j in range(1, iterations):
    subex3 = out3.Takeouts[j]
    extrac3 = [extrac3[i]+subex3[i] for i in range(len(extrac3))]
extrac3 = [(extrac3[i]/scale) for i in range(len(extrac))]
fig3 = plt.figure()
ax3 = fig3.add_subplot(111)
rects5 = ax3.bar(ind + width, contrib3, width, color = "blue")
rects6 = ax3.bar(ind + width * 2, extrac3, width, color = "red")
ax3.set_xlim(-width, len(ind) + width)
ax3.set_ylabel("Contributions/Extractions")
ax3.set_title("Alpha: " + str(1) + " Beta: " + str(1))
```

```
    xTickMarks = ["Agent" + str(i) for i in range(agents)]
    ax3.set_xticks(ind + width)
    xtickNames = ax3.set_xticklabels(xTickMarks)
    plt.setp(xtickNames, rotation = 45, fontsize = 10)
    ax3.legend((rects5[0], rects6[0]), ("Contributions", "Extractions"))
    fig3.show()

if figure3_8 == 1:
    runs = 10
    agents = 5
    eta = 15
    iterations = 3000
    groups = 100
    m = 0.001 #Probability of facing an agent from another group vs.
        from the
    #agent's own group
    epsilon = 0.015 #Probability of intergroup interaction
    subiterations = 1
    fa_range = 11 #Specifying the range of the frequency of asymmetric
        games,
    #fa
    out = []
    run_id = 1 #Gives the output file a unique id in order to avoid
        overwriting
    print("run_id: " + str(run_id))
    #print("running...")
    for j in range(0, fa_range):
        print("Running fa = " + str(j/10) + "...")
        results = [0, 0, 0, 0]
        for g in range(subiterations):#Collecting Averages of multiple
        #iterations for subiterations > 1
            alpha = 0 # starting value for alpha
            beta = 0 # starting value for beta
            #print("Running iteration " + str(g + 1) + " of " +
                str(subiterations) + "...")
            sublist = imitation_model(alpha, beta, eta, iterations,
                groups, m, epsilon, j/10)
            #Referring to the imitation_model function defined above
            results = [results[i]+sublist[(iterations - 1)][i] for i in
                range(len(results))]
            out2 = pd.DataFrame(sublist)

            plt.plot(out2[1])
            plt.xlabel("Round")
            plt.ylabel("Average CPR value")
            plt.title("Development of CPR, fa = " + str(j/10))
            plt.show()

            alpha = plt.plot(out2[2], label = "alpha")#, marker = "d")
            beta = plt.plot(out2[3], label = "beta")#, marker = "s")
```

```
            plt.legend()
            plt.xlabel("Rounds")
            plt.ylabel("Average values for alpha and beta")
            plt.title("Development of alpha and beta, fa = " + str(j/10))
            plt.ylim(-1, 1)
            plt.hlines(0, 0, iterations)
            plt.show()

        results = (np.array(results))
        results = (results/subiterations)
        out.append(results)

    #Visualization

    out = pd.DataFrame(out)


    #Visualization
    plt.plot(out[1])
    plt.xlabel("Number of times an asymmetric game is played")
    plt.ylabel("Average CPR value")
    plt.xticks(range(11))
    plt.show()

    alpha = plt.plot(out[2], label = "alpha", marker = "d")
    beta = plt.plot(out[3], label = "beta", marker = "s")
    plt.legend()
    plt.xticks(range(11))
    plt.xlabel("Number of times an asymmetric game is played")
    plt.ylabel("Average values for alpha and beta")
    plt.ylim(-1, 1)
    plt.hlines(0, 0, 10)
    plt.show()

    filename = str("SelectionModel_ID" + str(run_id) + ".csv")
    out.to_csv(str(filename))
    print("Output saved as " + str(filename))




#Printout of information on the running model
print("Model specs: ")
print("runs " + str(runs))
print("agents " + str(agents))
print ("iterations " + str(iterations))
if figure3_6 != 1:
    print("eta " + str(eta))
if figure3_8 != 1:
```

```
    print("fa " + str(fa))
print("")

if figure3_6 != 1 and figure3_7 != 1 and figure3_8 != 1:
    batch.run_all()
    out = batch.get_model_vars_dataframe() # Stores the collected data

#Saving the output as an analyzable .csv file
if figure3_6 != 1 and figure3_7 != 1 and figure3_8 != 1:
    filename = str(runs) + "runs_" + str(agents) + "agents_" +
        str(iterations) + "iterations_" + str(eta) + "eta_" + str(fa) +
        "fa.csv"
    out.to_csv(str(filename))
    print("Output saved as " + str(filename))
"""elif figure3_8 == 1:
    filename = str("results_" + str(eta) + "eta_" + str(iterations) +
        "iterations.csv")
    out.to_csv(str(filename))
    print("Output saved as " + str(filename))"""


#3D-Visualization
if visu3d == 1:
    out_one = out[out.alpha >= out.beta]
    out_two = out[out.beta > out.alpha]
    for j in out_two.index:
        out_two.set_value(j, "Generated_CPR", 0)
        out_two.set_value(j, "Gini_Contributions", 0)
        out_two.set_value(j, "Gini_Takeouts", 0)
    out = out_one.merge(out_two, how="outer")
    grouped_all = out.groupby(["alpha", "beta"]).mean()
    resource_values = grouped_all.Generated_CPR
    resource_list = np.array(resource_values)
    resource_list = ((resource_list) / runs)
    z_values = [resource_list[x:x+21] for x in range(0,
        len(resource_list), 21)]
    variables = []
    for j in range(-10, 11):
        variables.append(j/10)
        x = []
    for j in range(21):
        x.append(variables)
    y = []
    for j in variables:
        sublist = []
        for s in range(21):
            sublist.append(j)
        y.append(sublist)
    data = (x, y, z_values)
    X, Y, Z = data
```

```
contri_gini_values = grouped_all.Gini_Contributions
contri_gini_list = np.array(contri_gini_values)
a_values = [contri_gini_list[x:x+21] for x in range(0,
    len(contri_gini_list), 21)]
data2 = (x, y, a_values)
X, Y, A = data2
takeout_gini_values = grouped_all.Gini_Takeouts
takeout_gini_list = np.array(takeout_gini_values)
b_values = [takeout_gini_list[x:x+21] for x in range(0,
    len(takeout_gini_list), 21)]
data3 = (x, y, b_values)
X, Y, B = data3
def plot_3d(X, Y, Z, zaxis_label, azim, elev, invert_xaxis=True,
    invert_yaxis=True):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    X = np.array(X)
    Y = np.array(Y)
    Z = np.array(Z)
    if invert_xaxis==True:
        plt.gca().invert_xaxis()
    if invert_yaxis==True:
        plt.gca().invert_yaxis()
    ax.plot_wireframe(X, Y, Z)
    ax.set_xlabel('Beta')
    ax.set_ylabel('Alpha')
    ax.set_zlabel(str(zaxis_label))
    ax.set_title(str(zaxis_label))
    ax.view_init(azim, elev)
    plt.show()

plot_3d(X, Y, Z, "CPR level", 30, 60)
plot_3d(Y, X, A, "Gini Contributions", 30, 60, invert_xaxis=False,
    invert_yaxis=False)
plot_3d(X, Y, B, "Gini Takeouts", 30, 30, invert_xaxis=True,
    invert_yaxis=False)
```

Appendix B

Code the for the Possible Outcomes of the Probabilistic Choice Algorithm

```
# -*- coding: utf-8 -*-

import math
import matplotlib.pyplot as plt

eta_values = [0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16]
#Generated CPR for a given sum of investments:
def generated_resource(sum_of_contributions):
    if sum_of_contributions >= 0 and sum_of_contributions <= 1:
       produced_resource = 0
    elif sum_of_contributions >1 and sum_of_contributions <= 1.5:
       produced_resource = 0.5
    elif sum_of_contributions >1.5 and sum_of_contributions <= 2:
       produced_resource = 2
    elif sum_of_contributions >2 and sum_of_contributions <= 2.5:
       produced_resource = 4
    elif sum_of_contributions >2.5 and sum_of_contributions <= 3:
       produced_resource = 6
    elif sum_of_contributions >3 and sum_of_contributions <= 3.5:
       produced_resource = 7.5
    elif sum_of_contributions >3.5 and sum_of_contributions <= 4:
       produced_resource = 8.5
    elif sum_of_contributions >4 and sum_of_contributions <= 4.5:
       produced_resource = 9.5
    elif sum_of_contributions >4.5 and sum_of_contributions <= 5:
       produced_resource = 10
    return produced_resource

def investment_p(alpha, beta, eta, xj, yj):
    u_list = []
    for x in range(0, 11, 1):
```

```
        wi = 1 - x/10 + yj * generated_resource(x/10 + xj)
        wavg = 1 - 0.2 * x/10 - 0.2 * xj + 0.2 * generated_resource(x/10
            + xj)
        u_of_x = wi - alpha * max(wi - wavg, 0) + beta * max(wavg - wi,
            0)
        u_list.append(u_of_x)
    p_list = []
    denominator = 0
    for u in u_list:
        denominator += math.exp(eta * u)
    for u in u_list:
        numerator = math.exp(eta * u)
        p_list.append(numerator / denominator)
    return p_list
#Values of possible investments:
xs = []
for j in range(0, 11, 1):
    xs.append(j/10)
#stores expected investment values for given xj, yj, and eta values:
exp_list1 = []
exp_list2 = []
#looping through the defined values for eta:
for b in eta_values:
#looping through all possible values for xj: between 0 and 4 in 0.1
#increments:
    for z in range(0, 41, 1):
# looping through possible yj values from 0 to 1 in 0.01 increments:
        for g in range(0, 101, 1):
 # get probabilities for all 10 investment levels:
            prob_list1 = investment_p(-1, -1, b, z/10, g/100)
            prob_list2 = investment_p(1, 1, b, z/10, g/100)
 # calculating the expected value:
            exp_list1.append(sum([a*b for a,b in zip(prob_list1, xs)]))
            exp_list2.append(sum([a*b for a,b in zip(prob_list2, xs)]))
#Code Below: Visualization
yj_values = []
for s in eta_values:
    sublist = []
    for t in range(0, 41, 1):
        subsublist = []
        for q in range(0, 101, 1):
            subsublist.append(s)
        sublist.append(subsublist)
    yj_values.append(sublist)
plt.scatter(yj_values, exp_list1, color = "blue")
plt.title("Alpha: " + str(-1) + " Beta: " + str(-1))
plt.ylim(0, 1)
plt.yticks(xs)
plt.xscale("log", basex = 2)
plt.xticks(eta_values)
```

```
plt.xlabel("eta")
plt.ylabel("Range of possible exptected values for x")
plt.show()

plt.scatter(yj_values, exp_list2, color = "blue")
plt.title("Alpha: " + str(1) + " Beta: " + str(1))
plt.ylim(0, 1)
plt.yticks(xs)
plt.xscale("log", basex = 2)
plt.xticks(eta_values)
plt.xlabel("eta")
plt.ylabel("Range of possible exptected values for x")
plt.show()
```

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

München, den 22. September 2016

Joachim Reinhardt