Simulating Tax Policy: Agent Utility, Elections, and the Dynamics of Labor and Taxation with LLM Generative Agents

Samuel David Kleiner

Advisor: Professor Chi Jin

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

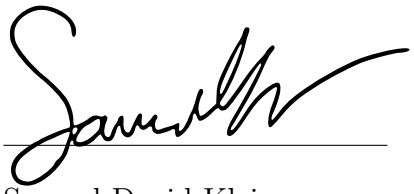
Department of Electrical and Computer Engineering

Princeton University

May 2025

I hereby declare that this Independent Work report represents my own work in accordance with University regulations.

I hereby declare that this Independent Work does not include regulated human subjects research.



Samuel David Kleiner

Simulating Tax Policy: Agent Utility, Elections, and the Dynamics of Labor and Taxation with LLM Generative Agents

Samuel David Kleiner

Abstract

Experimenting with tax policy in the real world can be prohibitively expensive and politically infeasible. Governments need innovative simulation and modeling techniques to evaluate policy impacts before deployment. Existing approaches in optimal income taxation theory create sup-optimal policies by relying on economic models that make simplifying assumptions about human behavior. This thesis argues that large language models [LLMs] learn tax policies that result in higher social welfare than the tax policies produced by existing economic models by providing a scalable, affordable method to model societal behavior and optimize for social welfare. We model policy decisions as an infinite dynamic game between a tax planner (leader) and workers (followers), optimizing for Stackelberg equilibria that maximize social welfare. We use LLMs to generate synthetic human data facilitating policy mechanism design, testing, and optimization. To increase realism, we implement simulation scenarios where the tax planner is elected by worker agents. We validate our LLM-based approach by comparing our results in a two worker agent, one tax planner simulation to a Stackelberg equilibria that we calculate through backwards induction. We investigate the effect of different simulation scenarios and skill distributions on social welfare. We find that our LLM-based approach achieves higher social welfare than the tax policy calculated according to economist Emmanuel Saez's optimal income taxation formulas. Future work could implement extensions to Saez's formulas that incorporate more elements of human economic activity with the goal of achieving higher social welfare with learned policies in these more complicated scenarios.

Acknowledgements

Thank you Seth Karten, who has guided me through the learning curves of conducting academic research this year.

Thank you to Professor Chi Jin for being my advisor, and organizing fascinating research presentations for the group, and demonstrating how to question assumptions most people do not see.

Thank you to Professor Jaime Fisac for being my second reader and taking on my project.

Thank you to my team and coaches for keeping it fun.

Thank you to my friends for the laughs along the way.

Thank you to Sarah for talking science with me.

Thank you to my family for being my foundation model.

Contents

List of Figures

Chapter 1

Introduction

Accurately evaluating tax policies requires large-scale, real-world experimentation that is politically and financially challenging. Traditional economic models simplify human behavior, limiting their real-world accuracy. To address these challenges, innovative and scalable methods are needed for simulating societal behavior and optimizing tax policy. This research leverages large language models (LLMs) to generate synthetic human data and optimize tax policy, enabling affordable policy design since these models are already trained and can model human behavior [16]. By modeling tax policy decisions as a dynamic Stackelberg game between the government and residents, we create a simulation for tax policy generation and testing, and lay the groundwork for policy generation and testing in other policy areas.

1.1 The Problem

This thesis investigates whether formulating tax policy optimization as a Stackelberg game using synthetic human data generated by LLM generative agents results in higher social welfare than the optimal tax policy proposed by traditional economic models.

1.2 Why LLMs

Simulating humans' preferences in response to tax policies with LLMs allows users of this method to maximize any social welfare function and simulate any agent utility that can be articulated by natural language. Our simulation's formulation as a Stackelberg game does allow for other methods to solve it than LLMs. Backwards induction is a classic method for solving Stackelberg games, but our grid-search implementation of backwards induction taken in Appendix F.1 is intractable for large numbers of agents. Our implementation has a time complexity of $O(11^{\texttt{num\_brackets}+\texttt{num\_agents}})$ since each tax bracket is discretized into 11 possible rates: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. To find Stackelberg equilibria, we could have used methods from Multi-Agent Reinforcement Learning without LLMs [26] [51]. However, LLMs have the ability to take natural language input, like demographic and personal information, and have that information affect the action that the LLM outputs [15] [34]. Also LLMs' emergent ability to do in-context learning [12] [32] [50] allows for the same learning capability provided by more traditional Multi-Agent Reinforcement Learning [MARL] techniques.

Another advantage to our method is that using LLMs without fine-tuning allows us to avoid spending time and money on training and fine-tuning a new model. Furthermore, prompt engineering allows our simulation to be extremely flexible in its design without requiring any retraining as new features are implemented and tested.

We understand that LLMs are not all-powerful. There are limitations. An artificial tax game setting may be out-of-distribution with respect to the LLM's training dataset. There is a growing body of research on the abilities and limitations of LLMs to model human preferences that shows LLMs can model human behavior, but LLMs do not exhibit all the same biases that humans do [16] [43] [44]. Therefore, we validate the LLMs ability to converge to Stackelberg equilibria in our simulation by showing that it converges to the same Stackelberg equilibria as backwards induction for small numbers of agents and tax brackets.

1.3 Related Work

While harnessing the advantages of LLMs for our work, we drew inspiration from several sources of prior research.

1.3.1 Simulating Human Believable Agents

[34] designed a simulation where multi-agent interactions followed human-believable behaviors. This work provided an example of a multi-agent human simulacra simulation.

1.3.2 The Integration of Artificial Intelligence into Economics

The prior work of [52] and [53] built a two-level deep reinforcement learning simulation to optimize tax policies in a simulated economy where agents can gather resources, trade, and build houses. Their work has been followed by other attempts at mechanism design in economics using artificial intelligence. Our work furthers the integration of artificial intelligence into the field of economic simulation and mechanism design.

1.3.3 Modeling Noisily Rational Human Behavior

Humans don't always behave with perfect rationality. Instead, they act in a "noisily rational" manner, making generally reasonable decisions but with some inconsistency. Research by [25] has demonstrated the effectiveness of models that balance data from actual human behavior and humans' theoretically optimal actions. Their work also shows how mechanism design can be used to achieve specific policy goals, like reducing the inefficiency of traffic congestion. We use LLMs to model a balance between human's noisily rational behavior and theoretically optimal economic output. We use mechanism design to maximize social welfare.

1.3.4 Inverse Game Theory

Our approach simulates the effects of tax policy from the bottom up by having each agent solve their own personal optimization problem. Inverse game theory demonstrates the advantages of this bottom up approach. [10] showed that by learning the parameters of individual agents' personal optimization problems, we can better model how these agents perceive and respond to the leader in Stackelberg games. In our simulation, each agent performs exploration and exploitation in their own personal optimization problem, creating a more accurate simulation of strategic interactions.

1.3.5 Using LLMs to Find the Optimal Income Tax

Our simulation is based on the Mirrleesian framework from the field of optimal income taxation theory. The field seeks to design tax policy to maximize social welfare according to a social welfare function. It is difficult to find the tax rates that raise maximum revenue while encouraging economic growth, which promotes higher social welfare through redistribution from the government. If tax rates are too high, people are discouraged from working, but if they are too low, not enough money is raised and then redistributed through a variety of government services. Furthermore, the elasticity of labor has a cyclical relationship with determining the optimal income tax rates. The elasticity affects the optimal rates which in turn affects the elasticity, creating a difficult, circular optimization problem.

Using the product of income equality and economic productivity as a measure of social welfare, the prior work of [52] found a 16% increase in social welfare while using deep reinforcement learning techniques to adjust tax policy compared to using baseline policies based on the economist Saez's optimal taxation framework proposed in 2001 [38]. We use LLMs instead of deep reinforcement learning techniques to learn optimal tax policies.

Saez's more recent work suggests that the socially optimal top tax rate - for people with the income level of U.S. CEOs - could be 83% [35]. However, there is significant disagreement among economists. A 2019 University of Chicago survey of a panel of economic experts found that 20 economist disagreed and 8 agreed with the statement that: "Raising the top

federal marginal tax on earned personal income to 70 percent [...] would raise substantially more revenue (federal and state combined) without lowering economic activity" [37]. This disagreement demonstrates the need to improve the accuracy of tax policy simulations. The work of Saez and others in optimal income taxation theory uses numerical and theoretical simulations that usually estimate population parameters, like labor elasticity, and then derive optimal tax rates. This approach is limited by the Lucas critique [27] explained in Section 3.3.3.

1.3.6 Why Simulate Agent Responses to Income Tax: Atkinson-Stiglitz Theorem

It is necessary to focus on a tractable subset of the general problem of simulating human response to tax policy. Our simulation's tax policy only consists of the income tax. Income tax is the primary way the U.S. government raises revenue [9]. Another reason to focus on improving the simulation of income taxes is an important result in optimal income taxation theory: the Atkinson-Stiglitz theorem [2] [11]. It assumes weak separability between consumption goods and labor in utility. In practical terms, this means that how hard someone works doesn't change their relative preferences for goods, like apples versus oranges. Their overall budget depends on their labor, but the ratio at which they would be willing to substitute one good for another does not. It also assumes homogeneity across agents in consumption sub-utility. This means that different people may have different overall utility functions due to different preferences for labor/leisure tradeoffs, but they are assumed to have the same relative preferences for different consumption goods once their income level is fixed. With these assumptions, Atkinson and Stiglitz found that commodity taxes are not useful, so all redistribution should be done through the income tax. This result motivates our focus on the income tax.

1.4 Novel Contributions

The work of this thesis went towards the LLM Economist project described in [22]. We extend the frontier of economic policy simulation by introducing several innovations that differentiate our work from previous research, particularly the AI Economist developed in [52] and [53]. While we build upon the AI Economist's foundation of using artificial intelligence for tax policy optimization, our approach introduces several novel innovations. Our most significant innovation is the use of LLMs for human-behavior modeling, which opens up a whole new realm of possibilities for increasing the realism of our simulation. We also introduce a democratic scenario that implements the election of the tax planner from among worker agents allowing for democratic mechanism design instead of having a fixed, centralized tax planner. This better reflects real-world democratic processes where policymakers are selected by constituents. Furthermore, we introduce a platforms feature where every election period, worker agents decide if they want to run in the election, and, if they do, they output their proposed tax policy changes. All worker agents then see these proposed changes and vote. They record the elected tax planner's proposed tax policy changes and actual tax policy action the elected tax planner takes. Worker agents' memory of policy performance, elected tax planner's platforms, and elected tax planner's actions, enables agents to form coalitions, engage in strategic voting, and make decisions based on their utility functions.

1.5 Challenges and Considerations

Ensuring accuracy in our simulation requires consideration of how personal values and demographics impact human behavior, which has been studied in [29]. We need to exercise caution regarding the biases and reliability of model outputs, especially since most studies on the impact of personal values and demographics on human behavior involve western, educated, industrialized, rich, and democratic [WEIRD] populations [1]. Furthermore, LLMs have been shown to exhibit bias [4].

1.6 Thesis Organization

The remainder of this thesis is structured as follows: Chapter 2 provides background on the Stackelberg Game and game theory, establishing the theoretical framework for modeling tax policy as a leader-follower game. Chapter 3 explores Optimal Income Taxation Theory, reviewing foundational contributions from Mirrlees and Saez while introducing key concepts like social welfare maximization and isoelastic utility functions. Chapter 4 examines Large Language Models and In-Context Learning, and explains how these technologies can generate synthetic human behavior. Chapter 5 details our methodology for designing and implementing the our tax policy simulation as a Stackelberg game. Chapter 6 presents our experimental results, analyzing the performance of our approach compared to traditional economic models and optimal income taxation formulas. Chapter 7 discusses how our results achieved our initial goals. Chapter 8 reviews possible extensions to our simulation as potential directions for future work. The appendices provide engineering standards, more detailed data from our ablations and experiments, mathematical derivations, extensions to Saez's Optimal Income Taxation Formulas, and the most important elements of our simulation code.

Chapter 2

Stackelberg Game Theory

Stackelberg games have been used to model other economic situations, like consumer response to pricing when two companies sell the same product [28] [21], and have grown popular with the increased focus on AI as a good model for "computing optimal strategies to commit to" [28]. There is a growing body of theory and experiments to prove the effectiveness of Stackelberg games as a model.

2.1 Infinite Leader-Follower Games

Leader-follower games, particularly those modeled as infinite games of fixed duration, can be used to model decision-making scenarios where one agent (leader) commits to a strategy, and other agents (followers) respond. The term, "infinite games of fixed duration", means that the game is modeled as if it will continue infinitely, so that there are no changes in strategy based on an approaching end to the game [5].

2.1.1 Threat or Reward Strategies

Strategic threats can be used by the leader to encourage a specific response from the follower agents. For example, if the leader commits to taxing all income at 100% if followers do not work more than ten hours. the followers will probably decide to work more than ten hours.

However, if they do not, the leader has to accept the consequences and follow through on their threat for their threat strategy to be effective. The leader can also design policies with rewards to incentivize specific responses. By using reward strategies, policies can incentivize tax compliance or specific labor patterns.

We see threat and reward strategies used in real-world resource allocation policies. An example of resource allocation where threat and reward strategies are used is energy allocation. Los Angeles energy market prices can change depending on the time of day to incentivize use during peak production and discourage use during peak consumption [23]. These strategies are also used in resource allocation for compute. Amazon Web Services offers reduced prices for batch processing in data centers to utilize compute during off peak hours [3]. Our simulation allows for experimentation with threat and reward strategies in resource allocation through government tax revenue redistribution.

2.2 Mechanism Design and Policy

Mechanism design is a field within economics and game theory that focuses on designing rules known as "mechanisms" to achieve specific objectives. The rules are design to align agents' objectives such that when agents act rationally, the outcome of the game is the desired outcome of the mechanism designer. We can reformulate the generation of policies, like tax policy as a mechanism design problem. The current "mechanism design approach" to tax policy in optimal income taxation theory is flawed. This approach determines optimal resource allocation using a social welfare function, and designs a tax policy to achieve that allocation. This two step process fails because it inadequately incorporates humans' behavior response, which results in different implementations producing different outcomes [36]. We do mechanism design of tax policies with LLMs as they generate human-believable behavior for each agent, simulating the preferences and decisions of both leaders and followers to better incorporate the behavior response of worker agents to tax policies.

2.3 Stackelberg Game

The Stackelberg game is the ideal formalization for conducting mechanism design by learning optimal income taxation in our two-timescale economic simulation. The tax planner agent is the leader while the worker agents (the followers) respond. Our work is supported by recent work in computational mechanism design where the leader's commitment to its policy guides the followers to Stackelberg equilibria [7], which can lead to higher follower welfare [8].

The Stackelberg game offers a strong theoretical foundation for analysis of our simulation. Its sequential decision-making structure [46], removes the circularity issues that occur in simultaneous-move (static) games where each agent's optimal action depends on the other agents' actions. The sequential structure supports equilibria convergence by allowing the tax planner to use the reinforcement learning [RL] equivalent of backward induction. This is necessary for complex environments where backwards induction is not tractable. The tax planner optimizes its policy by learning and then anticipating its followers' behavior responses to different tax policies. The Stackelberg game allows our simulation's worker agents with bounded rationality to do no-regret learning by adapting their behavior responses incrementally based on past experiences rather than calculating optimal responses immediately, which would require the assumption of fully rational agents. [7]. The tax planner also does no-regret learning. We distinguish between the RL version of backward induction and no-regret learning in our simulation by noting that the tax planner observes workers' actions. This allows it to anticipate their responses to changes in tax policy, giving backward induction a significant role in accelerating convergence by guiding global policy adjustments. The worker agents do not anticipate the effect of their actions on the global policy, so they are only doing no-regret learning to minimize their personal regret in response to the policy. All worker agents collectively doing no-regret learning is still a powerful guiding force towards convergence. In our simulation, all followers move simultaneously after observing the leader's move. This implementation of a Stackelberg game is still a dynamic game because there are distinct stages of play - the leader stage followed

by the followers stage - even though the followers' subgame is itself a static game. Overall, the sequential design of the Stackelberg game facilitates the tax planner's ability to learn and anticipate agents' behavior response through the RL version of backwards induction. It also allows the worker agents to do no-regret learning, enabling our simulation to converge at Stackelberg equilibria.

2.4 Stackelberg Equilibria

Stackelberg equilibria are the Stackelberg game equivalent of Nash equilibria. No agent can unilaterally improve its position when the leader's plays its optimal strategy in response to the anticipated reactions of rational followers. Generalized Stackelberg equilibria extend these concepts to multi-leader, multi-follower games where there are interdependent constraints between players' strategies, which is relevant for resource allocation problems, like tax policy. Stackelberg equilibria are only stable under specific conditions:

- Both the leader and followers have perfect knowledge of each other's payoff functions and strategies [21], which is generally unrealistic for real-world applications [20].
- The leader correctly predicts the followers' responses.
- The followers are assumed to act rationally, always choosing their optimal response [49].
- The leader must commit to its strategy and cannot change it after followers make their decisions [20].

While our agents do not have perfect knowledge of each other's payoff functions and strategies, we encourage exploration for specific periods within our simulation runs to allow the agents to learn each other's payoff functions and strategies. We validate in 6.1 that the leader can correctly predict the followers' response for a small simulation where the Stackelberg equilibria is tractable. As we have established, humans act noisily rationally and so do our followers in periods where they balance exploration and exploitation. However to

converge to stable Stackelberg equilibria at the end of our simulation runs, our agents are prompted to act rationally by practicing pure exploitation. To meet the final condition in our simulation, the leader commits to its tax policy for one full tax period.

Chapter 3

Optimal Income Taxation Theory

Optimal income taxation theory provides a set of mathematical guidelines to how tax systems should be designed to maximize social welfare. It balances equity with productivity. It seeks to promote equity through the government's redistribution of tax revenue and increase productivity by encouraging labor.

The development of optimal income taxation theory began in earnest with the work of James Mirrlees in [31], and has received significant contributions from Edward Saez. While the field has many contributors, we will focus on their innovations as the fundamental works. The theory incorporates humans' behavioral responses from empirical evidence. We summarize the development of the Mirrleesian framework to provide the mathematical and theoretical basis for Saez's optimal income taxation formulas. We provide an overview of the critical economic concepts for Saez's formulas, and use them to establish a policy that we compare our learned policies against.

3.1 Simple Model with No Behavioral Responses

3.1.1 Framework

We begin with the simplest model where income z is fixed for each agent. The model incorporates no behavioral responses.

Assumptions and constraints:

- Utility $u(c)$ is strictly increasing and concave, and c is after-tax income
- Income distribution has probability density $h(z)$
- Tax function $T(z)$ determines tax paid by agent with income z.
- Consumption $c = z - T(z)$ is a agent's post-tax income. We will use $\tilde{z} = c = z - T(z)$ in our simulation.
- E is the required government revenue. E consists of government administrative costs, regulatory, defense, and intelligence agency funding, and other programs that consume tax revenue without redistributing it to citizens.

3.1.2 Utilitarian Optimization

If the government maximizes utilitarian social welfare:

$$\max_{T(z)} \int_0^\infty u(c)h(z)dz = \int_0^\infty u(z - T(z))h(z)dz \tag{3.1}$$

$$\text{subject to the constraint} \int_0^\infty T(z)h(z)dz \geq E \tag{3.2}$$

3.1.3 Solution

A full derivation of the Langrangian, its first-order condition, and the derivation of optimal income tax for utilitarian social welfare can be found in Appendix D.

The Lagrangian is:

$$\mathcal{L} = [u(z - T(z)) + \lambda \cdot T(z)] \cdot h(z) \tag{3.3}$$

First-order condition:

$$0 = \frac{\partial \mathcal{L}}{\partial T(z)} = [-u'(z - T(z)) + \lambda] \cdot h(z) \Rightarrow u'(z - T(z)) = \lambda \tag{3.4}$$

The Lagrange multiplier is cost of the constraint, and keeps the objective function bound to that constraint. The result, $u'(z-T(z)) = \lambda$, can also be understood as the idea that the utilitarian social welfare objective is maximized when the marginal utility u' of consumption $(z - T(z))$ (the additional happiness gained by one additional unit of post-tax income) is constant across all income levels. Maximizing the utilitarian social welfare objective requires post-tax income $z - T(z)$ to be a constant regardless of an agent's pretax income, z. For perfect equalization of post-tax income, a 100% marginal tax rate is required. This extreme result occurs because we have assumed no behavioral responses to taxation. In the real world, agents consider a 100% marginal tax rate to be unfair, and it has an extremely negative effect on agents' incentive to work. When approximations of agents' behavioral response are implemented, the utilitarian social welfare objective is still maximized by the equalization of marginal utility across income levels, but a 100% marginal tax rate and equal post-tax income is not required.

3.2 The Mirrlees Model

3.2.1 Framework

Mirrlees's work in [31] introduced a more realistic model for deriving the optimal income tax by incorporating labor supply responses:

- agents maximize utility $u(c, l)$ subject to the constraint $c = wl - T(wl)$
- l is labor supply, w is wage rate (which we call skill in our simulation), and $T(\cdot)$ is a nonlinear income tax function
- Skill w is private information (known only by each agent) with probability density $f(w)$

3.2.2 Social Welfare Maximization

In [31], Mirrlees introduces the general social welfare objective that maps individual agent's utility to social welfare, so the government maximizes:

$$\text{SWF} = \int G(u(c,l))f(w)dw \tag{3.5}$$

Subject to:

$$\int T(wl)f(w)dw \geq E \tag{3.6}$$

When $G(\cdot)$ is linear, $G(u) = u$ [38]. We see this in the utilitarian case in Appendix 3.1.2. Yet, the chosen $G(\cdot)$ is usually strictly concave, so that agents with lower utilities have a greater effect on the social welfare score. Thus to maximize social welfare the government's tax policy should focus on helping lower utility agents for whom the marginal utility is higher.

3.2.3 Key Results from Mirrlees

We review some key results from Mirrlees that guide the design of optimal tax policies and are reflected in our simulation design.

Marginal Tax Rate Constraint

An optimal marginal tax rate, $T'(\cdot)$, follows the constraint: $0 \leq T'(\cdot) \leq 1$. This result rules out negative marginal tax rates where $T'(\cdot) < 0$. A negative marginal tax rate can occur in the real world through a benefit, like the Earned Income Tax Credit[19] where the government transfers a net positive amount of cash to an agent. This result also rules out marginal tax rates $T'(\cdot) > 1$, which would equate to the government taking all earned income in that tax bracket and then requiring additional payment from that agent based on how much they earned in the bracket with $T'(\cdot) > 1$. Thus, $T'(\cdot) > 1$ would completely disincentivize an agent from working to produce the incomes taxed at $T'(\cdot) > 1$.

Marginal Tax Rate for Highest Earners

Marginal tax rate $T'(\cdot)$ should be zero for incomes at the top of the income distribution, $h(z)$, if skill distribution, $f(w)$ is bounded [31] [18]. On this analytical result, Mirrlees writes, "I would also hesitate to apply the conclusions regarding individuals of high skill: for many of them, their work is, up to a point, quite attractive, and the supply of their labour may be rather inelastic (apart from the possibilities of migration)" [31]. We present an intuitive explanation for this theoretical result. If the highest earning agent made \$1 million per year under a marginal tax rate, $0 \leq T'(\cdot) \leq 1$, and the government reduced the marginal tax rate to zero for all income over \$1 million, highest earning agent would be incentivized to work more. This would increase production, yet the government would not lose any revenue because no one was earning more than \$1 million before its policy change. This policy for top earners is clearly impractical, and, as Mirrlees points out, does not take into account the labor inelasticity of highly-skilled individuals who may be unlikely to change their behavior significantly based solely on their marginal tax rate. Moreover, Saez notes that the zero marginal tax rate result only definitively applies to the single highest earner [38], making it even more impractical to implement in a tax policy.

Marginal Tax Rate for Lowest Earners

Assuming every agent outputs nonzero labor and the lowest $z = wl > 0$, then $T'(\cdot) = 0$ at the bottom. Although this result was not explicitly calculated by Mirrlees in [31], it was later proved to be a result of the Mirrleesian framework by [40].

3.3 Saez's Framework

Saez was one of the first economists to apply the economic model developed by Mirrlees to real tax policies using empirical data by linking earnings elasticities to optimal income tax formula [41] [38]. In [38], Saez extended the work of Mirrlees by deriving the optimal general non-linear income tax, and matched the skill distribution he used in numerical simulations to U.S. Income distributions. By using the same social welfare calculation and

the same initial skill distribution, we compare the optimal tax policy found by our LLM-based approach to an optimal tax policy calculated from Saez's general non-linear income tax formula, taking inspiration from [52] and [53].

3.3.1 Saez's Optimal Income Taxation Formulas

Building on the Mirrlees framework, Saez proved in [38] that the optimal tax rate for income z is:

$$T'(z) = \frac{1 - G(z)}{1 - G(z) + a(z)e} \tag{3.7}$$

where $T'(z)$ is the optimal marginal tax rate at income z; G(z) is the social welfare weight, representing how much society values redistributing income to individuals earning z or less; a(z) is the Pareto parameter, describing the shape of the income distribution above z; and e is the elasticity of taxable income with respect to the net-of-tax rate, $(1 - T'(z))$. The net-of-tax rate is the fraction of an extra dollar an agent gets to keep.

3.3.2 Calculating a Saez Optimal Tax Policy

To calculate an optimal tax policy according to Saez's formulas, we use the elasticities for low, middle, and high earners calculated by Saez and Gruber in [17]. Since each run of the simulation depends on the agents' skill distribution, we calculate $G(z)$ and $a(z)$ for each tax bracket based on the agents' skill levels. Our implementation can be found in Appendix F.2. We use a piecewise linear tax policy in our simulation since the U.S. federal income tax policy is piecewise linear. A piecewise linear tax policy is a policy with set marginal income tax rates for defined income brackets.

Calculating G(z): The Social Welfare Weight

$G(z)$ quantifies how much society values additional income for individuals earning above threshold z. We assign welfare weights to each agent inversely proportional to their income, so $g(z_i) = 1/z_i$. We normalize these weights so they sum to one. Then, for a given z, we compute $G(z)$ as the sum of all welfare weights for incomes above z, divided by $(1 - F(z))$,

where $F(z)$ is the fraction of incomes below z.

$$G(z) = \frac{\sum_{i:z_i \geq z} g(z_i)}{1 - F(z)} \tag{3.8}$$

Calculating a(z): The Pareto Parameter

The parameter $a(z)$ is the local Pareto parameter of the income distribution at income level z. It captures how quickly the density of incomes decreases above income z. For most income levels, we use the formula:

$$a(z) = \frac{z \cdot f(z)}{1 - F(z)} \tag{3.9}$$

where $f(z)$ is the density of the income distribution at z, estimated using kernel density estimation. For the top tax bracket, we define $a(z)$ using the mean income m of those in the bracket:

$$a(z) = \frac{m}{m - z} \tag{3.10}$$

where z is the bracket's lower bound. This calculation for the top bracket better handles the heavy-tailed nature of income distributions.

Implementation for Discrete Brackets

Once $G(z)$ and $a(z)$ are calculated, along with the elasticity e from [17], we compute the optimal marginal tax rate using Saez's formula:

$$\tau(z) = \frac{1 - G(z)}{1 - G(z) + a(z)e} \tag{3.11}$$

For each bracket, we calculate a single tax rate using a representative income level z. For non-top brackets, z is chosen as the midpoint of the bracket.

$$z = 0.5 \cdot (b_{\text{start}} + b_{\text{end}}) \tag{3.12}$$

For the top bracket, z is chosen closer to the start to better represent the majority of taxpayers in that bracket

$$z = b_{\text{start}} + 0.1 \cdot (b_{\text{end}} - b_{\text{start}}) \tag{3.13}$$

This approach transforms Saez's continuous formula into a practical, implementable tax schedule with discrete brackets. The choice of where to evaluate z within each bracket is a design decision that can affect the resulting rates, but using the midpoint (or slightly above the lower bound for the top bracket) is a reasonable approach.

3.3.3 Susceptibility to Lucas Critique

Economists calculate the empirical labor elasticity from historical data, it is a fixed input in Saez's optimal income taxation formulas. Yet, as Lucas points out in [27], a new tax policy will result in changes to the behavior of agents, which results in a new labor elasticity, which results in a new optimal tax rate. This cycle continues. Saez's formulas cannot avoid the Lucas critique because of his formulas top down nature. They requires calculating population wide averages to produce optimal tax rates, and do not model the behavior of individual agents. However, the behavior of agents differs among different classes of agents [22]. In our approach, we model each agent's reaction to tax rates instead of estimating and fixing an elasticity for a large population group. By learning the optimal Stackelberg equilibria from the individual agents, we avoid making inaccurate population-wide assumptions about human behavior.

3.4 Isoelastic Utility

Widely used to model risk-averse preferences, isoelastic utility is also known as constant relative risk aversion. We will use this form:

$$u_i = \hat{z}_i - c \cdot l_i^{\delta} \tag{3.14}$$

where c is the labor disutility coefficient, l_i is the labor of agent i, $\hat{z}$ is the pre-tax income, and δ is the labor disutility exponent. An entity with a $\delta > 1$ means that the agent is risk - in our simulation risk is labor - averse. As labor increases, the additional utility for every additional unit of labor decreases. Intuitively, someone working and getting paid for two hours of work instead of one is doubling their salary at a level of labor that is still quite low; whereas, someone working 81 instead of 80 hours is seeing a small increase in their income when they are already working extremely hard.

3.5 Calculation of Social Welfare Metric

Using the same metric for social welfare calculation as [52] and [53], we sum all agent's utilities and divide by their pre-tax incomes. Here is our implementation from our planner.py file:

```
1 swf = sum([u[i]/max(z[i],1) for i in range(len(u))])
```

Code/swf.py

By comparing the swf results from Saez's optimal taxation policy and our learned policy, we determine if our method learns a better policy than the policy calculated from Saez's formulas.

Chapter 4

Large Language Models and In-Context Learning

4.1 Large Language Model

Large Language Models are machine learning models trained on extensive amounts of text data, often including much of the internet. These models, such as OpenAI's GPT, are capable of generating human-like text, answering complex queries, and performing tasks across a wide range of domains. LLMs are a subset of foundation models, a broader category of pre-trained models designed to perform a variety of tasks without or with minimal fine-tuning.

4.1.1 Attention, Transformers, and Decoder-Only Models

The underlying architecture of most major LLMs at the moment is the transformer, introduced by [45]. Transformers revolutionized natural language processing by employing an attention mechanism to process input sequences in parallel rather than sequentially, making them extremely efficient. The attention mechanism is a weighted sum of all the tokens in an input. The weights depend on the importance of each token to others in the input token sequence, so greater "attention" is paid to tokens that are the most relevant to other

tokens. The attention mechanism allows the model to "draw global dependencies between input and output" [45]. With enough training data, this results in nuanced understanding of inputs and useful response generation. The transformer architecture consists of two main components: the encoder and decoder. The encoder processes the input sequence and generates contextual embeddings. Contextual embeddings are representations of words that taken into account their context instead of static embeddings, like Word2Vec [30], which assign a single vector to a word regardless of its context. The decoder generates output sequences based on the processed embeddings.

Decoder-only models, like Generative Pretrained Transformer [GPT] models, focus exclusively on generation tasks by predicting the next token in a sequence, so it is best suited for tasks like text completion or generation. This also means that when engineering prompts, the order of the words in the prompt is critical. Prompt engineering is the process of crafting inputs to guide the model's outputs.

This thesis leverages LLMs to model human behavior due to their unique capabilities of generating human-believable text from natural language inputs and their extensive pretrained knowledge. Since LLMs can be trained on most of the internet, they are exceptionally good at producing one-shot human believable text. They simulate believable human behavior, even in complex scenarios like tax policy decision-making. Furthermore, unlike other optimization methods, LLMs can interpret and respond to inputs expressed in natural language, such as demographic details or policy descriptions. Using LLMs in our simulation removes the need to invest time and money to train unique models for unique situations. Prompt engineering enables efficient customization of our simulation.

4.1.2 In-Context Learning

Our simulation relies on LLMs' ability to improve their performance on a task when provided with examples in the input. This is an emergent ability known as In-Context Learning [ICL]. As an example, when tasked with sentiment analysis of tweets, the accuracy of LLMs increased if examples of correct input-output pairs were included in the prompt. Notably, the accuracy of LLMs decreased below the baseline accuracy - the accuracy when

no examples were given - when examples of sentiment analysis with the wrong answer were given [47]. This phenomenon shows LLMs' ability to learn from context, allowing us to do a version of more traditional reinforcement learning methods in our simulation.

Chapter 5

Methodology

Within our tax policy Stackelberg game, we implement a classic RL loop in which agents are initialized, they act in the simulation depending on their state, and receive a reward from their environment. The tax planner and worker agents balance exploration and exploitation of tax policy and labor choices, allowing the tax planner to optimize its policy based on the learned responses of the worker agents.

5.1 Algorithm Pseudocode

Here is pseudocode that describes the implementation of our simulation.

Algorithm 1 Tax System Simulation

Initialize agent skills: $skills = \{s_1, \ldots, s_N\}$
if uniform distribution **then**
    skills $\leftarrow \{s_i$.uniform() for i in 1 to $N\}$
else if us_income distribution **then**
    skills $\leftarrow \{s_i$.GB2() for i in 1 to $N\}$
end if
Create agents:
Initialize workers $\{\mathcal{W}_i\}_{i=1}^{N}$ with skills ${s_i}_{i=1}^{N}$ and isoelastic utility
Initialize tax planner $\mathcal{P}$ with swf utility and tax rates if the planner is fixed
Main simulation loop:
for each timestep t **do**
    Get current worker statistics (income and utility)
    if t mod two_timescale $= 0$ **then**
        if democratic scenario **then**
            Execute voting process:
            Agents declare candidacy and platforms (if enabled)
            Agents vote for preferred candidates
            Count votes and determine leader
            Inform agents of new leader
            Set tax policy:
            Elected leader chooses the change in tax policy
        else if planner is LLM **then**
            Tax planner sets new tax rates based on worker statistics
        end if
    end if
    Agents perform actions:
    for each agent **do**
        Agent decides labor hours based on tax rates and their utility function
    end for
    Apply taxes and distribute benefits:
    Calculate pre-tax incomes from labor choices
    Apply tax rates to calculate post-tax incomes and total tax revenue
    Update utilities:
    for each agent **do**
        Update agent utility based on income, tax paid, and rebate
    end for
end for

5.2 Agent Objectives: Utility Functions

5.2.1 Worker Objective: Isoelastic Utility

All rational, also known as egotistical, worker agents are initialized with a skill level, v_i, and have an isoelastic utility function. We use this form:

$$u_i = \hat{z}_i - c \cdot l_i^{\delta},$$

where $\hat{z}_i = z_i - T(z_i) + \frac{1}{N}\sum_{j=1}^{N} T(z_j)$ represents post-tax income, c is the labor disutility coefficient, and δ is the labor disutility exponent.

Workers' Objective: Each worker $\mathcal{W}_i \in \mathcal{W}$ chooses actions $l_t^i \in \mathcal{A}$ at each timestep t when exploitation is occurring to maximize their expected individual utility, with the objective $\max \mathcal{J}_{\mathcal{W}_i}$:

$$\max_{l_0^i,\ldots,l_{T-1}^i} \mathbb{E}\left[\sum_{t=0}^{T-1} u_i(o_t, l_t^i, \tau_{\lfloor t/K \rfloor})\right] \tag{5.1}$$

Workers cannot directly observe each other's actions, but receive feedback through the tax rebate, which is used to update their utility function. This utility serves as the reward for each timestep.

5.2.2 Tax Planner Objective: Social Welfare Function Utility

The tax planner agent has a social welfare utility function, defined by this formula:

$$SWF = \sum_i \frac{u_i}{z_i}, \tag{5.2}$$

The tax planner's objective is to maximize the sum of all agents' utility divided by pre-tax income to promote equity. The tax planner is provided with relevant history, and a prompt to output new tax rates for three tax brackets. If the number of brackets is larger than the number of agents, the state space is too large for true exploration to occur. We do our final experiment with 100 agents and income brackets: $[[0, 90000), [90000, 159100], [159100, \infty)]]$.

Tax Planner's Objective: At the beginning of each tax year k, the planner chooses

a tax policy $\tau_k \in \mathcal{T}$ to maximize their expected social welfare objective $\max \mathcal{J}_{\mathcal{P}}$:

$$\max_{\tau_0,\tau_1,\ldots,\tau_{\lfloor T/K \rfloor - 1}} \mathbb{E}\left[\sum_{t=0}^{T-1} \texttt{swf}(o_t, \boldsymbol{l}_t, \tau_{\lfloor t/K \rfloor})\right] \tag{5.3}$$

where, o_t is the observation, and $\boldsymbol{l}_t = (l_t^1, \ldots, l_t^N)$ are the actions of all workers at timestep t. The social welfare acts as the reward for each timestep.

5.3 Scenarios

5.3.1 Rational Scenario

If the simulation is set to the rational scenario, there is one tax planner agent, and n worker agents. The simulation is structured as a two timescale optimization problem where the tax planner agent sets tax rates on a slower timescale (the leader's stage in the Stackelberg game) with the goal of optimizing social welfare, and workers output their labor on a faster timescale (the followers stage - a static subgame), trying to optimize their isoelastic utility functions at every timestep.

5.3.2 Democratic Scenario

If the simulation is set to the democratic scenario, the tax planner is elected from among all worker agents every two timescale. The elected tax planner then outputs the tax policy as well as outputting labor. Agents vote for the tax planner based on their personal history of their labor, utility, and which agent was the leader from past timesteps.

Democratic Scenario with Platforms

To enhance the realism of the simulation, we implemented a platforms feature. When the platforms feature is enabled, all worker agents who want to run in an election output their proposed tax policy changes. Every agent will then receive the list of candidates and their proposed tax policy changes when they are voting. This enables agents to vote based on the proposed changes as well as their personal history of their labor, their utility, which

agent was the tax planner, and what the tax planner's proposed and actual policy was from past timesteps. The elected tax planner is under no constraint that requires them to follow through on their proposed tax policy.

5.3.3 Mathematical Formulation for all Scenarios

$$\text{Maximize} \quad SWF = \sum_i \frac{u_i}{z_i} \quad \text{(Tax Planner)} \tag{5.4}$$

$$\text{Maximize} \quad u_i = z_i - c \cdot l_i^{\delta} \quad \text{(Worker } i\text{)} \tag{5.5}$$

$$\text{Subject to:} \quad T_k \in [0\%, 100\%], \quad \Delta T_k \in \{-20\%, -10\%, 0\%, 10\%, 20\%\}. \tag{5.6}$$

5.3.4 Rational Scenario: Game Framework

We model the simulation of tax policy and labor decisions as a dynamic game involving two primary classes of agents: *workers* and *tax planners*. These agents operate in a structured environment characterized by the following components:

State Space (S)

The state space S represents all possible configurations of the game at any time step t:

- $s_t = (T_t, H_t)$, where T_t represents the tax policy at time t, defined as marginal tax rates for discrete income brackets, and H_t represents the historical context, including previous actions and outcomes. The outcomes are the aggregated historical pre-tax income $\hat{z}$ and utility u of all agents, and the total social welfare score.

Action Space (A)

Worker Agents:

- $a_{i,t}^w = l_{i,t}$, where $l_{i,t}$ is the labor choice of worker i at time t. Workers choose l_i to maximize u_i while balancing exploitation and exploration. When the time step is in the last 10% of the simulation or a slow timescale period, the historical message

is updated to instruct the LLM to output the best labor choice to maximize their utility, switching to pure exploitation.

Tax Planner Agent:

- $a_t^p = (\Delta T_{1,t}, \Delta T_{2,t}, \Delta T_{3,t})$, where $\Delta T_{k,t}$ represents the change in tax rates for the k-th bracket at time t. Each $\Delta T_{k,t}$ is constrained to $\{-20\%, -10\%, 0\%, 10\%, 20\%\}$.

Observation Space (O)

- Each agent observes a partial view of the state s_t:

  Workers: $o_{i,t}^w = (T_t, v_i, h_{i,t})$, where v_i is the worker's skill level and $h_{i,t}$ is the personal history of labor and utility.

  Tax Planner: $o_t^p = H_t$

Game Dynamics and Information Structure

Agents act sequentially within a time step.

- If it is time for an election:

  Step 1: Tax planner observes aggregated labor and income data to set or update tax policy if it is time for a new policy.

  Step 2: In a simultaneous-move subgame, workers output their labor choice based on the current tax policy and their personal utility.

5.3.5 Rational Scenario Diagrams

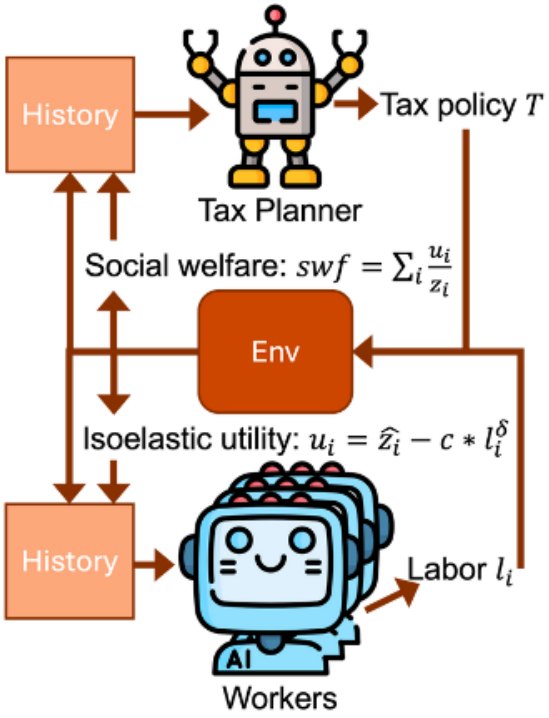


Figure 5.1: Diagram of Rational Scenario Information Flow

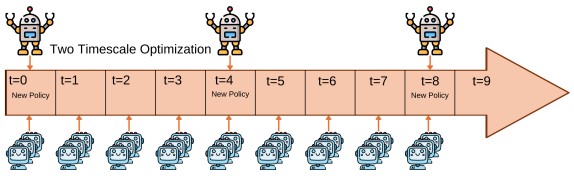


Figure 5.2: Diagram of Rational Scenario Timescale

5.3.6 Democratic Scenario: Game Framework

We model the simulation of tax policy and labor decisions as a dynamic game involving one class of agents: *workers*. One of the worker agents also serves as a *tax planner* during each tax period. These agents operate in a structured environment characterized by the following components:

State Space (S)

The state space S represents all possible configurations of the game at any time step t:

- $s_t = (T_t, H_t, P_t, C_{t/K})$, where P_t represents the current elected tax planner, and C_t represents the set of worker agents and their platforms running in an election during an election timestep. C_t is only included in the state space if the platforms feature is enabled.

Action Space (A)

Worker Agents:

- $a^w_{i,t} = (l_{i,t}, e_{i,t}, c_i)$, where $e_{i,t}$ is the vote of worker i for tax planner, and c_i is the worker agent's proposed tax policy changes if the platforms feature is enabled, and it is an election timestep.

Tax Planner Agent:

- $a^p_t = (\Delta T_{1,t}, \Delta T_{2,t}, \Delta T_{3,t}, l_{i,t}, e_{i,t}, c_i)$.

Observation Space (O)

- Each agent observes a partial view of the state s_t, **Workers:** $o^w_{i,t} = (T_t, v_i, h_{i,t})$, where $h_{i,t}$ is the personal history of labor, utility, and which agent was tax planner. If the platforms feature is enabled, the personal history includes what the tax planner's proposed tax policy and actual policy was.

  Elected Tax Planner: $o^p_t = (T_t, v_i, h_{i,t}, H_t)$, where the elected tax planner maintains the history of a worker agent, while adding, H_t during the period when the tax planner is in office.

Game Dynamics and Information Structure

Agents act sequentially within a time step.

- If it is time for an election:

  Step 1: If the platforms feature is enabled, all worker agents decide if they are running in the election, and output their proposed tax policy changes if they are.

  Step 2: All worker agents and the current tax planner, vote on the new tax planner, receiving candidates platforms if the platforms feature is enabled.

  Step 3: The elected tax planner observes aggregated labor and income data, and outputs their tax policy changes.

- For every timestep, t:

  Step 4: In a simultaneous-move subgame, all worker agents and the current tax planner decide labor efforts based on the current tax policy, personal utility, and personal histories.

5.3.7 Democratic Scenario Diagram

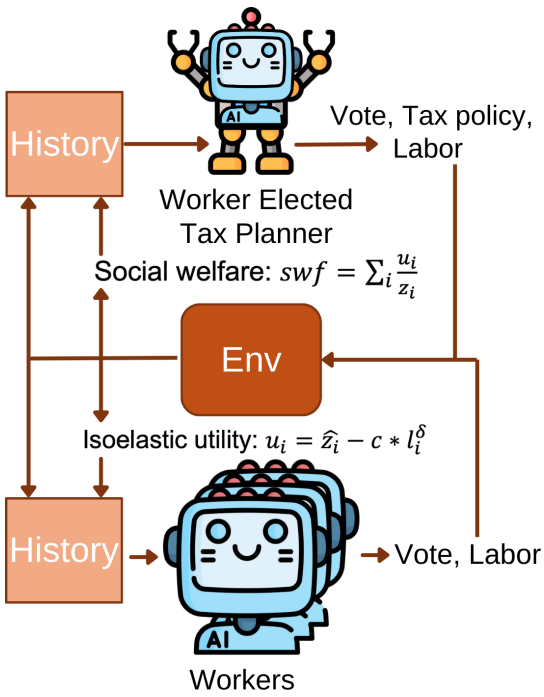


Figure 5.3: Diagram of Democratic Scenario Information Flow

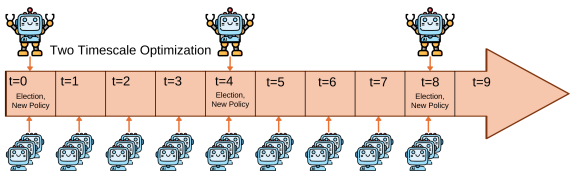


Figure 5.4: Diagram of Democratic Scenario Timescale

Chapter 6

Results

With the game dynamics established, we performed ablations to ensure that LLMs could accurately optimize the actions of the tax planner and worker agents to maximize their utility. We also tested various simulation parameters to determine the parameters that allowed the LLM to find the optimal solution. The simulation parameters tested include the number of timesteps, history length, and tax year length.

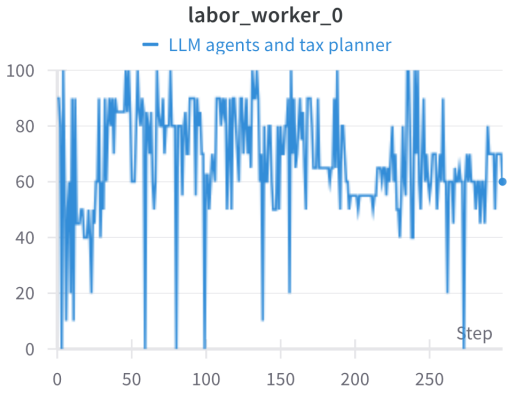
Ablation means to surgically remove. Our ablations remove complexity from the simulation by fixing the outputs of some agents to create a simpler optimization problem for the LLM. We use agents with fixed outputs and LLM agents in different combinations to demonstrate that the LLM outputs the correct actions to maximize social welfare according to a calculation of the Stackelberg equilibria using backwards induction in Appendix F.1. We used a local instance of LLaMa - Meta's LLM - to run these tests since it is open-source.

6.1 Ablations

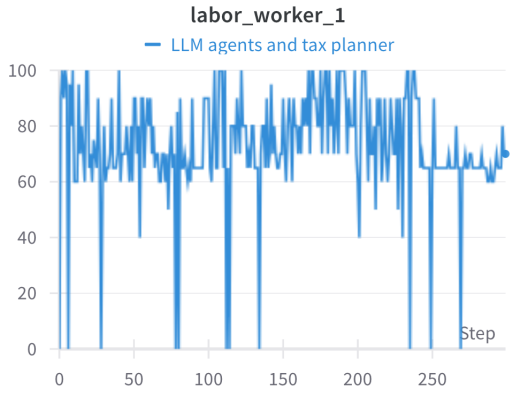
We used a simple simulation with one tax planner agent and two worker agents to validate the ability of LLMs to solve our two timescale optimization problem. The Stackelberg equilibria with two worker agents and two tax brackets, has the optimal tax rates [100, 0], and optimal labor of 60 for both workers. We tested every combination of Fixed and LLM agents. The results from the final test is shown below, and the rest of the results can be

found in Appendix B.4.

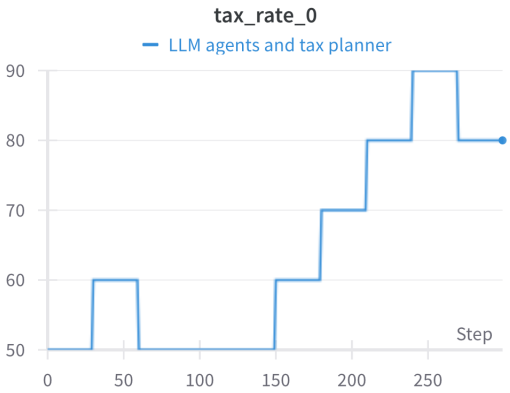
6.1.1 LLM Workers, LLM Tax Planner



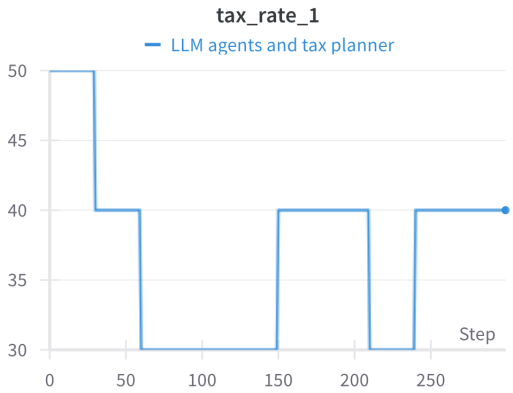
(a) Labor Worker_0, LLM



(b) Labor Worker_1, LLM



(c) Tax Rate for First Bracket, LLM



(d) Tax Rate for Second Bracket, LLM

Figure 6.1: Ablation Study Results: LLM Workers, LLM Tax Planner

We can see that the LLM performs exploration as well as exploitation, which it is instructed to do in the worker agent's system prompt:

```
1 self.system_prompt = 'You are ' + self.name + ', a citizen of
    Princetonia. Your skill level is ' + str(self.v) + ' out of
     159.1.'\
2         ' Each year you will have the option to choose the
            number of hours of labor to perform. \
```

```
3         You will receive income z proportional to the number
              of hours worked and your skill level. \
4         Your goal is to maximize your ' + utility_name + ' u.
              \
5         Make sure to sufficiently explore different amounts of
               LABOR before exploiting the best one for maximum
              utility u. \
6         Once you find the maximum utility, only output LABOR
              corresponding to maximum utility u. \
7         Use the JSON format: {\"LABOR\": \"X\"} and replace \"
              X\" with your answer.\n'
```

Code/worker_sys_prompt.py

The total prompt sent to the LLM is the user prompt followed by the system prompt. The user prompt is the agent's personal history, $h_{i,t}$.

We focus on the last 10% of the timesteps in the simulation since the LLM is instructed in the user prompt to switch to purely exploitation during the last 10% of the timesteps. We can see that for an LLM tax planner, and two LLM workers, LLaMa comes very close to the correct solution. While LLaMa touches on the correct value of 60, it is often at 65. These results still gave us confidence in the LLM's ability to solve for the correct solution.

6.2 Convergence and Simulation Size

Table 6.1 shows the number of steps required for convergence as we increase the number of agents in the simulation [22].

Table 6.1: Convergence for Different Numbers of Rational Workers

Number of Agents	Convergence Steps
2	10
3	20
5	50
10	120
50	800
100	2000

6.3 Experiments

Having established in Section 6.1 that LLaMa can converge to Stackelberg equilibria, we ran longer experiments with 100 agents. These longer runs allowed us to investigate how the social welfare score of our learned tax policy in the rational scenario, democratic scenario, and democratic scenario with platforms compares to Saez's policy discussed in Section 3.3.1. We also investigate whether different skill distributions affect the resulting social welfare score and learned tax policy.

6.3.1 Skill Distributions

We ran experiments with two distributions of skill levels: a uniform distribution over the skill ranges of the first two income brackets, and an Generalized Beta of the Second Kind (GB2) distribution with an upper bound of skill that equates to an initial income of $10 million.

Creating U.S. Income Distribution Feature

We matched the GB2 distribution to the income distribution of the United States based on census data from the 2023 American Community Survey data from IPUMS USA. We tested

multiple distributions used to model income distributions, which can be seen in Figures 6.2 through 6.5.

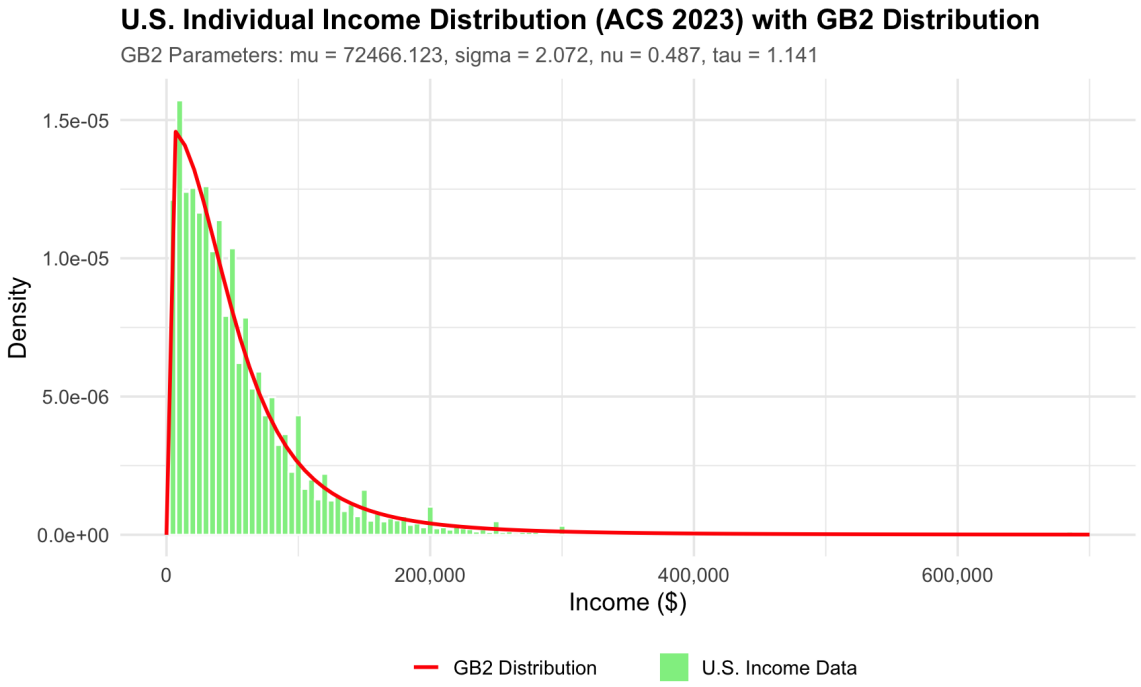


Figure 6.2: GB2 Distribution

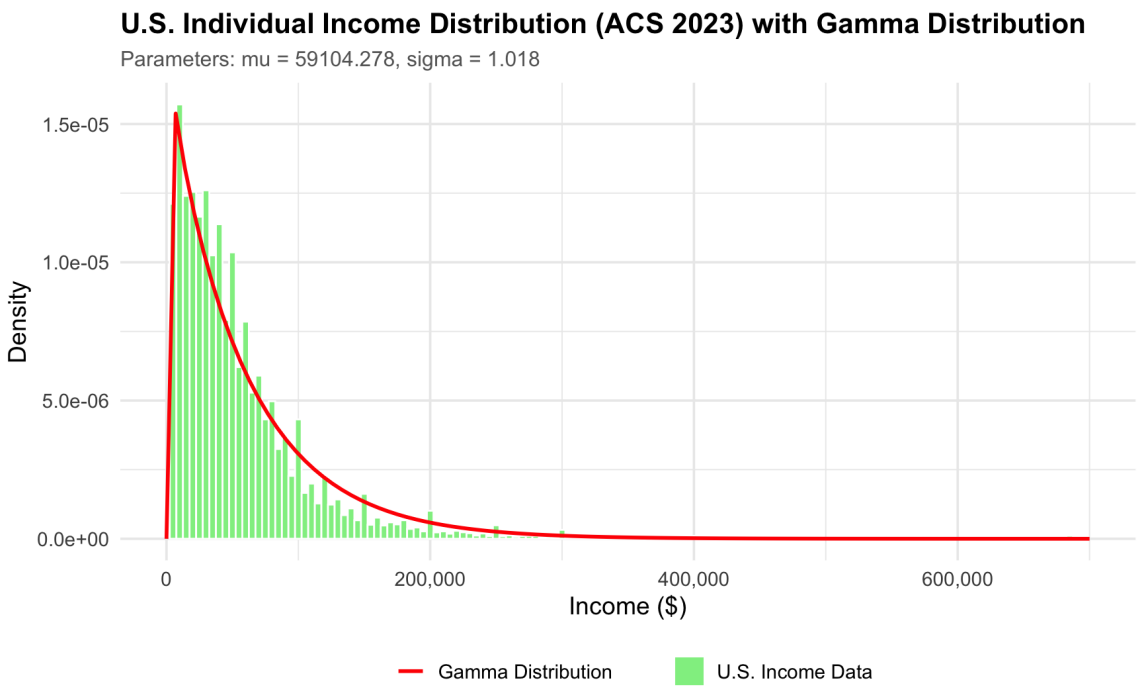


Figure 6.3: Gamma Distribution

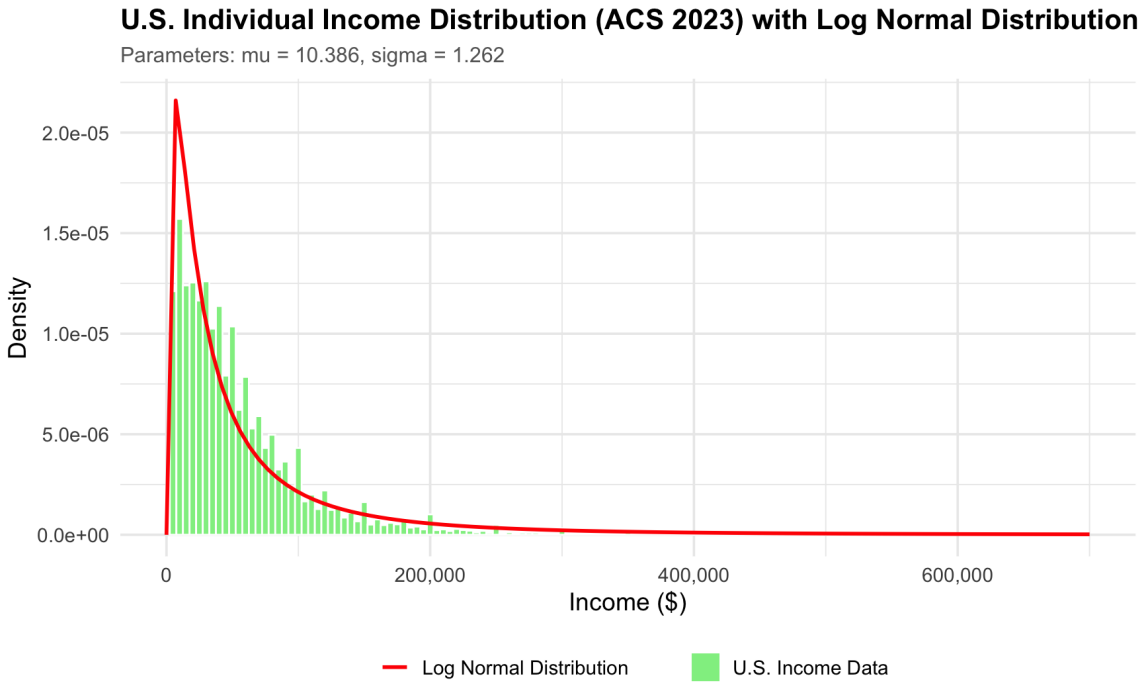


Figure 6.4: Lognormal Distribution

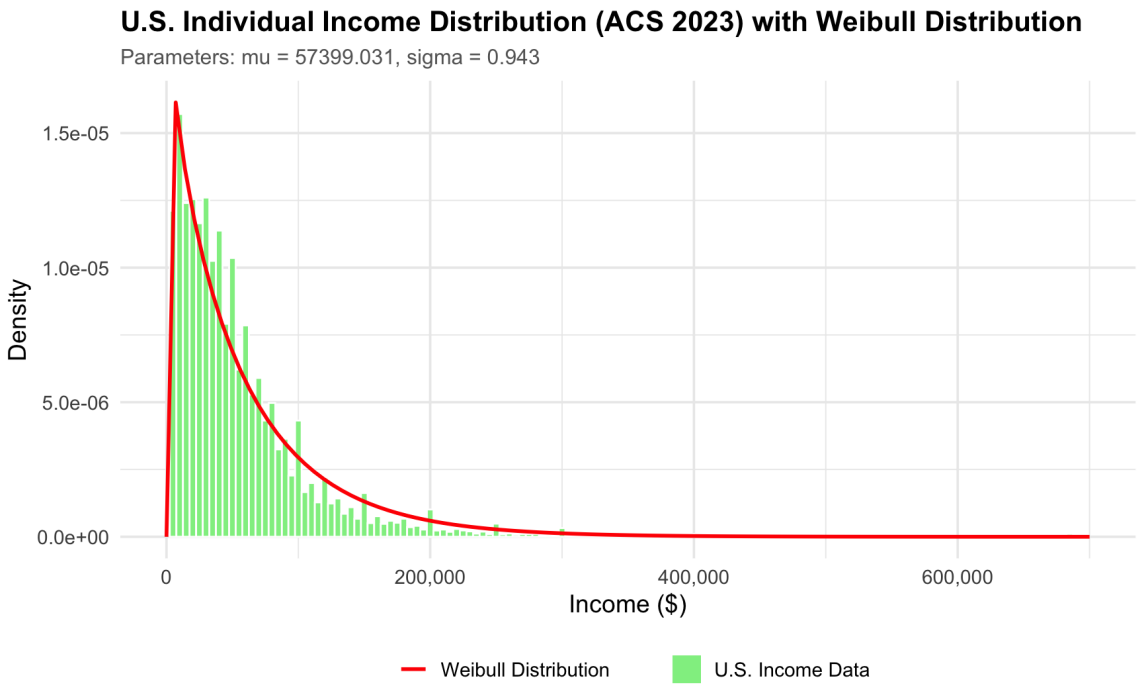


Figure 6.5: Weibull Distribution

These distributions were evaluated by Akaike Information Criterion [AIC] [6] and inspection. An AIC score for a model is calculated from its number of independent variables and maximum likelihood estimate (how likely it is for the model to produce the empirical data). AIC is less affected by issues like ties or huge sample sizes - common in income distributions - compared to Kolmogorov-Smirnov (K-S) tests, which is also commonly used to evaluate models [24]. The Generalized Beta of the Second Kind (GB2) distribution was chosen because it had the lowest AIC score in Table 6.2. In our simulation, we divide the samples from the GB2 distribution by 100 to transform the income number to a skill level, s_i. We assume the fit of the GB2 distribution is sufficient to generate a set of agent skill levels that mimics the U.S. Income distribution.

Table 6.2: AIC Comparison of Different Distributions

Distribution	**Gamma**	**Weibull**	**Log Normal**	**GB2**
AIC	5.78×10^9	5.78×10^9	5.80×10^9	5.76×10^9

6.4 Results

We ran eight experiments with 100 agents each to compare the performance of a Saez optimal tax policy with our learned tax policy, and to explore the emergent phenomena created by different skill distributions. The social welfare and tax rate data from each experiment is shown in Appendix C.1.4.

6.4.1 Social Welfare Scores

We use 0.05 exponential moving average smoothing on our social welfare data.

Uniform Distribution

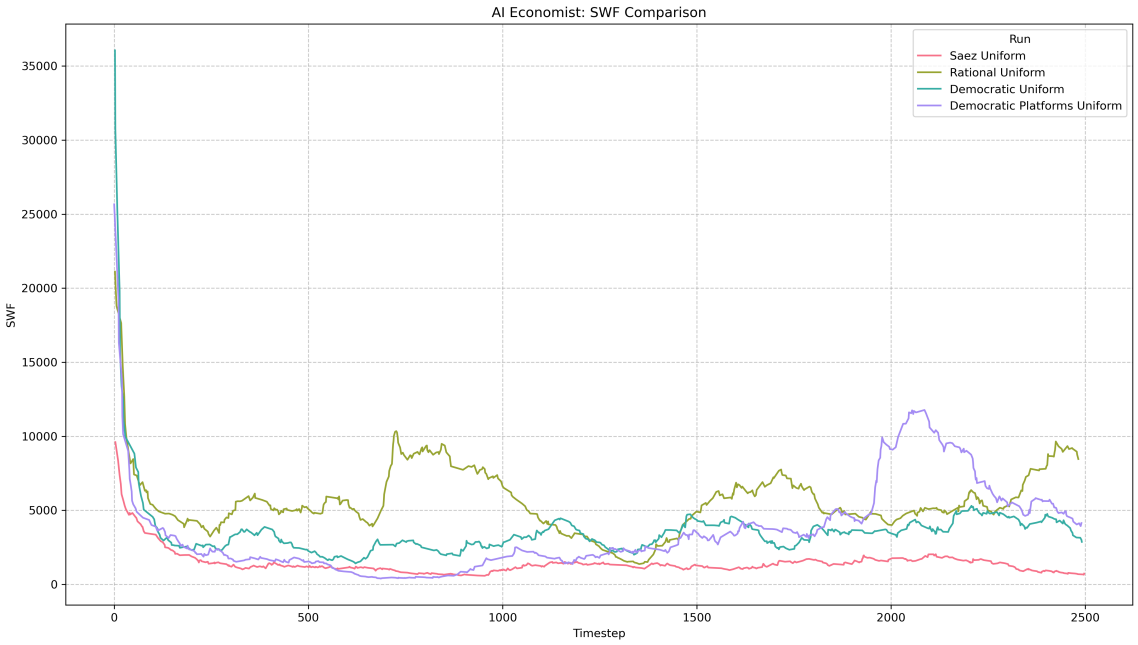


Figure 6.6: Social Welfare

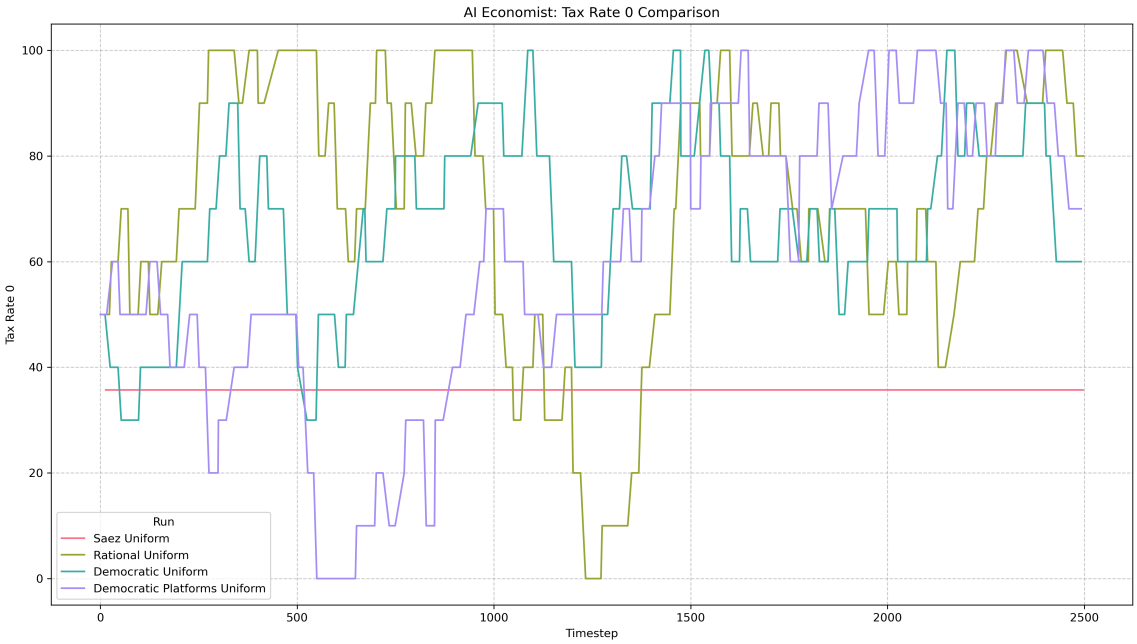


Figure 6.7: Tax Rate for First Bracket

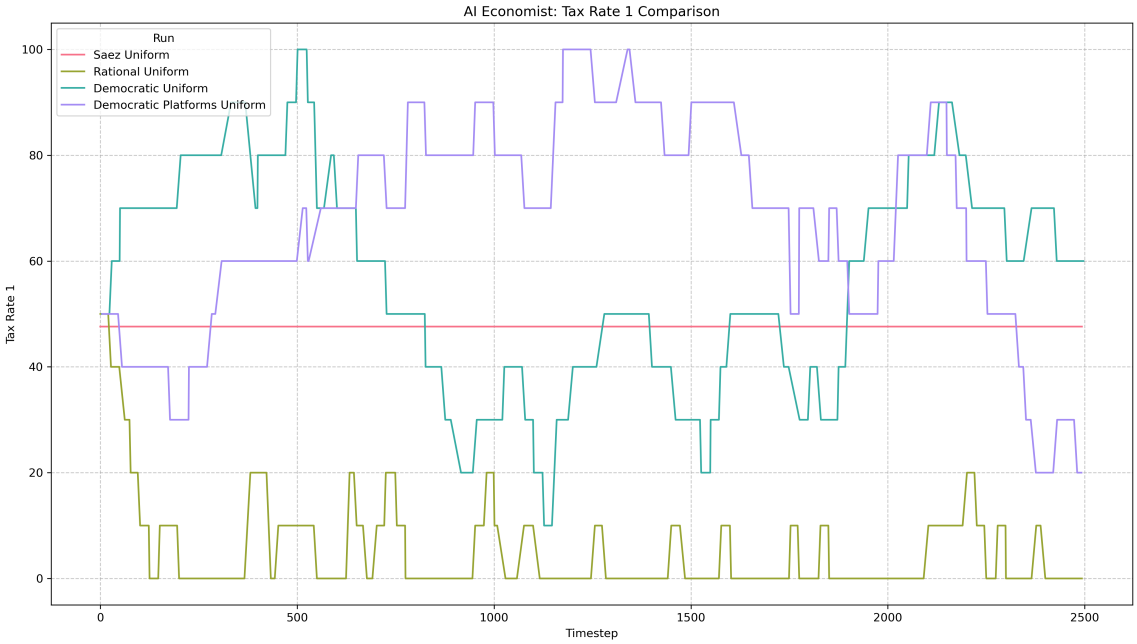


Figure 6.8: Tax Rate for Second Bracket

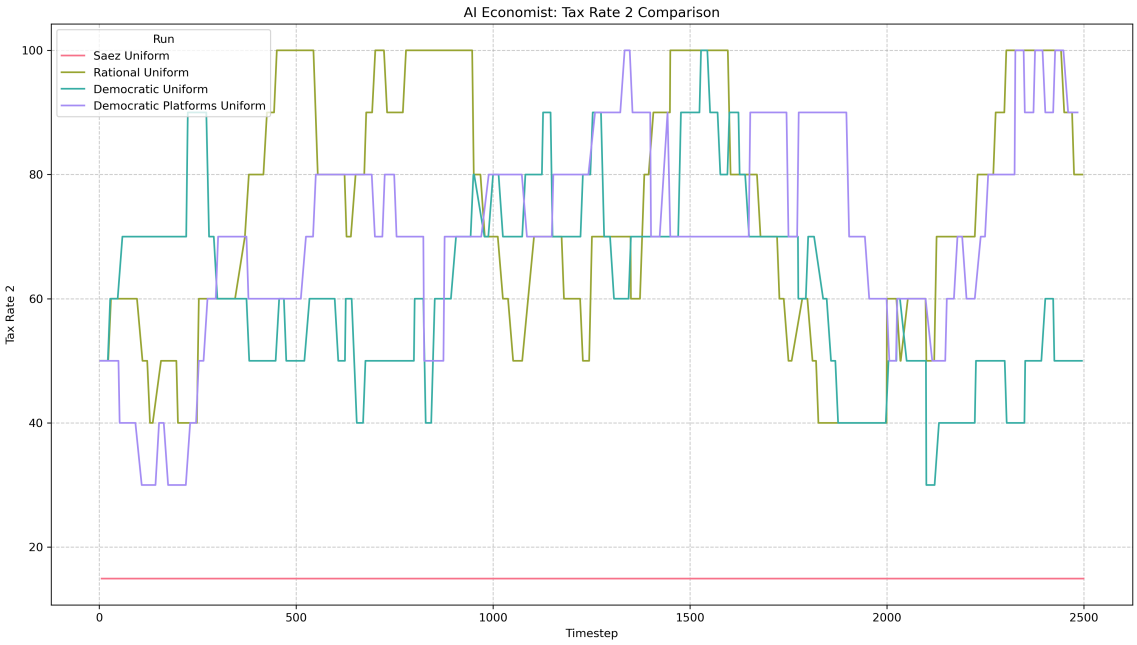


Figure 6.9: Tax Rate for Top Bracket

U.S. Income Distribution

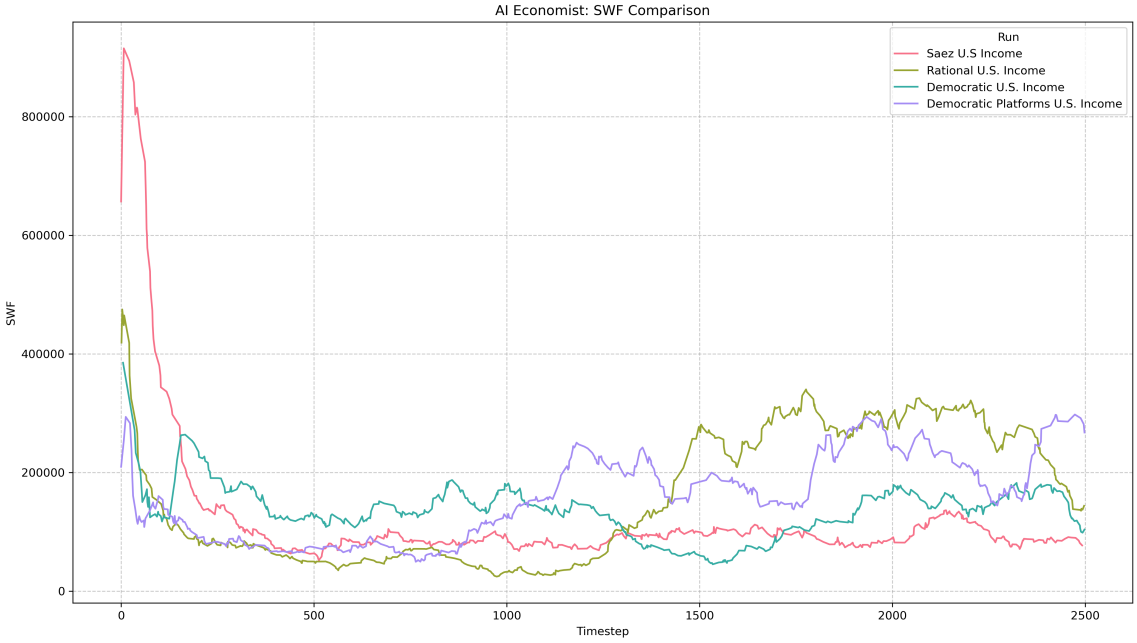


Figure 6.10: Social Welfare: U.S. Income Distribution

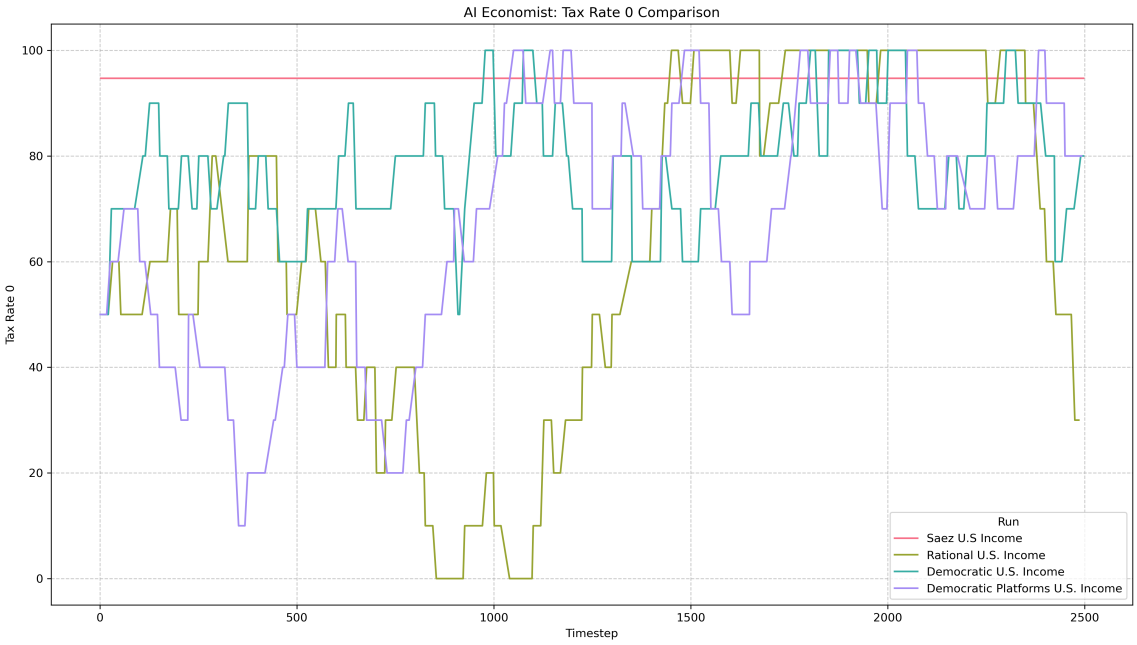


Figure 6.11: Tax Rate for First Bracket

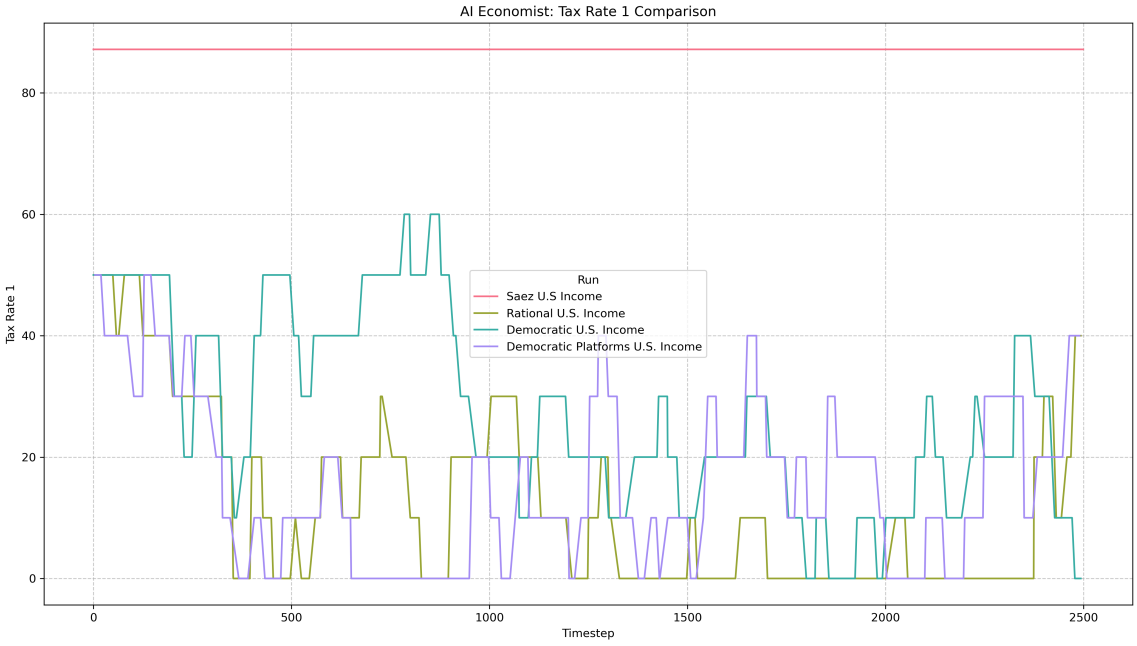


Figure 6.12: Tax Rate for Second Bracket

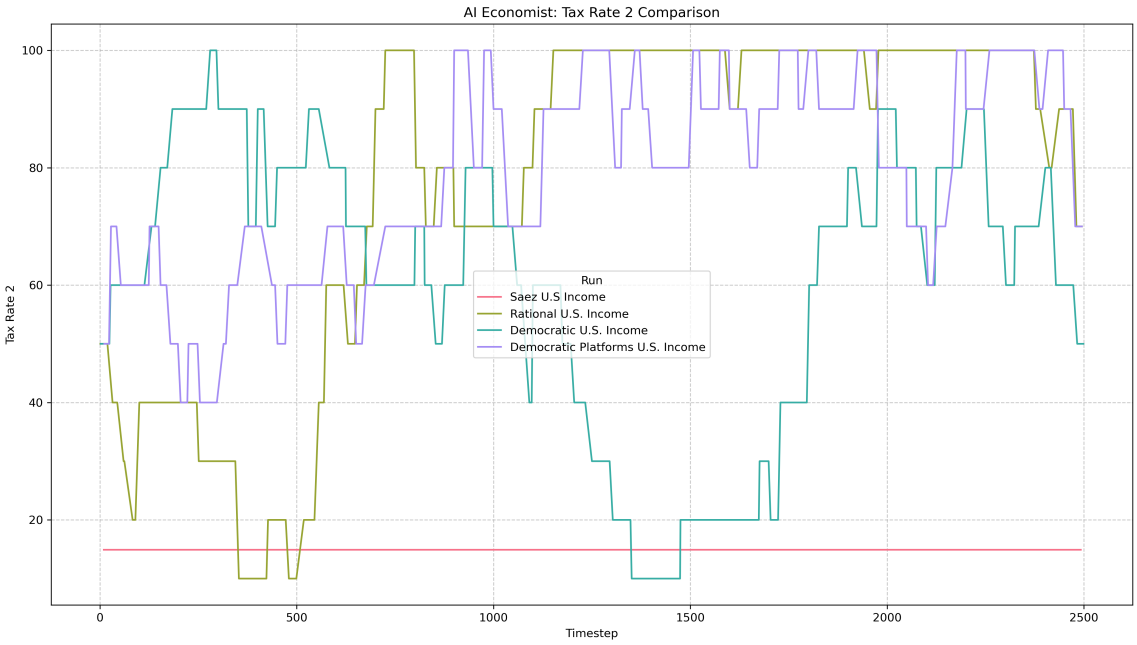


Figure 6.13: Tax Rate for Top Bracket

6.4.2 Comparing Saez's Tax Policy To Our Learned Policies

For accurate analysis, we focus on the results from the last 250 timesteps, which is the 10% of timesteps when the LLM agents are doing pure exploitation.

Uniform Distribution

We see that all of our learned policies across all three scenarios results in higher social welfare than Saez's policy. We note that our learned policies across all simulations runs have higher first and top bracket tax rates than Saez's policy while we learn a variety of second bracket tax rates. Over the last 250 timesteps, we generally see the learned policy from the democratic scenario with platforms result in higher social welfare than the rational scenario, which results in higher social welfare than the democratic scenario.

U.S. Income Distribution

For the U.S income distribution, our learned policies also produce higher social welfare scores in the last 250 timesteps than Saez's policy. We generally see a high tax rate for

the first bracket from both our learned policy and Saez's policy. We see significantly lower learned tax rates for the second bracket than Saez's policy, and significantly higher learned tax rates for the top bracket than Saez's policy. Over the last 250 timesteps, we see the learned policy from the rational scenario finish with the highest social welfare, followed by the democratic scenario with platforms, and then the democratic scenario.

Effect of Skill Distributions on Social Welfare and Policy

The social welfare scores for the U.S. income distribution are all significantly higher than the uniform distribution. This is because the U.S. income distribution has individuals with far higher skill levels than the uniform distribution. This leads to far greater revenue for the government to redistribute to agents with lower incomes. This result demonstrates the importance of highly skilled individuals in raising tax revenue.

We learn a "U-shaped" tax policy - characterized by higher marginal rates at low-incomes and high incomes - for the uniform and U.S. income distributions. In comparison, Saez's policy for the uniform distribution has an approximately flat rate of tax for low and middle incomes, and an extremely low rate of tax for high-earners. This shows that our method learns a tax policy that avoids the result from Saez's formulas that the tax rate should be zero for the highest earner discussed in Section 3.2.3. Saez's policy for the U.S. income distribution also has an extremely low rate of tax for high-earners while increasing the approximately flat rate of tax for low and middle incomes to extremely high rates. Overall, we see that our learned policies and Saez's policy follow the same general shape regardless of the two skill distributions tested.

6.4.3 Elected Leaders in Democratic Scenario Experiments

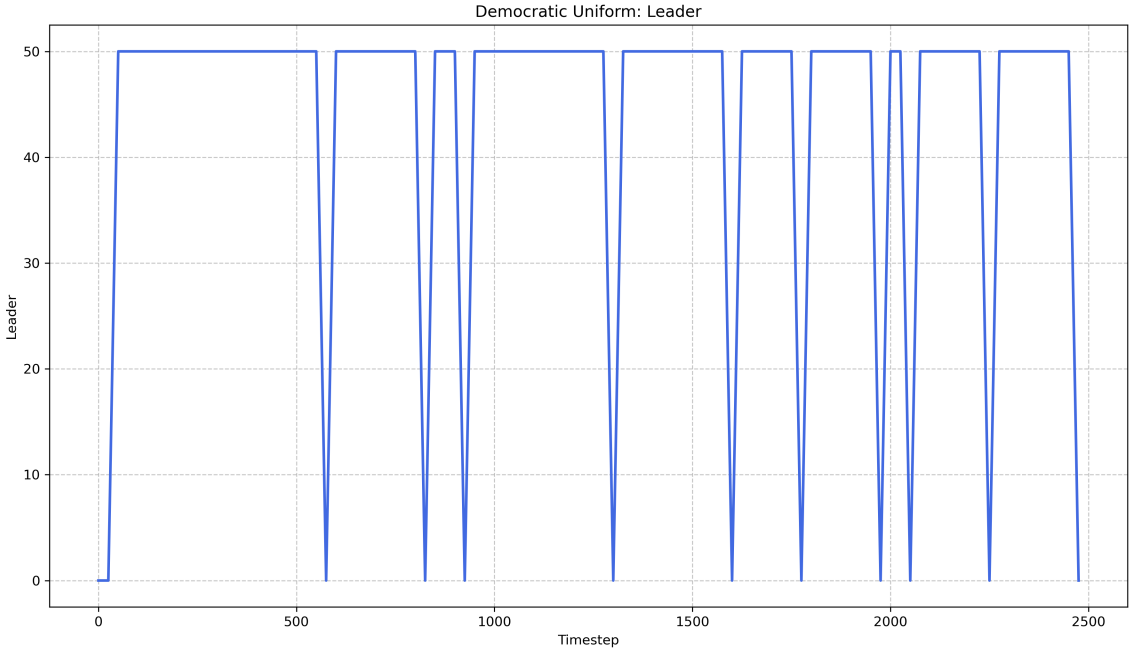


Figure 6.14: Democratic Scenario: Uniform Distribution

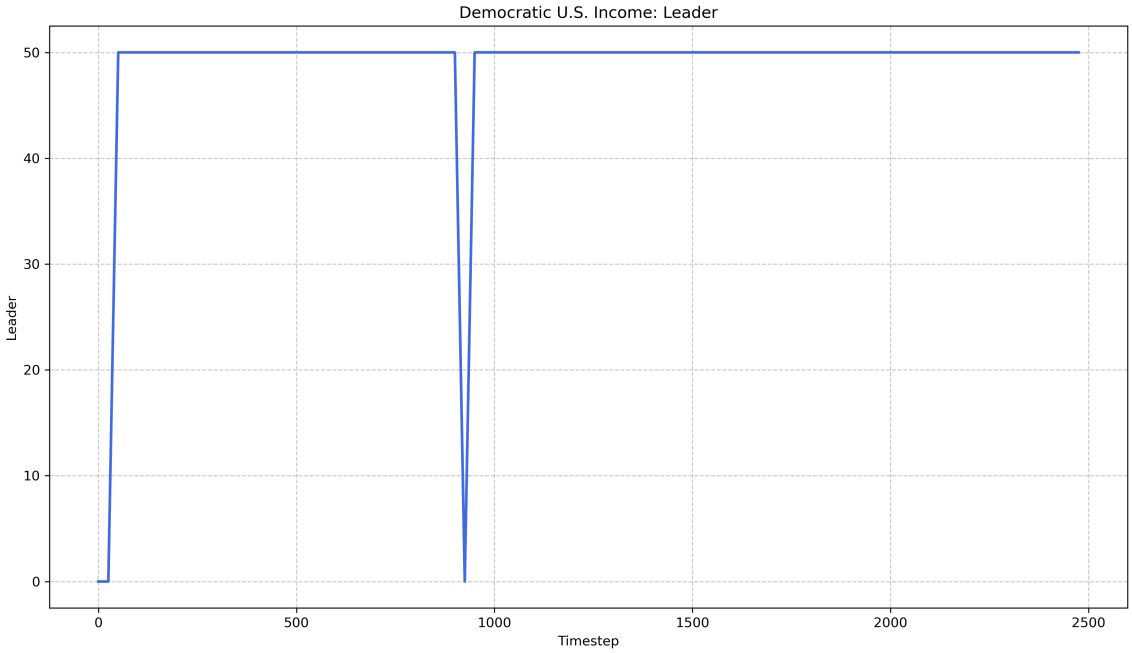


Figure 6.15: Democratic Scenario: U.S. Income Distribution

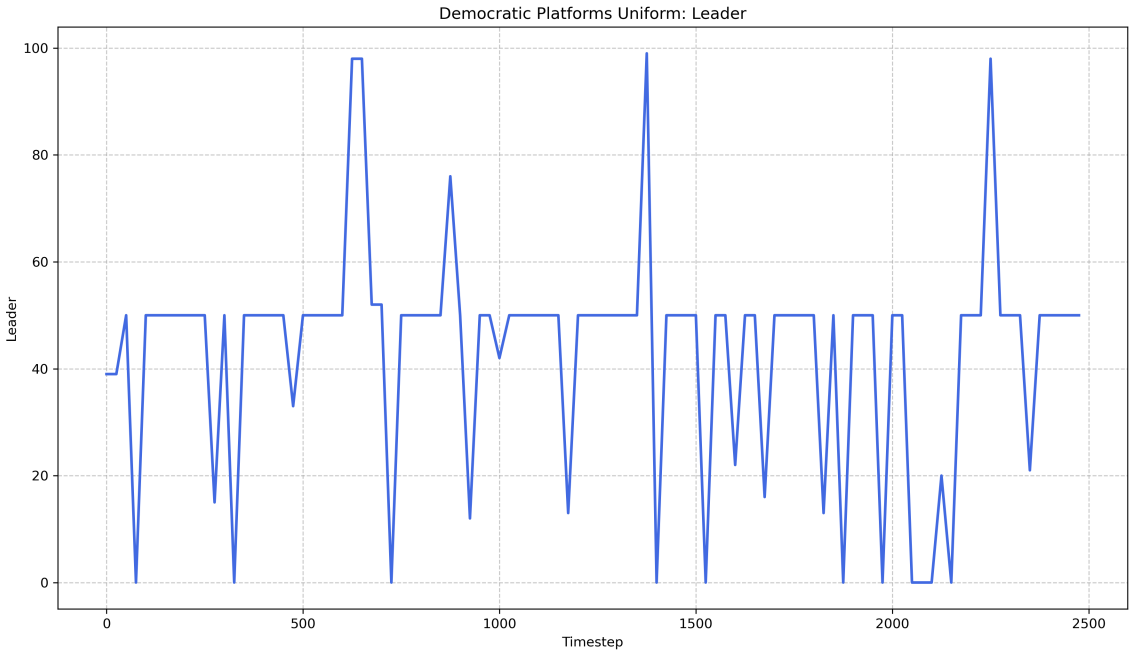


Figure 6.16: Democratic Scenario with Platforms: Uniform Distribution

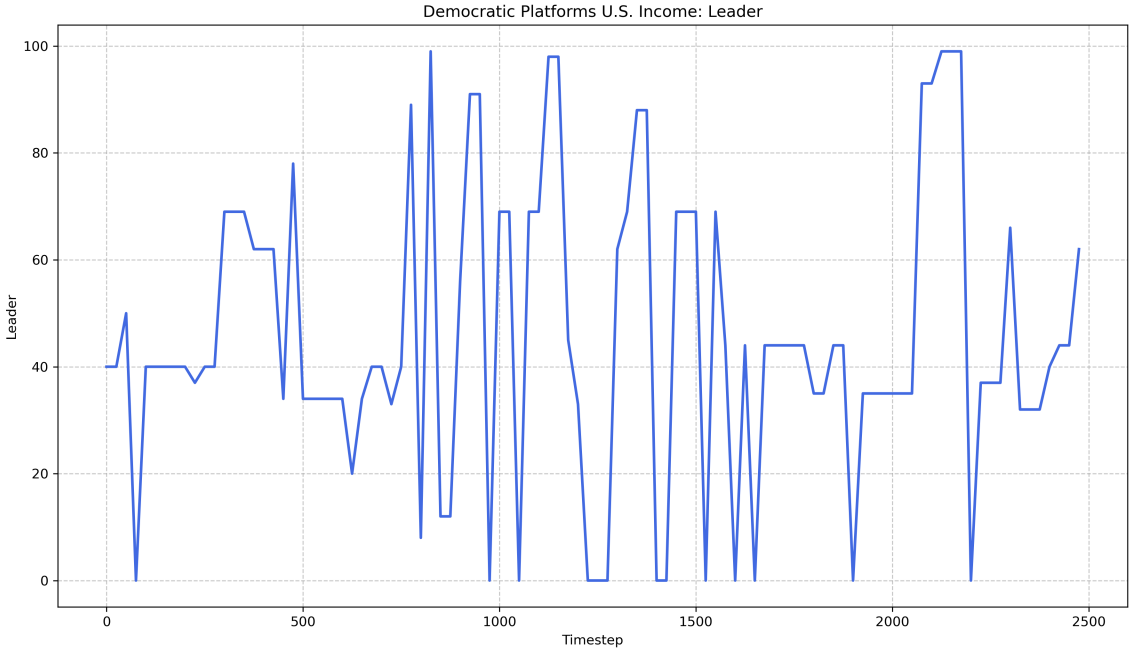


Figure 6.17: Democratic Scenario with Platforms: U.S. Income Distribution

Figures 6.14 through 6.17 show that the platforms feature encouraged more exploration in the worker agents' choice of elected leaders. This result occurred in all of our test runs of

democratic scenarios as well as our final experiments.

Chapter 7

Discussion

The results in Section 6.4 show a clear benefit to learning tax policies compared to Saez's tax policy in our simulation. We do not see a scenario - rational, democratic, or democratic with platforms - that clearly outperforms the others. Considering that the addition of the elections in the democratic scenarios and the addition of the platforms feature were implemented to increase realism, it is positive to see that the additional complexity introduced by those features did not result in a significant decrease in social welfare compared to the rational scenario. This result demonstrates that the additional complexity did not push the LLM past its ability to learn a good policy in a complicated optimization problem.

We established in Section 6.1 that LLMs have the ability to converge towards optimal Stackelberg equilibria using in-context learning. We demonstrated the power of learned policies to create higher social welfare than the policy produced by Saez's optimal income taxation formulas in simulations with 100 agents where calculating the Stackelberg equilibria through backwards induction is intractable. The rapid advancement in LLM capabilities promises significant improvements in the realism and accuracy of our simulation and any economic simulations that incorporate LLMs. These economic simulations with LLMs enable governments and economists to experiment extensively with policy alternatives, advancing both practical mechanism design and theoretical developments in optimal income taxation theory.

We recognize that nuanced historical factors and interest groups influence real-world tax

policy. Furthermore, there is an ethical concern - discussed in Section 1.5 - in assuming that LLMs accurately mimic human preferences. Yet, as fierce disagreement among economists shows, optimal income taxation theory is far from a solved field. Ultimately, our work is a meaningful advancement toward realistic, affordable, and computationally accessible methods for policy experimentation and mechanism design. By effectively integrating LLMs into economic modeling, this approach holds great promise for advancing optimal income taxation theory and enhancing policy-making processes worldwide.

Chapter 8

Future Work

8.1 Future Directions

We plan to explore a variety of new simulation scenarios in future work.

8.1.1 Influence of Utility Distributions

We plan to investigate the effect of different agent utility mixes on social welfare. We plan to investigate the effect of agent utilities other than isoelastic for the worker agents and social welfare for the planner agent. We could have a "greedy" tax planner that focused on maximizing its own isoelastic utility when planning. We also could use "altruistic" worker agents whose objective is positive social welfare, or "adversarial" worker agents whose objective is negative social welfare. In this future work, we plan to investigate what percentage of altruistic agents are needed to influence the group's social welfare, and if there is a mathematical relationship that we can establish between agent utility mixes and social welfare. For the democratic scenario, can we learn a mathematical relationship between utility mixes, social welfare, and election results? We can also ask questions about the rate of convergence with different agent utility mixes.

8.1.2 Multi-LLM Interactions

We plan to explore scenarios where multiple LLMs are used simultaneously within a single simulation to evaluate emergent behaviors. Would one LLM be able to take advantage of a less powerful LLM to increase the utility of the stronger LLM's generative agent?

8.1.3 Multi-Agent Communication

We plan to implement communication channels between agents to see emergent interaction patterns. This could involve agents exchanging messages to align on collective goals, negotiate trade-offs, or provide feedback on the tax planner's policy. This future direction is inspired by [15]. We want to know how information propagates through the simulation, and how that information could affect election results.

8.1.4 Extensions to Saez's Optimal Income Taxation Theory

We plan to implement several extensions to Saez's optimal income taxation formulas that have been made to incorporate more elements of human economic activity. Future work could add these features to our simulation with the goal of achieving more optimal tax policies with these more complicated scenarios as well. Our simulation currently does not consider migration effects (explained in Appendix E.1.1) where agents can migrate between tax jurisdictions. Future work could explore a multi-agent simulation with two competing tax jurisdictions. We also do not currently incorporate tax avoidance responses (explained in Appendix E.1.3), or rent seeking effects (explained in Appendix E.1.4).

Appendix A

Engineering and Industrial Standards

The independent project described in this thesis incorporated the following engineering and industrial standards:

A.1 Programming Languages

- Python: A Python codebase was developed to run our multi-agent simulation
- R: Used to make plots
- LaTeX: Used to write this thesis

A.2 Software

- Overleaf: Used to write my thesis
- slurm: Open-source job scheduler used to run experiments on Princeton's Della HPC

A.2.1 Industry-Wide Accepted File Standards

- .txt: Used for various instructions and record-keeping

- .csv: Used for containing U.S. Income Data, and samples from GB2 distribution representing U.S. Income Data

A.2.2 Large Language Models

Open Source:

- llama3:8b: Used for the majority of testing and all final experiments

Closed Source:

- gpt-4o-mini-2024-07-18: Used for some small tests

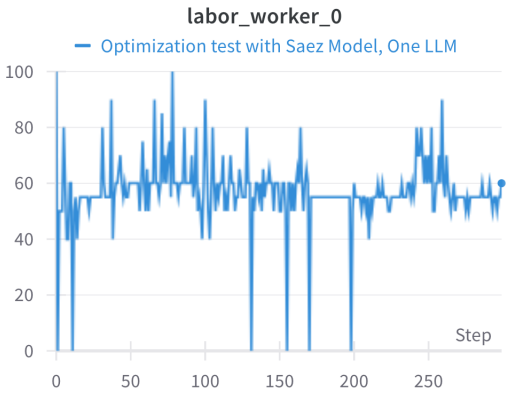
A.3 Artifical Intelligence Ethical Standards

According to the standards established by ISO/IEC TR 24368:2022, we require anyone who uses this system, a modification of it, or our results to be transparent about what model they are using. Users must recognize that the preferences of different socioeconomic groups are not necessarily evenly represented in LLMs' training data. To use this simulation to inform policy, we recommend an accountability framework that includes human oversight. Users should establish a review process where tax policy experts validate model-generated simulation results. Furthermore, they should include mechanisms for citizens to provide feedback on how accurately their preferences are being modeled.

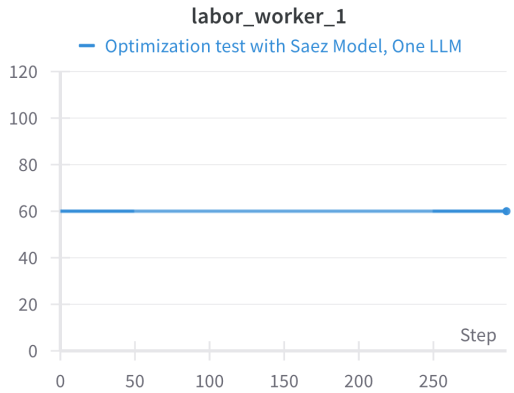
Appendix B

Ablations

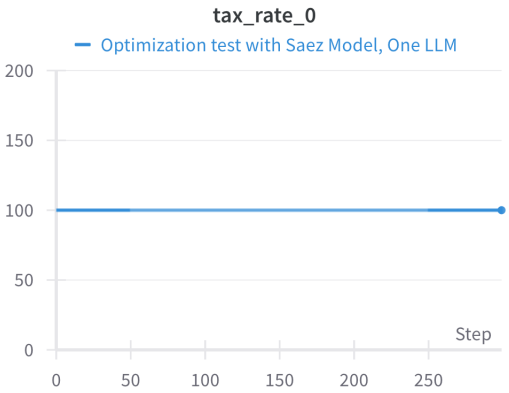
B.1 One LLM Worker, Fixed Tax Planner



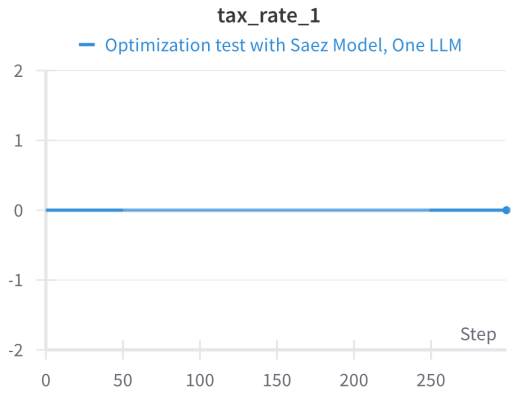
(a) Labor Worker_0, LLM



(b) Labor Worker_1, Fixed



(c) Tax Rate for First Bracket, Fixed

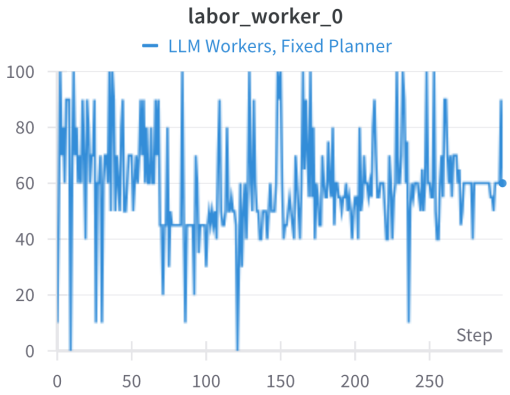


(d) Tax Rate for Second Bracket, Fixed

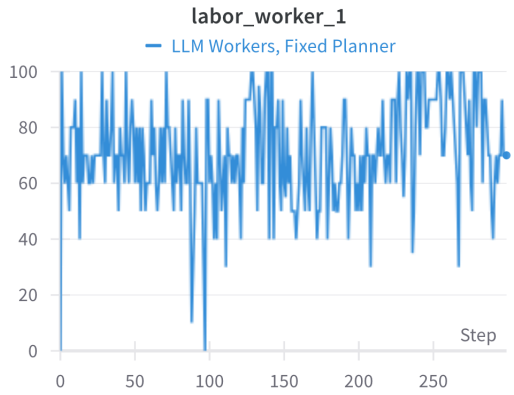
Figure B.1: Ablation Study Results: One LLM Worker, Fixed Tax Planner

Figure B.1 shows that when the tax planner's rates are fixed to [100, 0] and one worker's labor output is fixed to 60, LLaMa found that the other worker's optimal labor output to maximize its isoelastic utility was 60, which matches the results of the Stackelberg equilibria. Therefore, with one LLM worker, a fixed worker, and a fixed tax planner, we can see that LLaMa finds the correct solution with a labor output of approximately 60 for the LLM worker.

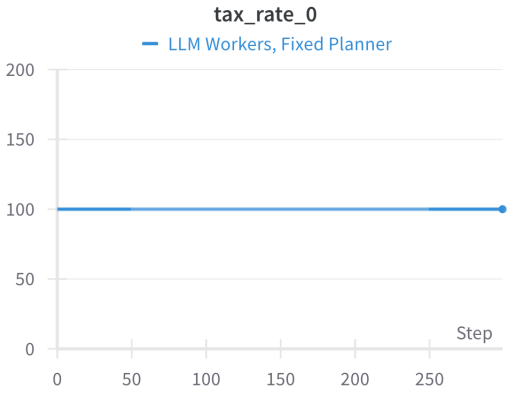
B.2 LLM Workers, Fixed Tax Planner



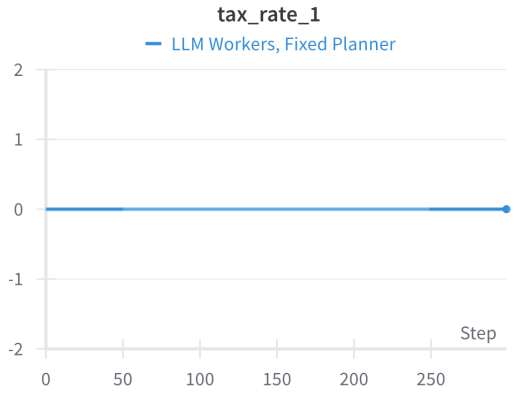
(a) Labor Worker_0, LLM



(b) Labor Worker_1, LLM



(c) Tax Rate for First Bracket, Fixed

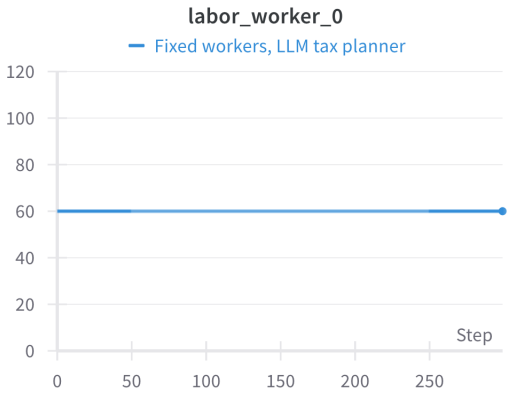


(d) Tax Rate for Second Bracket, Fixed

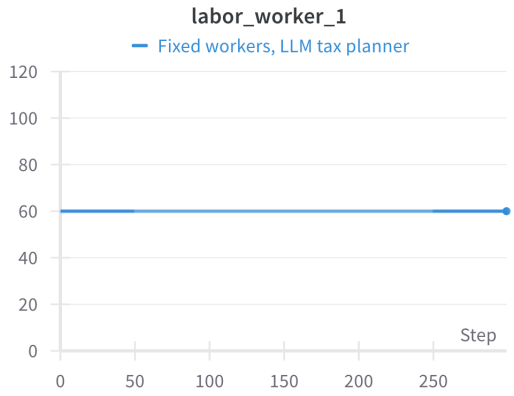
Figure B.2: Ablation Study Results: LLM Workers, Fixed Tax Planner

We can see that for a fixed tax planner and two LLM workers, LLaMa still finds the correct solution.

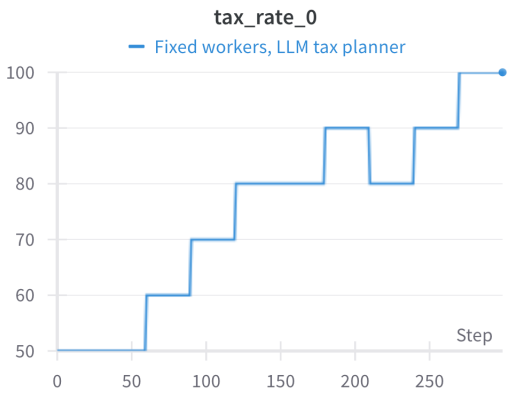
B.3 Fixed Workers, LLM Tax Planner



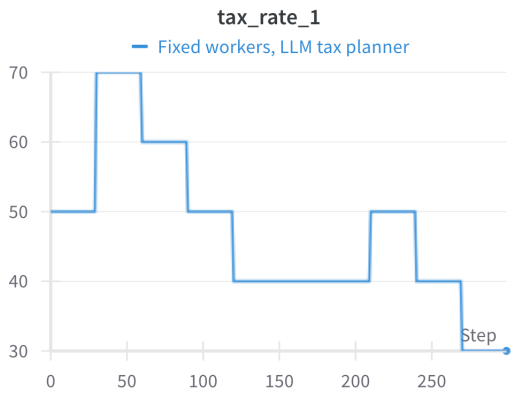
(a) Labor Worker_0, Fixed



(b) Labor Worker_1, Fixed



(c) Tax Rate for First Bracket, LLM

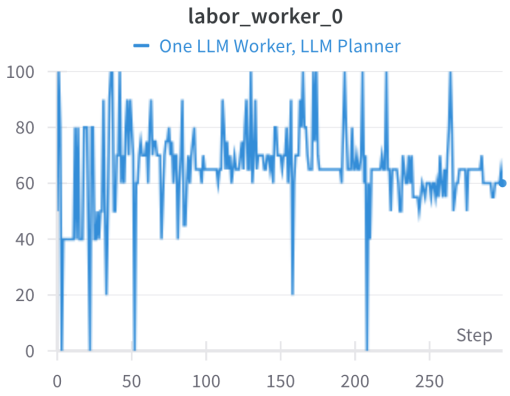


(d) Tax Rate for Second Bracket, LLM

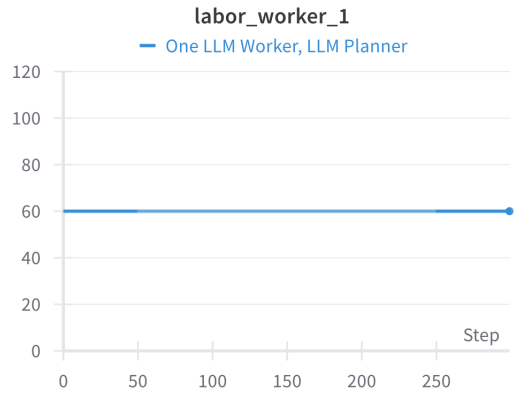
Figure B.3: Ablation Study Results: Fixed Workers, LLM Tax Planner

We can see that for an LLM tax planner, and two fixed workers, LLaMa finds the correct solution again.

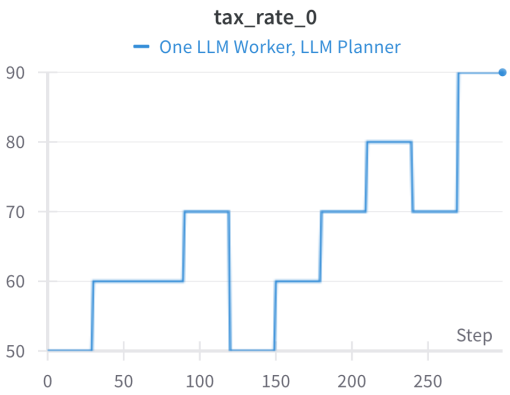
B.4 One LLM Worker, LLM Tax Planner



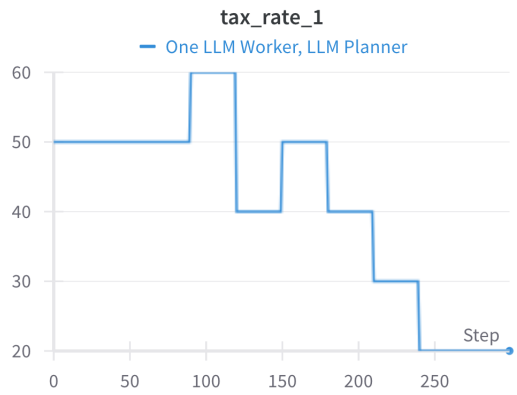
(a) Labor Worker_0, LLM



(b) Labor Worker_1, Fixed



(c) Tax Rate for First Bracket, LLM



(d) Tax Rate for Second Bracket, LLM

Figure B.4: Ablation Study Results: One LLM Worker, LLM Tax Planner

We can see that for an LLM tax planner, one fixed worker, and one LLM worker, LLaMa finds the correct solution.

Appendix C

Experiments

C.1 Data from 100 Agent Simulation Runs

Here is the social welfare and tax rate data from each of our eight 100 agent simulation runs.

C.1.1 Saez Planner

Uniform Distribution

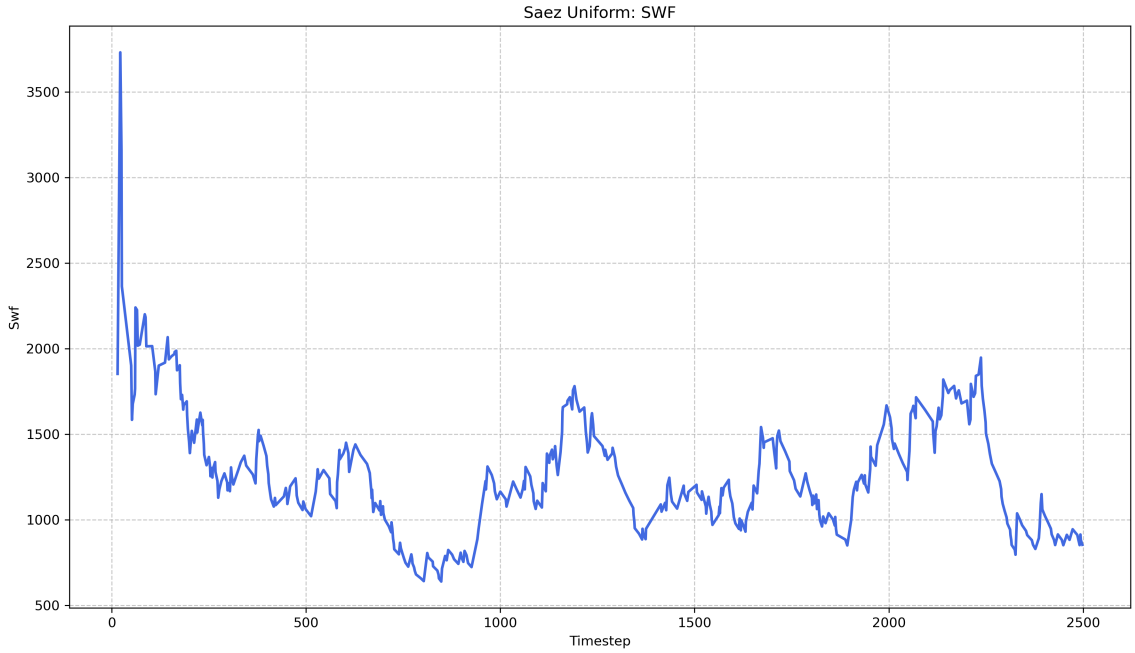


Figure C.1: Social Welfare Results

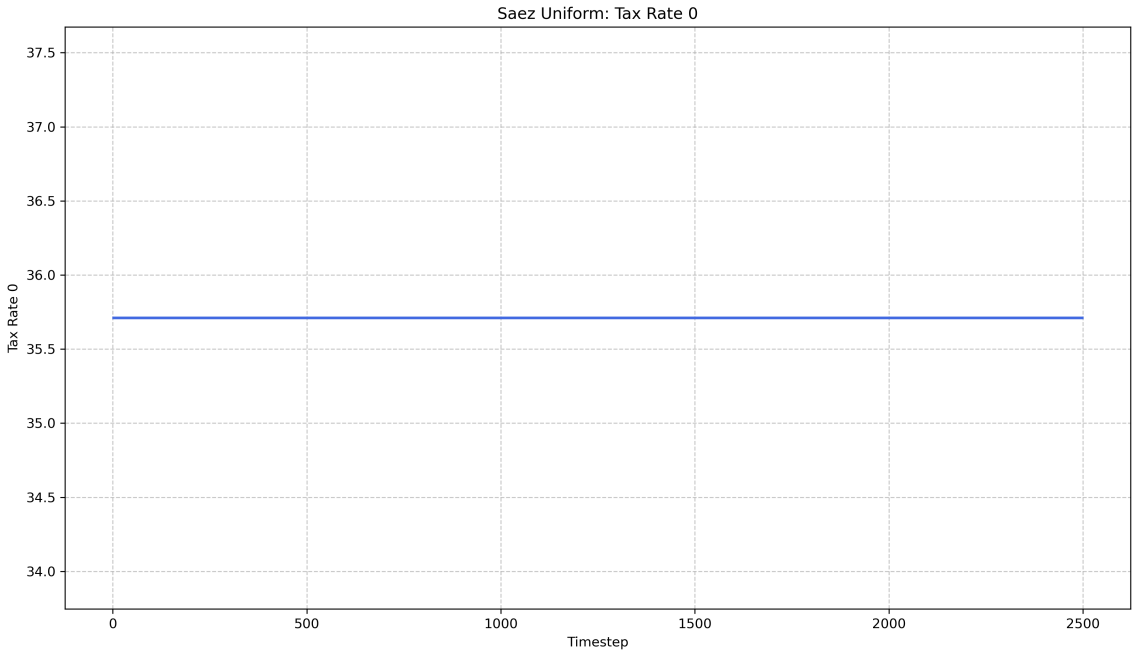


Figure C.2: Tax Rate for First Bracket

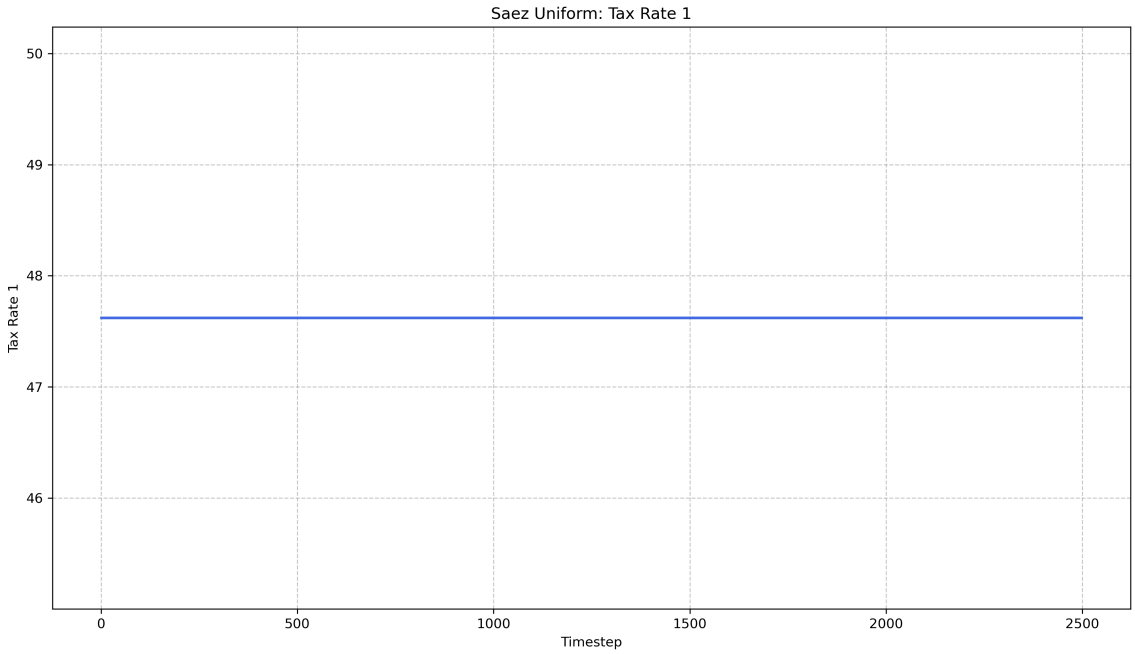


Figure C.3: Tax Rate for Second Bracket

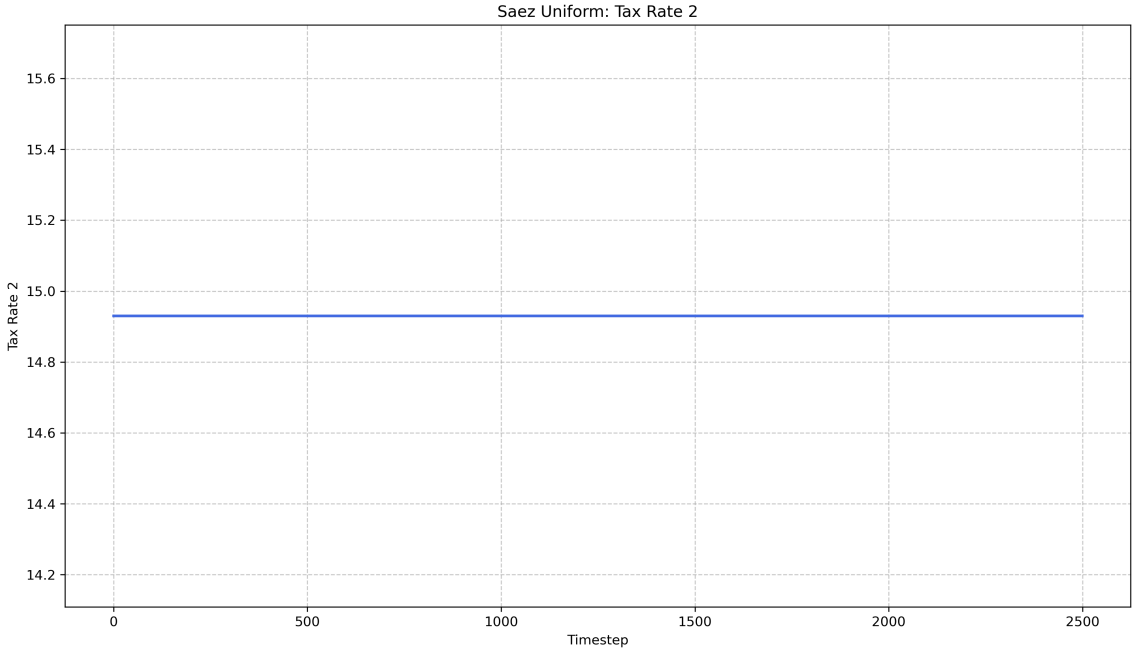


Figure C.4: Tax Rate for Top Bracket

U.S. Income Distribution

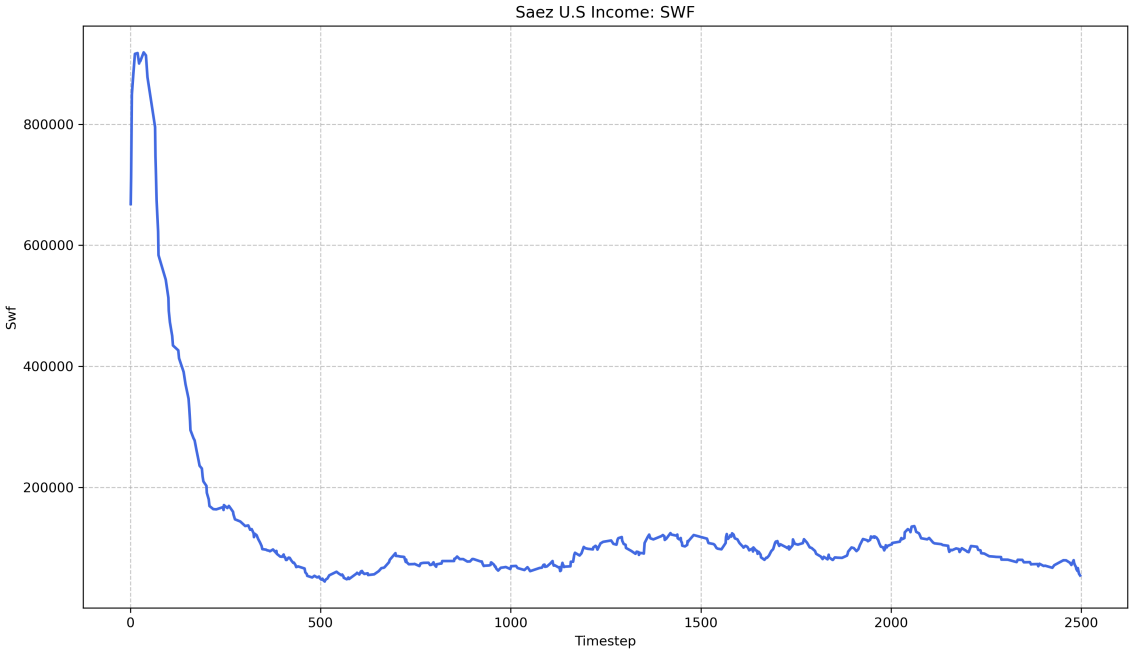


Figure C.5: Social Welfare Results

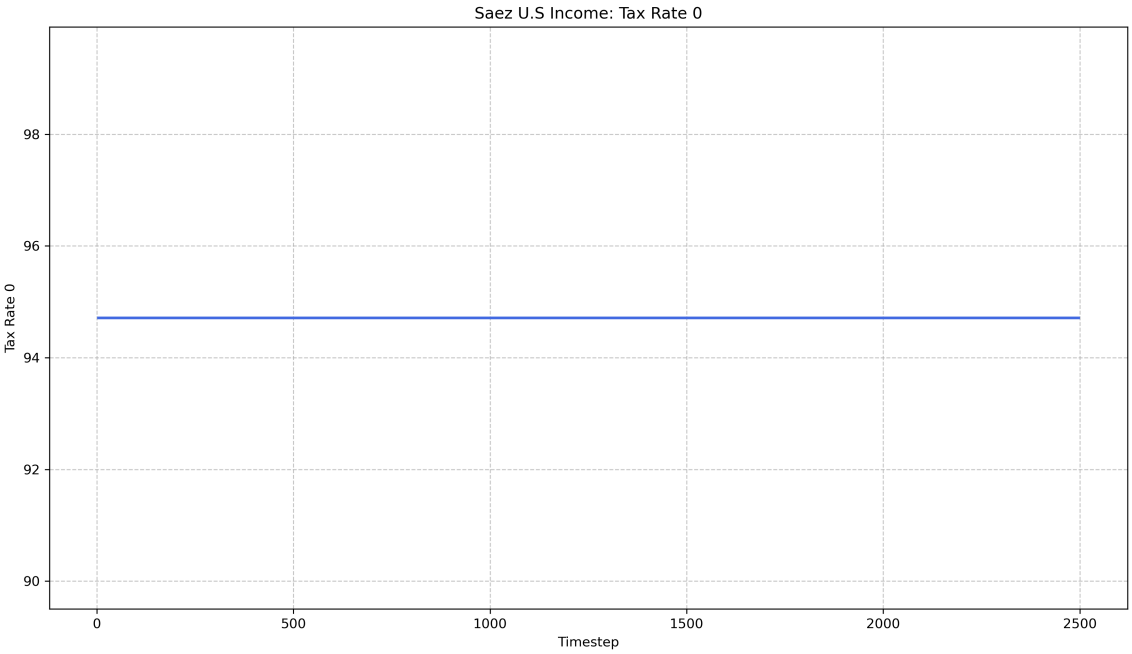


Figure C.6: Tax Rate for First Bracket

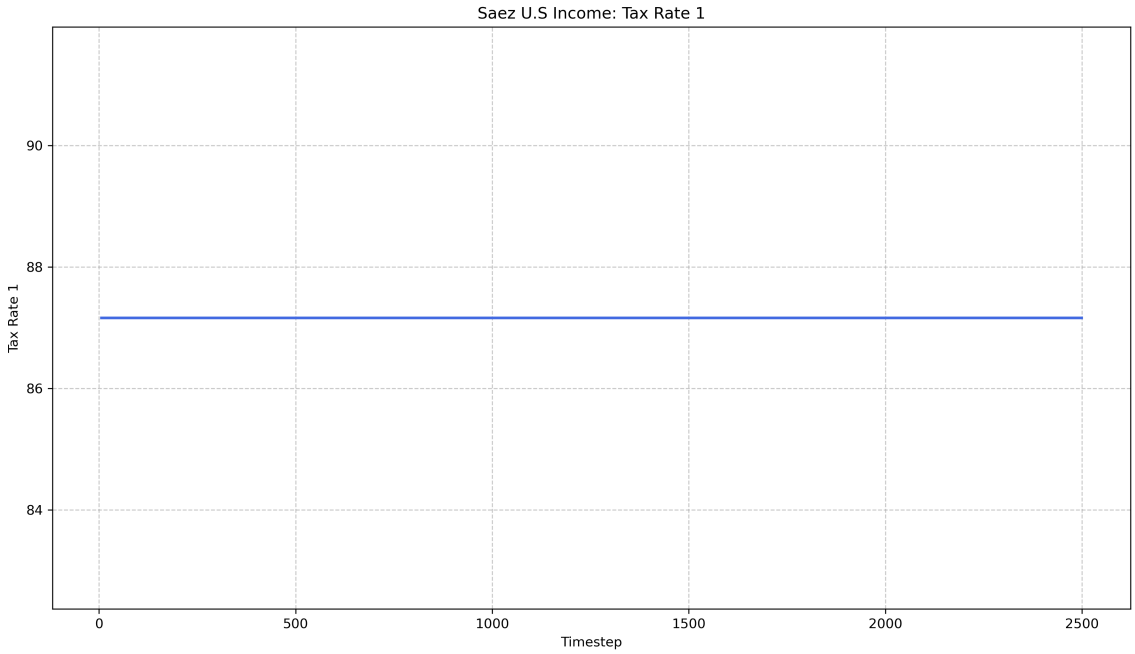


Figure C.7: Tax Rate for Second Bracket

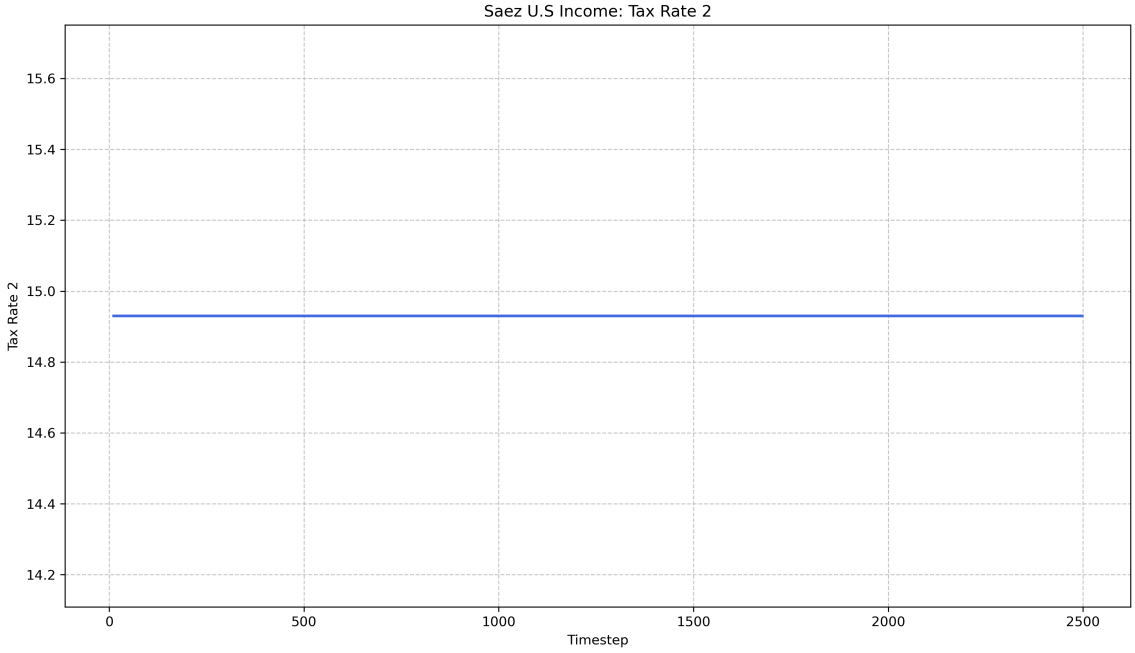


Figure C.8: Tax Rate for Top Bracket

C.1.2 Rational Scenario

Uniform Distribution

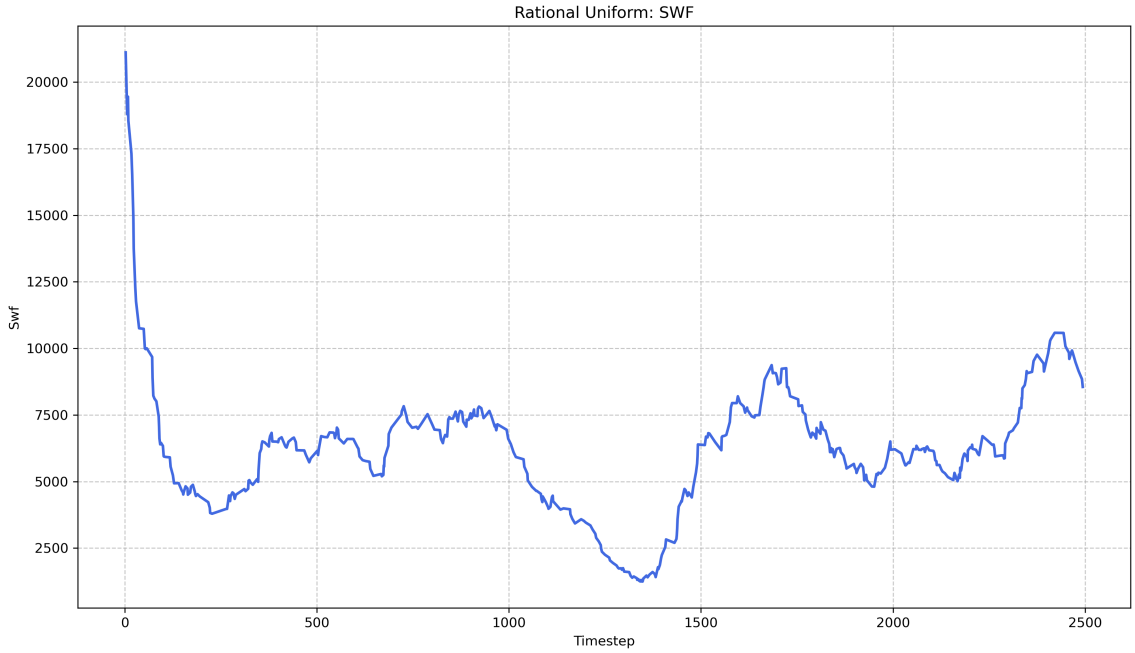


Figure C.9: Social Welfare Results

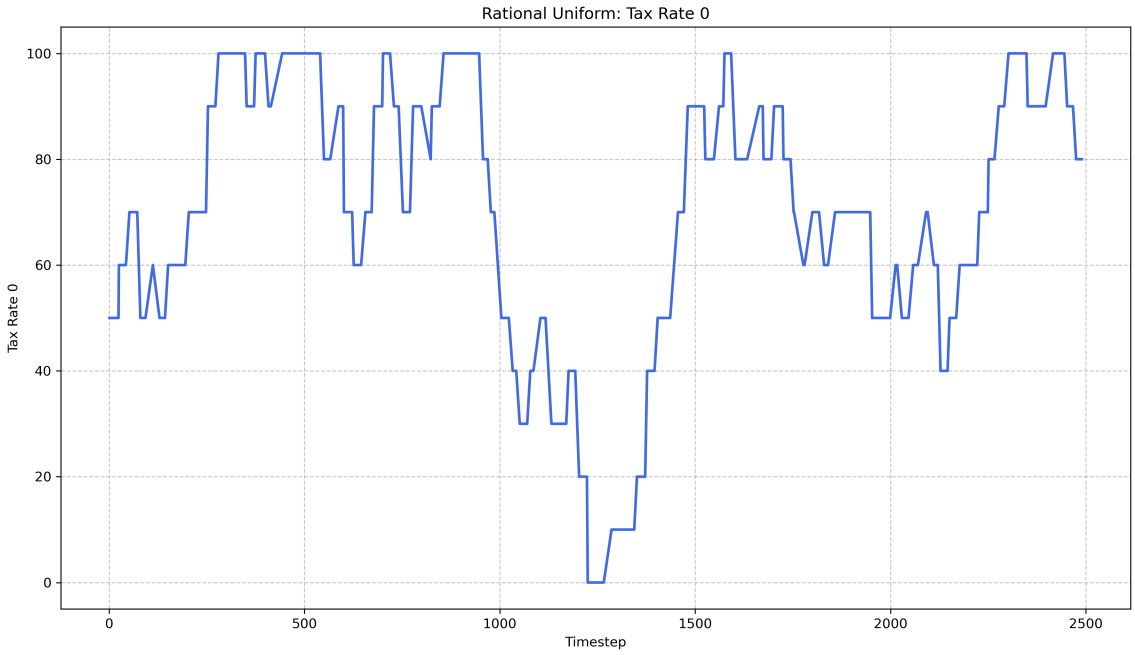


Figure C.10: Tax Rate for First Bracket

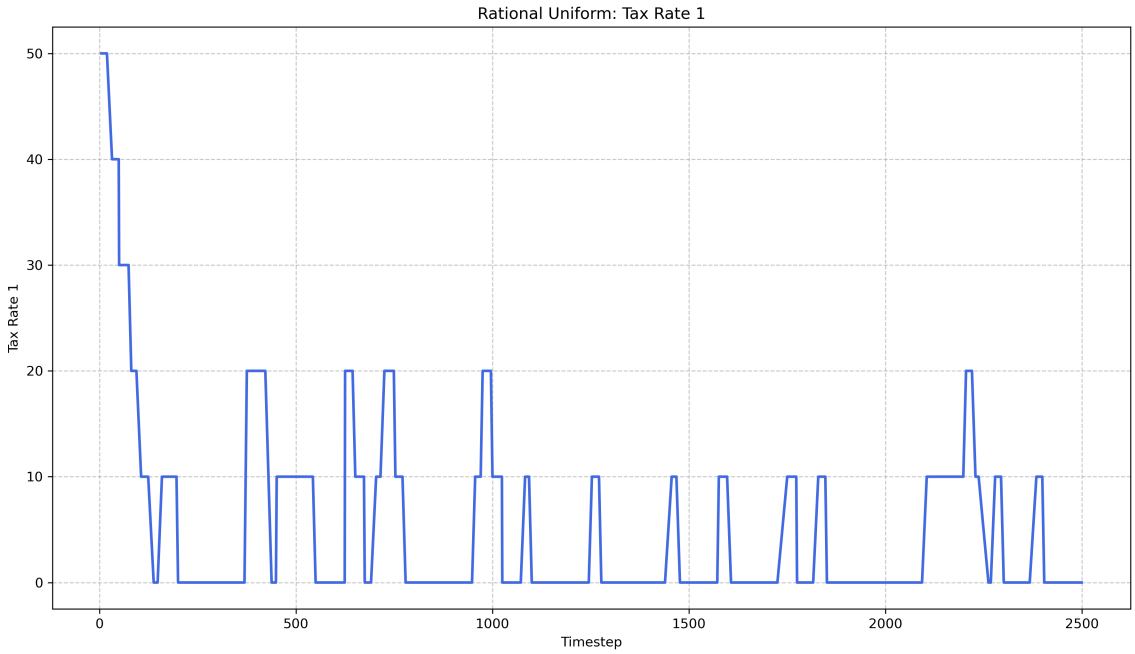


Figure C.11: Tax Rate for Second Bracket

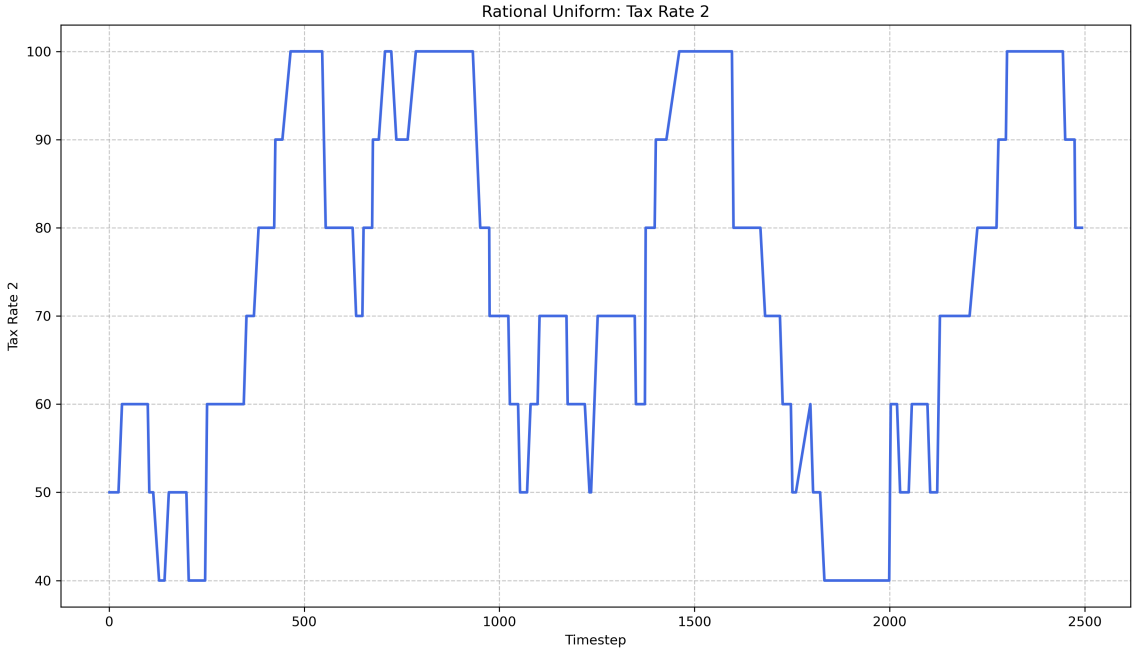


Figure C.12: Tax Rate for Top Bracket

U.S. Income Distribution

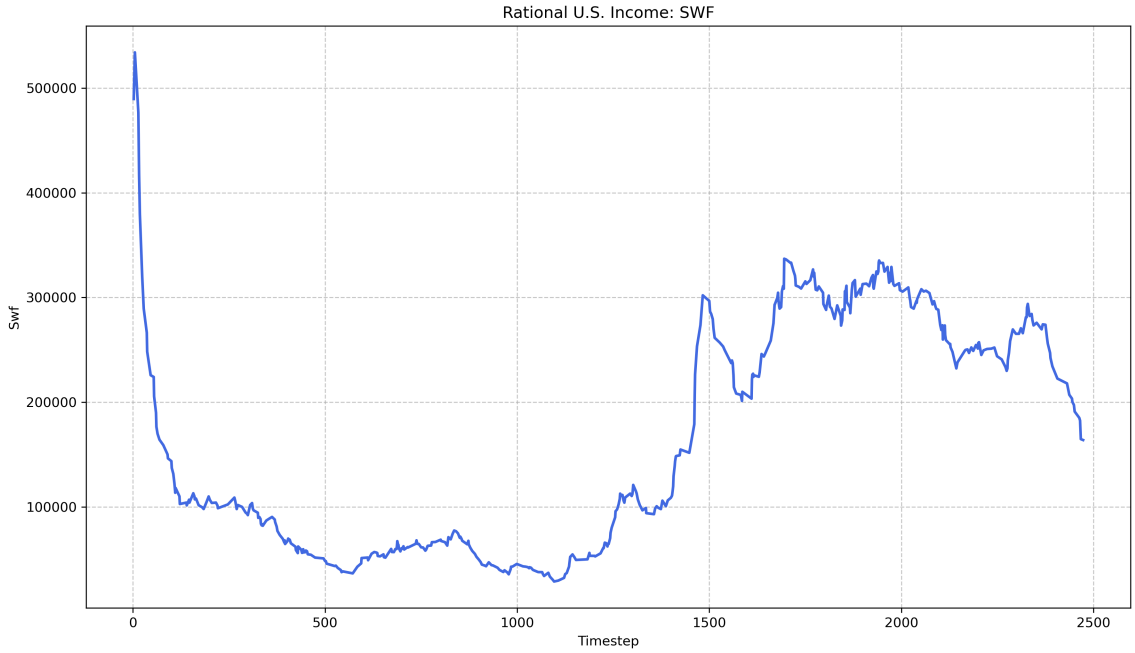


Figure C.13: Social Welfare Results

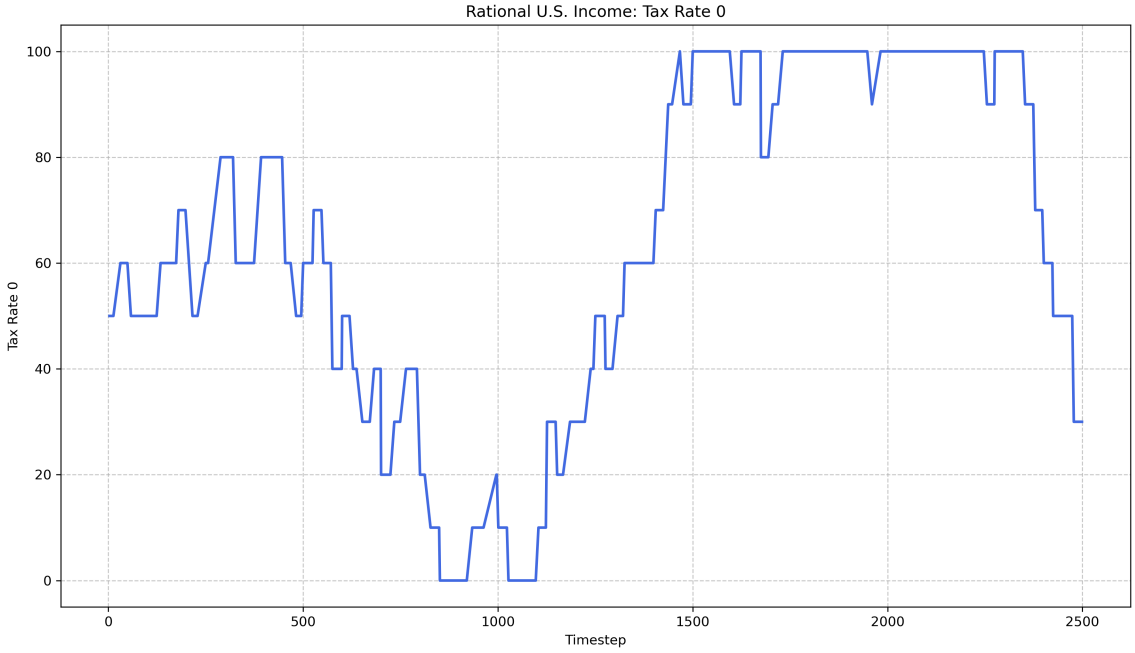


Figure C.14: Tax Rate for First Bracket

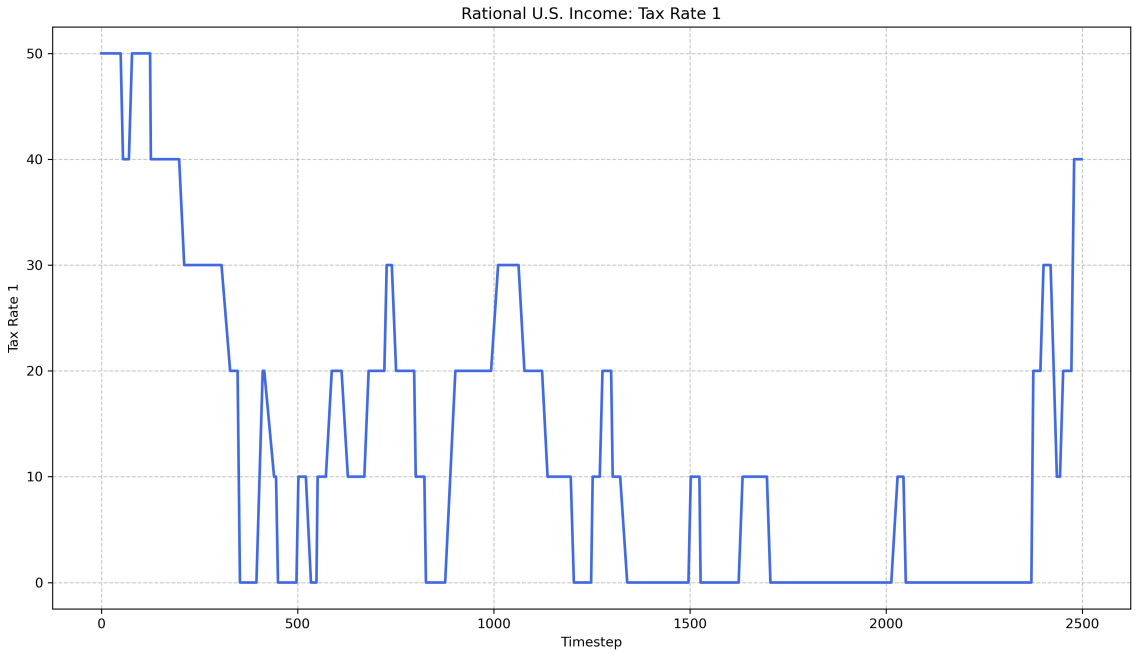


Figure C.15: Tax Rate for Second Bracket

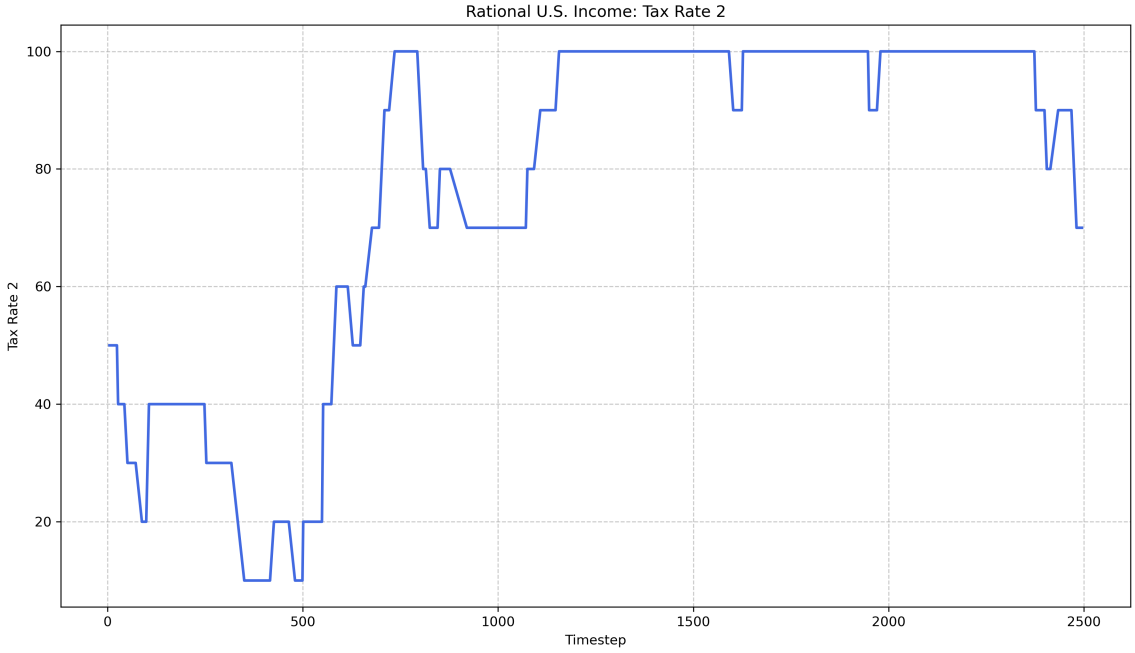


Figure C.16: Tax Rate for Top Bracket

C.1.3 Democratic Scenario

Uniform Distribution

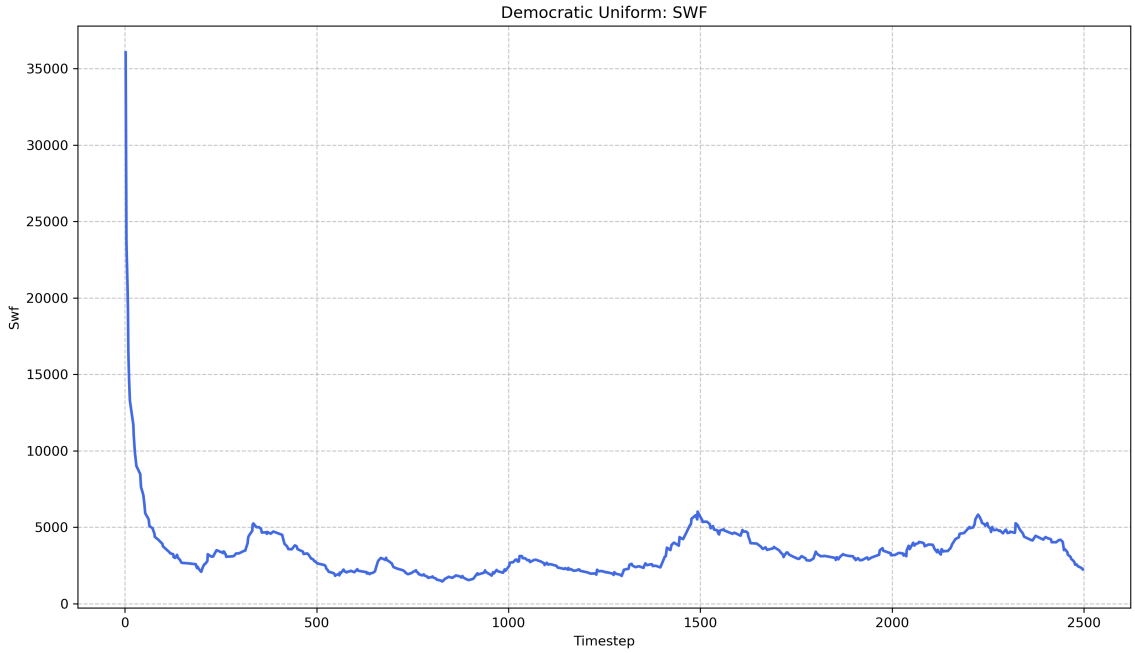


Figure C.17: Social Welfare Results

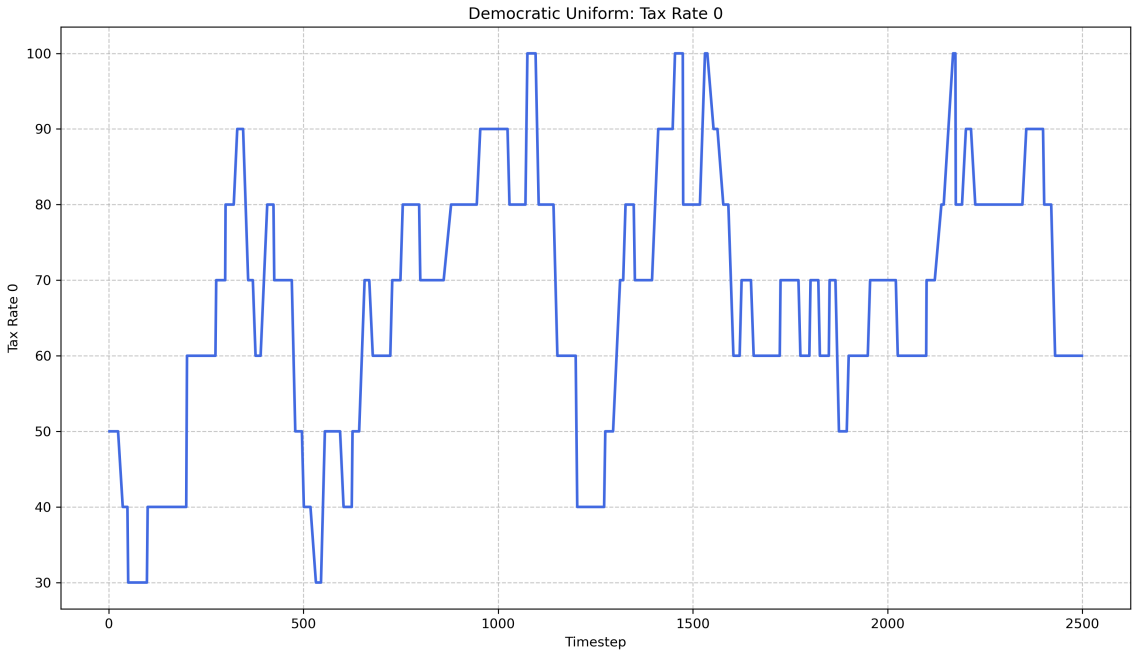


Figure C.18: Tax Rate for First Bracket

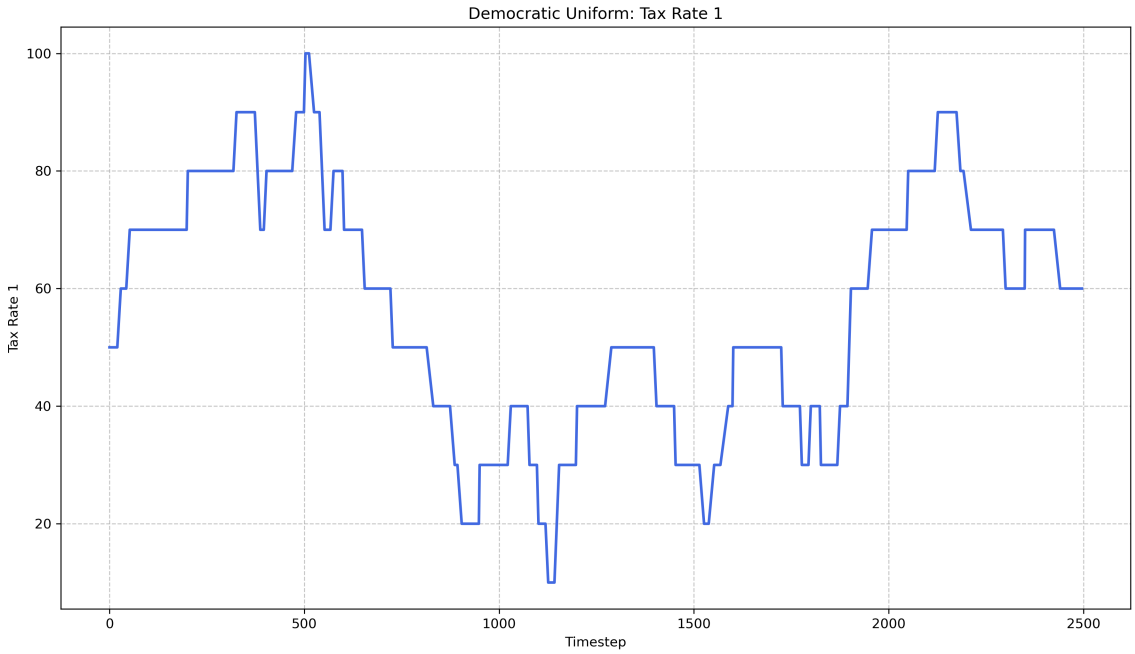


Figure C.19: Tax Rate for Second Bracket

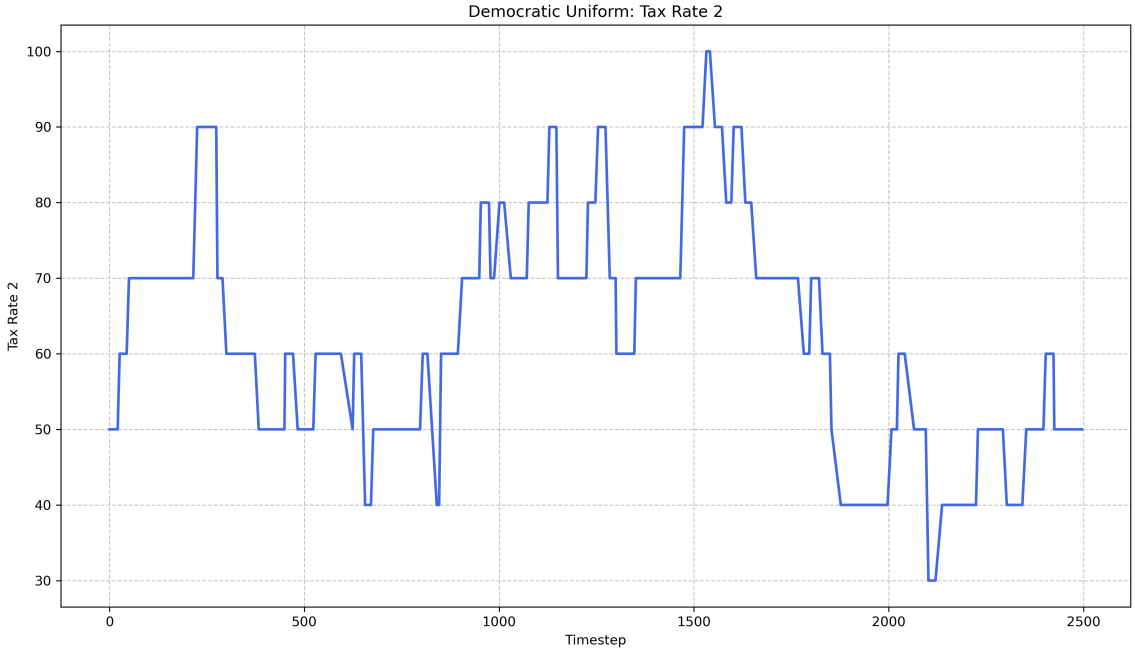


Figure C.20: Tax Rate for Top Bracket

U.S. Income Distribution

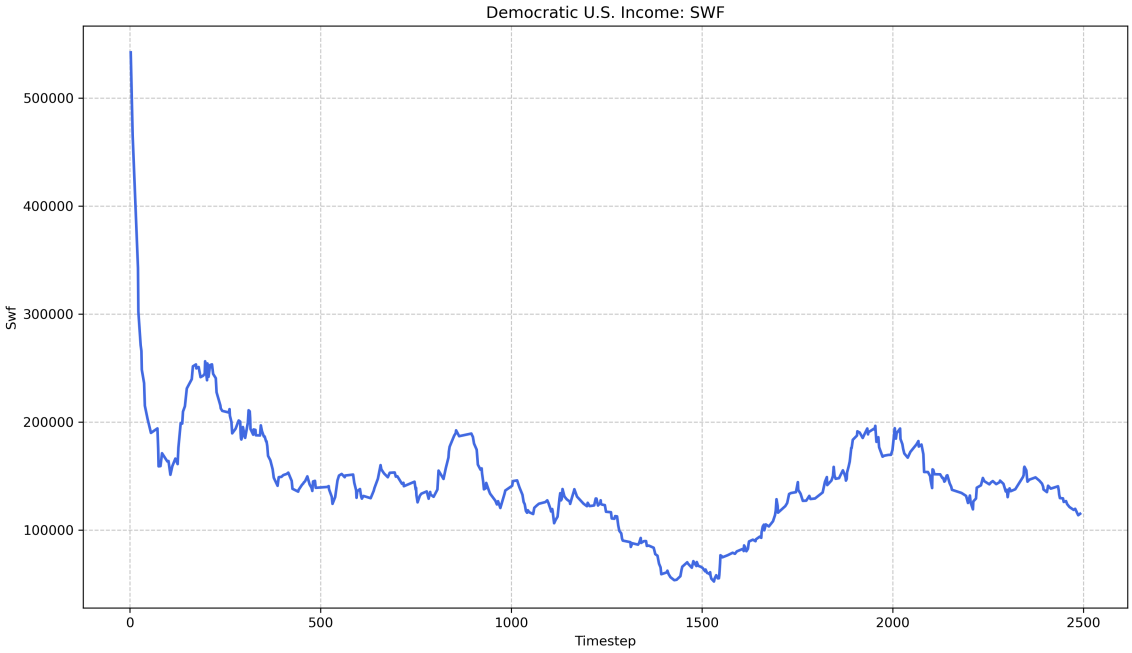


Figure C.21: Social Welfare Results

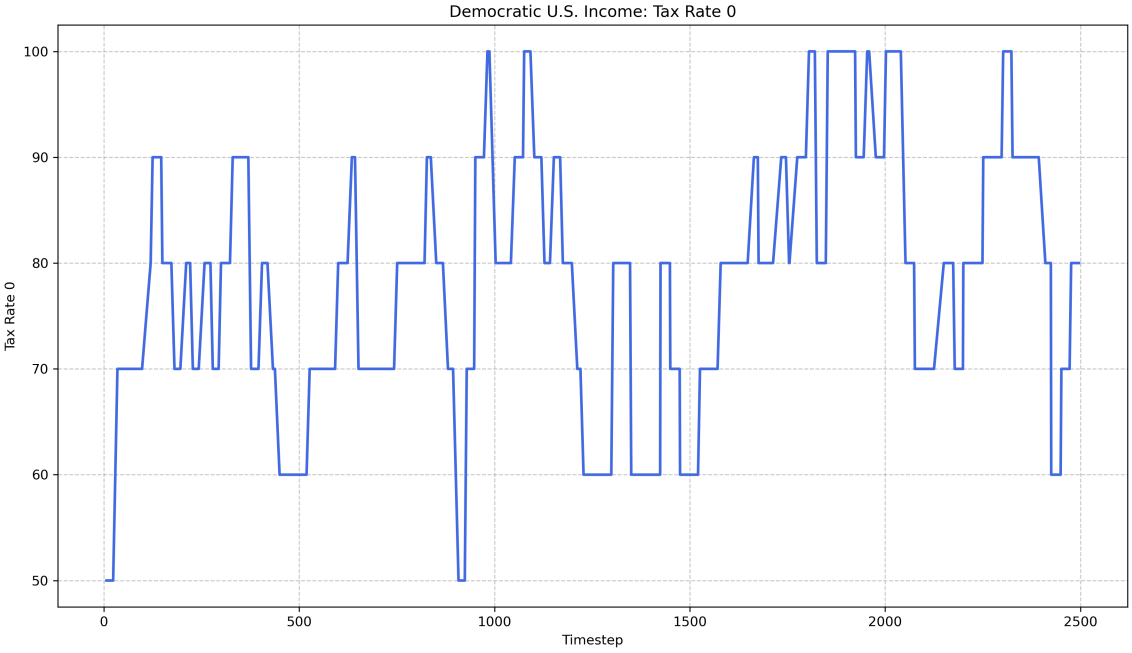


Figure C.22: Tax Rate for First Bracket

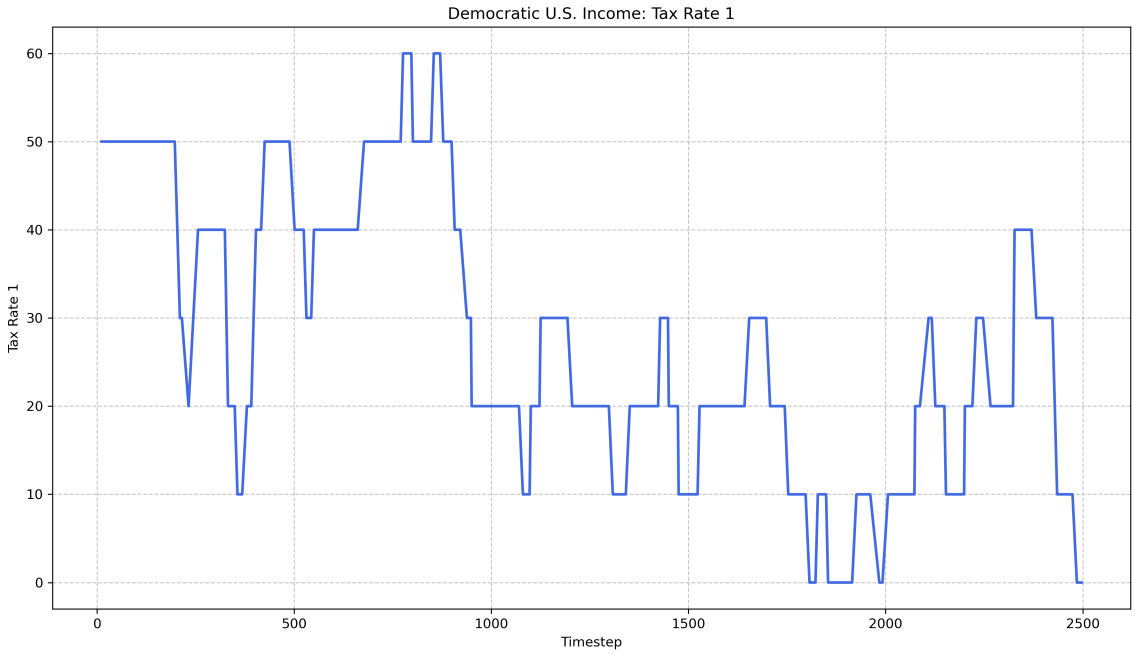


Figure C.23: Tax Rate for Second Bracket

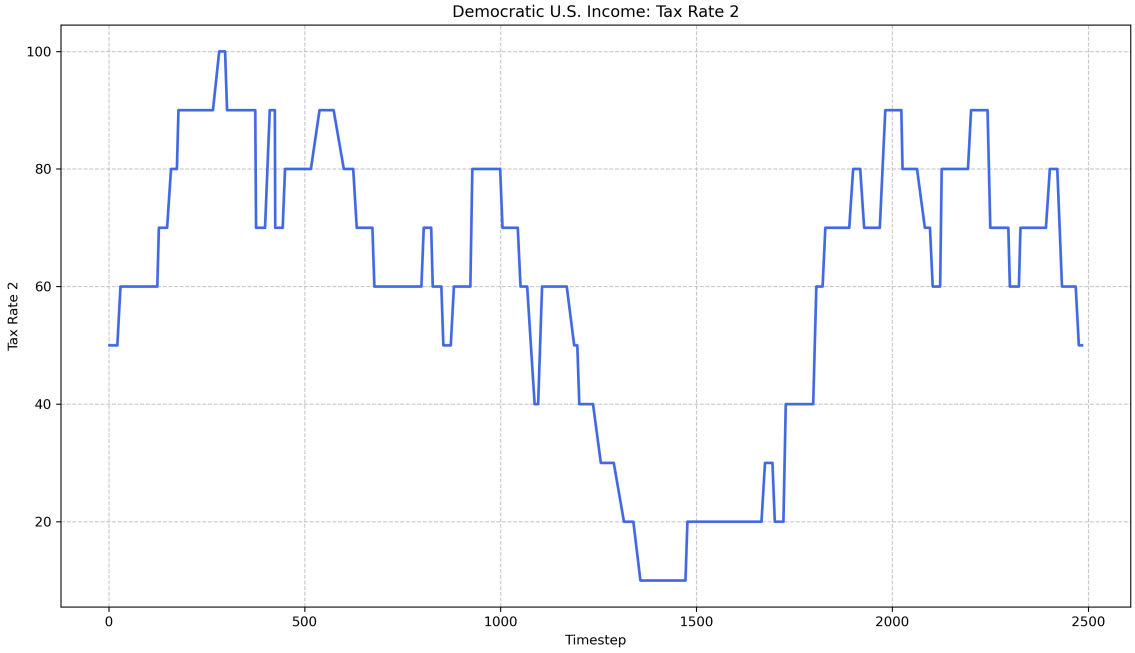


Figure C.24: Tax Rate for Top Bracket

C.1.4 Democratic Scenario with Platforms Feature

Uniform Distribution

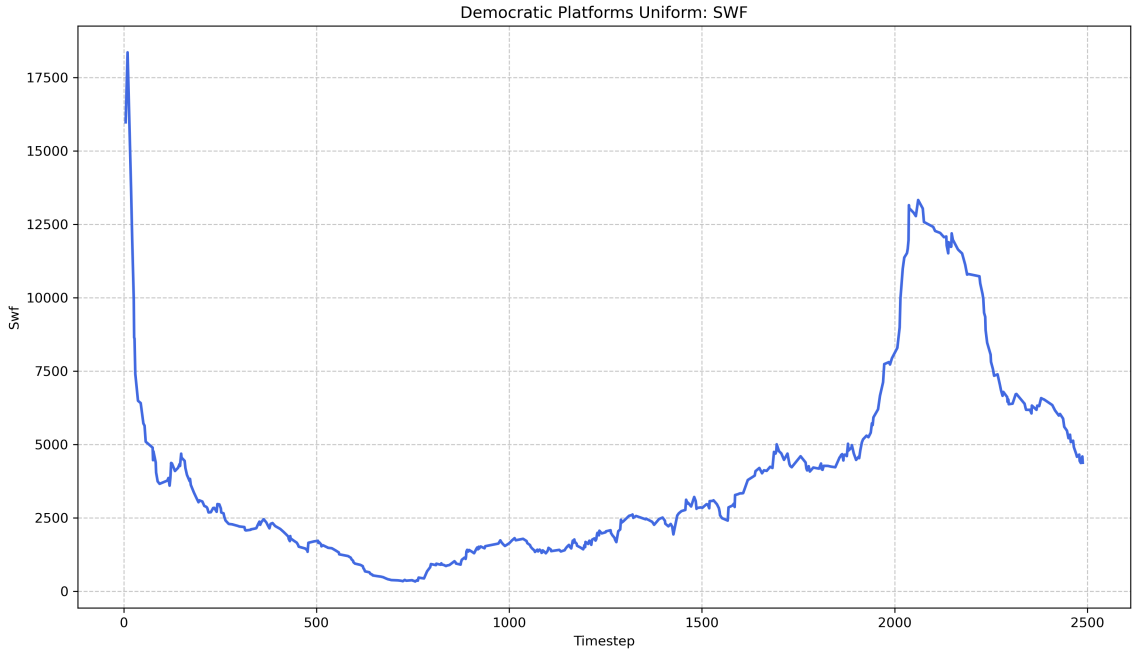


Figure C.25: Social Welfare Results

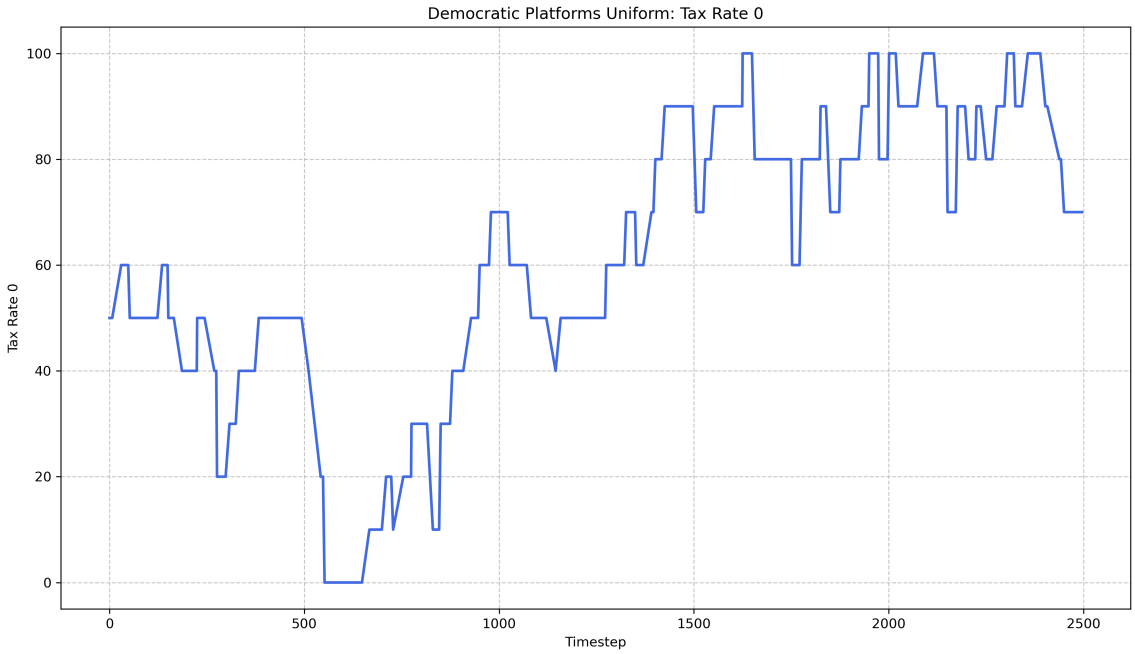


Figure C.26: Tax Rate for First Bracket

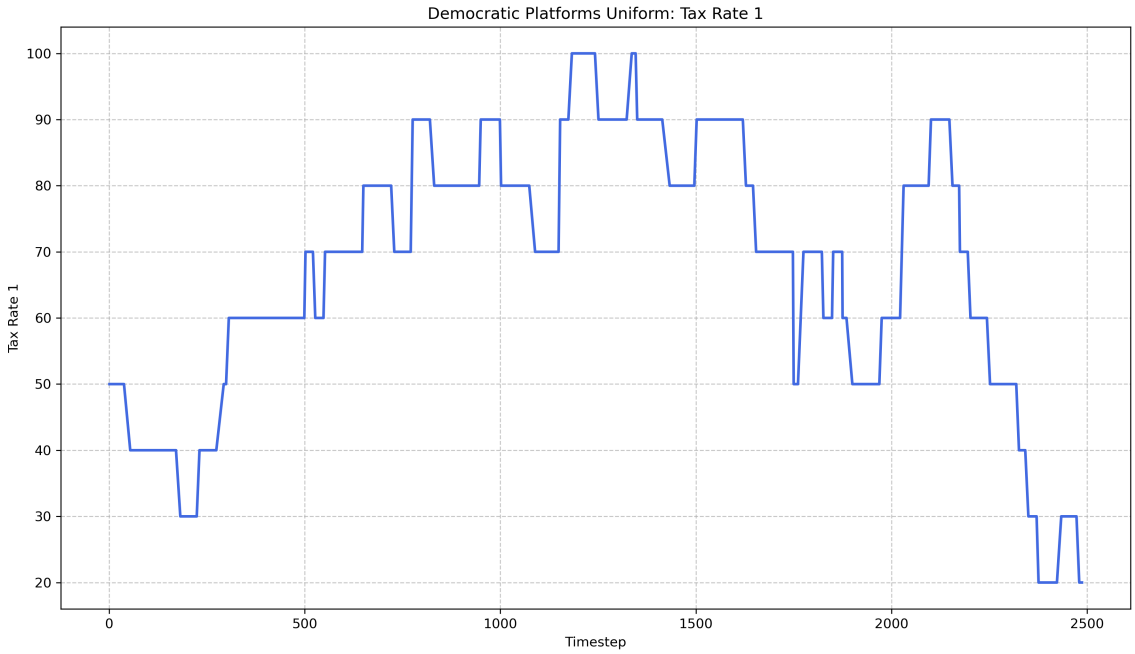


Figure C.27: Tax Rate for Second Bracket

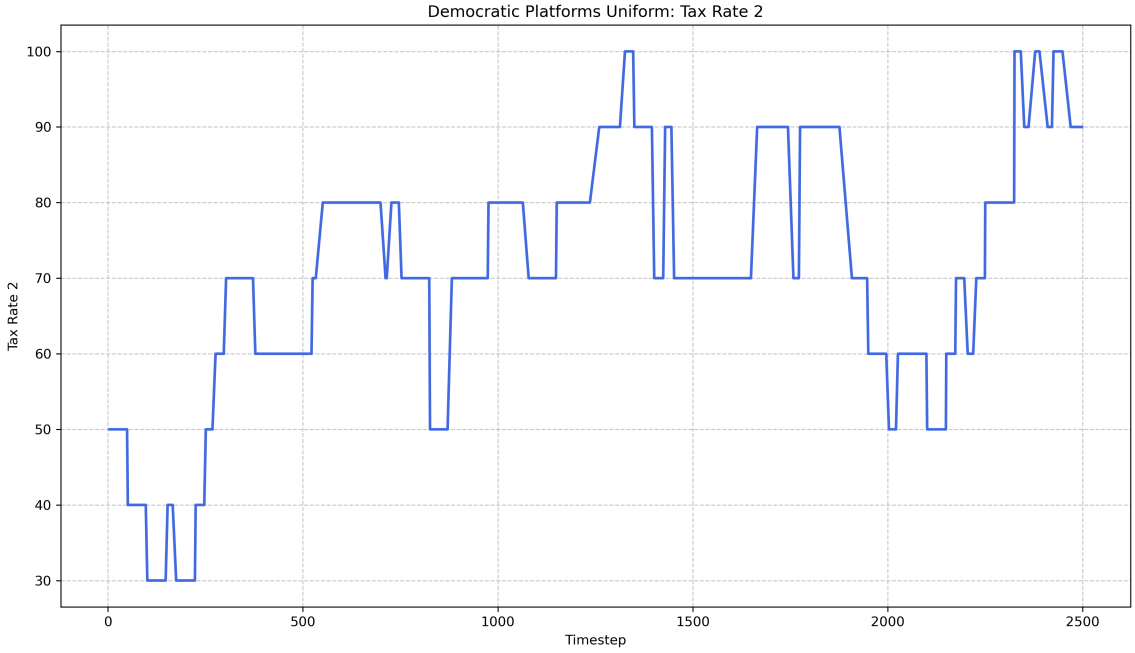


Figure C.28: Tax Rate for Top Bracket

U.S. Income Distribution

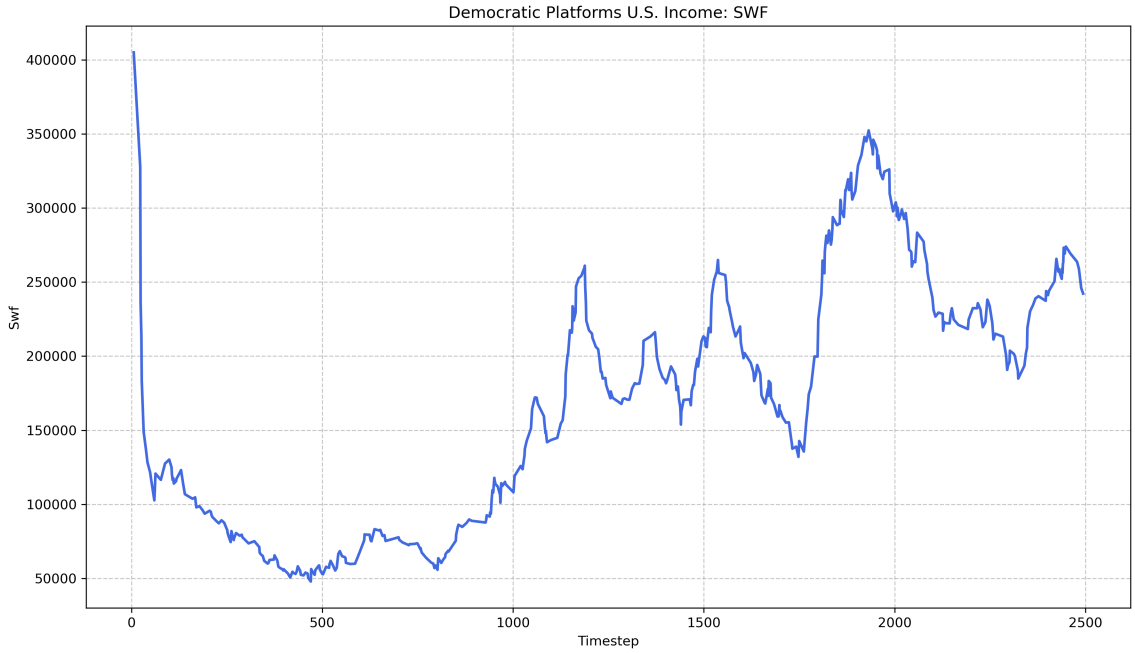


Figure C.29: Social Welfare Results

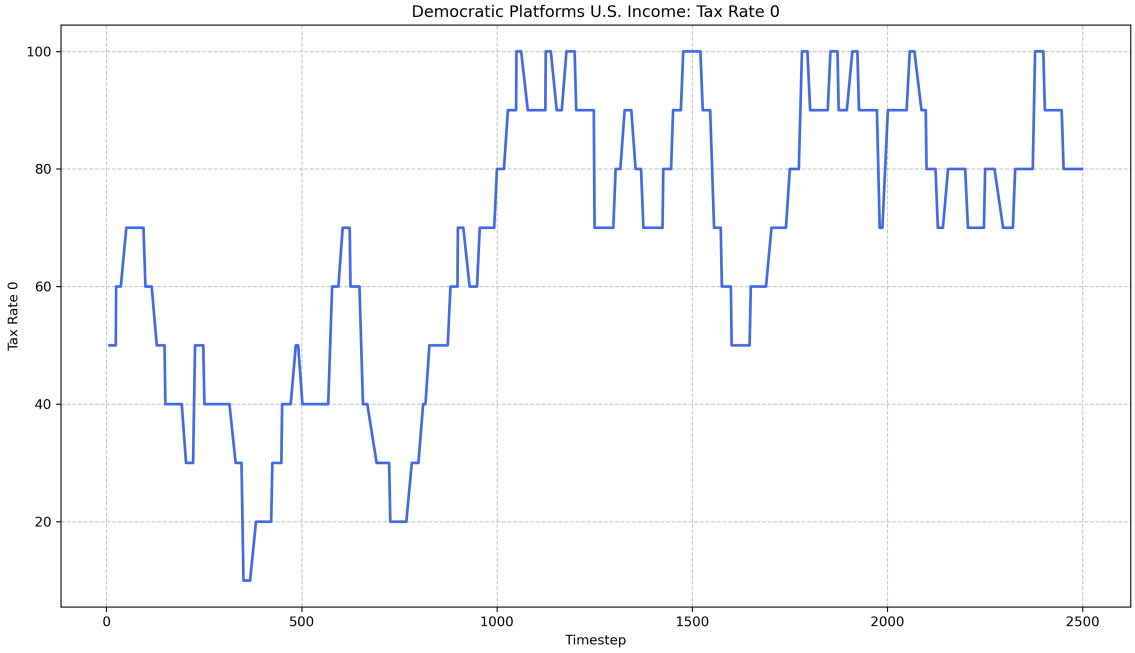


Figure C.30: Tax Rate for First Bracket

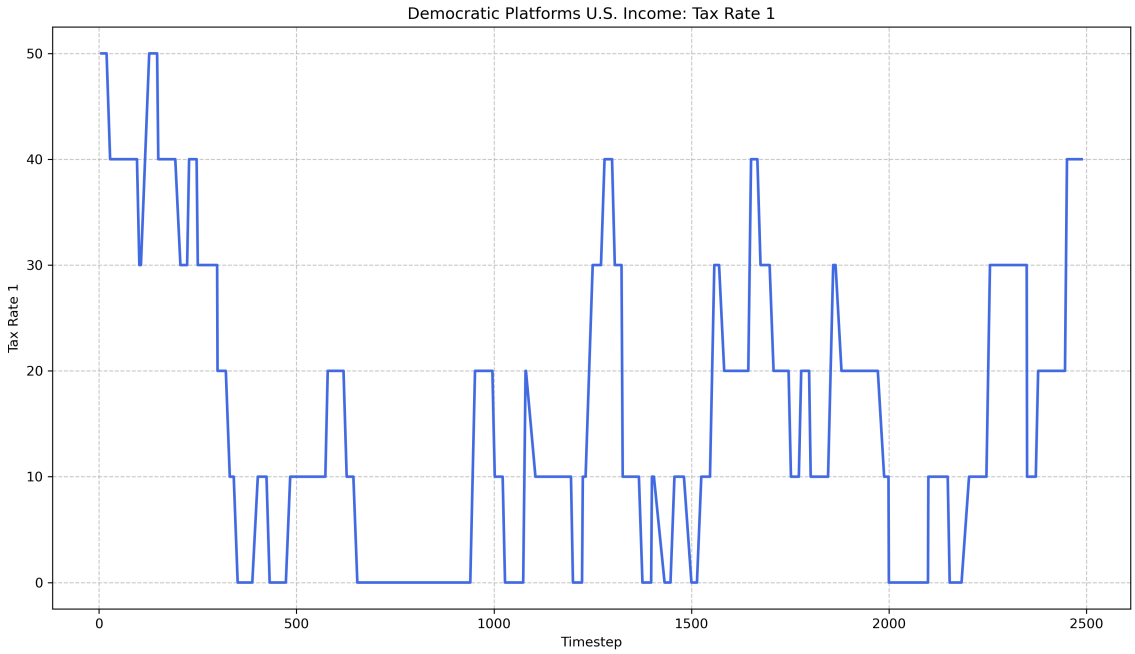


Figure C.31: Tax Rate for Second Bracket

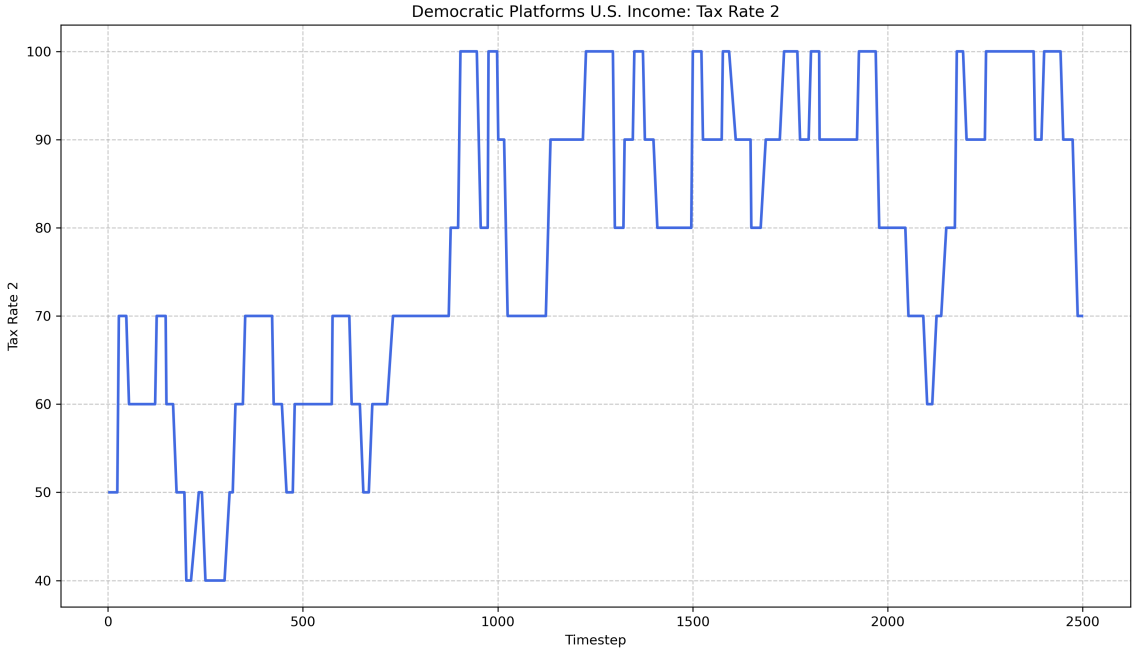


Figure C.32: Tax Rate for Top Bracket

Appendix D

Derivation of Optimal Income Tax for Utilitarian Social Welfare Using Simple Model without Behavior Response

$$\max_{T(z)} \int_0^\infty u(c)h(z)dz = \int_0^\infty u(z - T(z))h(z)dz \tag{D.1}$$

$$\text{subject to the constraint} \int_0^\infty T(z)h(z)dz \geq E \tag{D.2}$$

To solve this constrained optimization problem, we use the Lagrangian method. The Lagrangian combines the objective function and the constraint using a Lagrange multiplier λ:

$$\mathcal{L} = \text{Objective} + \lambda \cdot (\text{LHS of constraint} - \text{RHS of constraint})$$

Rewriting the constraint as $\int_0^\infty T(z)h(z)dz - E \geq 0$, the Lagrangian becomes:

$$\mathcal{L} = \int_0^\infty u(z - T(z))h(z)dz + \lambda \left(\int_0^\infty T(z)h(z)dz - E \right)$$

Combining the integrals:

$$\mathcal{L} = \int_0^\infty [u(z - T(z)) + \lambda \cdot T(z)]h(z)dz - \lambda E$$

Since λE is a constant term that doesn't affect the maximization of the objective with respect to $T(z)$, it is dropped from the Lagrangian:

$$\mathcal{L} = \int_0^\infty [u(z - T(z)) + \lambda \cdot T(z)]h(z)dz$$

The integrand can be used as $L(z)$, allowing the maximization of the objective with respect to every z:

$$\mathcal{L}(z) = [u(z - T(z)) + \lambda \cdot T(z)] \cdot h(z)$$

To find the optimal tax function, we differentiate the Lagrangian with respect to $T(z)$ and set it equal to zero:

$$\frac{\partial \mathcal{L}}{\partial T(z)} = \frac{\partial}{\partial T(z)}[u(z - T(z)) + \lambda \cdot T(z)] \cdot h(z) = 0$$

We evaluate each term separately:

1. For the utility term, we apply the chain rule since $u(z - T(z))$ is a composition of functions:

$$\frac{\partial u(z - T(z))}{\partial T(z)} = u'(z - T(z)) \cdot \frac{\partial (z - T(z))}{\partial T(z)}$$

Since z is a constant with respect to $T(z)$:

$$\frac{\partial (z - T(z))}{\partial T(z)} = 0 - 1 = -1$$

Therefore:

$$\frac{\partial u(z - T(z))}{\partial T(z)} = u'(z - T(z)) \cdot (-1) = -u'(z - T(z))$$

2. For the second term:

$$\frac{\partial(\lambda \cdot T(z))}{\partial T(z)} = \lambda$$

Combining these results, the first-order condition becomes:

$$[-u'(z - T(z)) + \lambda] \cdot h(z) = 0$$

Since $h(z) > 0$ (as it is a probability density function), we can divide by $h(z)$ to get:

$$-u'(z - T(z)) + \lambda = 0$$

Rearranging to solve for the utility yields:

$$u'(z - T(z)) = \lambda$$

Appendix E

Extensions to Saez's Optimal Income Taxation Formulas

E.1 Extensions

Mirrlees and Saez's work have been extended to incorporate more elements of human economic activity. These extensions are summarized by [41]

E.1.1 Migration Effects

When agents can migrate between jurisdictions:

- Migration responds to average - not marginal - tax rates
- For a linear tax with migration elasticity $\bar{e}_A$, the optimal top tax rate becomes:

$$\tau = \frac{1}{1 + a \cdot e + \bar{e}_A} \tag{E.1}$$

E.1.2 Coordinated Tax Policy with Migration

For coordinated tax policy across regions A and B with different taxes τ_A and τ_B:

$$\tau_A = \frac{1 - g_A - \tau_B e_A^B \cdot y_B / y_A}{1 - g_A + e_A} \tag{E.2}$$

Where e_A^B is the cross-elasticity of migration.

E.1.3 Tax Avoidance Responses

If a fraction s of the response to taxation is due to avoidance that shifts income to a population taxed at rate t:

$$\tau = \frac{1 + a \cdot t \cdot s \cdot e}{1 + a \cdot e} \tag{E.3}$$

E.1.4 Rent-Seeking Effects

When top earners receive rents rather than produce goods:

$$\tau^* = \frac{1 + a \cdot e_b}{1 + a \cdot e} = 1 - \frac{a(y/z)e_y}{1 + a \cdot e} \tag{E.4}$$

Where e_b is the bargaining elasticity component and e_y is the real labor supply elasticity.

Appendix F

Code

All of our simulation code will be released for NeurIPS 2025.

Code: github.com/sethkarten/LLM-Economist

F.1 stackelberg_calc.py

For reference, here is our code that calculates the Stackelberg equilibria for a given set of agents' skills through a grid-search implementation of backwards induction.

```
from worker import FixedWorker
from planner import FixedTaxPlanner
import argparse
import numpy as np
import sys
from itertools import product
from tqdm import tqdm


def main_loop(args):



    # init tax planner
```

```
13      tax_planner = FixedTaxPlanner('Joe', 'SAETZ_TWO',
            history_len=args.history_len, args=args)
14      # init N workers
15      num_agents = args.num_agents
16      num_brackets = 2
17      agents = [] # list of worker agents
18      # skills = [87.876,114.14,96.392]
19      skills = [25,50,75]
20      # skills =
            [87.87569972996748,114.13979337555013,96.39222654666969,87.2552926

21      for i in range(num_agents):
22          name = "worker_" + str(i)
23          agent = FixedWorker(name, history_len=args.
                two_timescale//2, skill=skills[i], args=args)
24          agents.append(agent)
25
26      tuple_rates_labor = (11,) * (num_brackets+num_agents)
27      tuple_rates_labor_agents = (11,) * (num_brackets+
            num_agents)
28      tuple_rates_labor_agents += (num_agents,)
29      utility = np.zeros(tuple_rates_labor_agents)
30      swf_arr = np.zeros(tuple_rates_labor)
31      tuple_rates_labor_rates = (11,) * (num_brackets+num_agents
            )
32      tuple_rates_labor_rates += (num_brackets,)
33      tax_arr = np.zeros(tuple_rates_labor_rates)
34      tax_rates = None
35      for i in range(num_agents):
```

```
        print(f'agent {i} skill: {agents[i].v}')
    # for worker_labor_arr, _ in np.ndenumerate([11]*
       num_agents*2):
    for tax_rate_arr in tqdm(list(product(range(11), repeat=
       num_brackets))):
        for k, worker_labor_arr in enumerate(list(product(
           range(11), repeat=num_agents))):
            if args.debug: print("TIMESTEP", k)
            logger = {}
            # get new tax rates
            tax_rates = np.array(tax_rate_arr) * 10
            tax_rates = tax_rates.tolist()
            tax_planner.tax_rates = tax_rates

            # calculate labor based on taxes
            for i in range(num_agents):
                agents[i].l = worker_labor_arr[i] * 10
                agents[i].z = agents[i].l * agents[i].v

            # calculate taxes
            pre_tax_incomes = [agents[i].z for i in range(
               num_agents)]
            post_tax_incomes, total_tax = tax_planner.
               apply_taxes(tax_rates, pre_tax_incomes)
            tax_indv = np.array(pre_tax_incomes) - np.array(
               post_tax_incomes)

            # calculate agent utility
            for i in range(num_agents):
```

```
59                  agents[i].tax = tax_indv[i]
60                  agents[i].update_utility(k, post_tax_incomes[i
                        ], total_tax / num_agents)
61                  agents[i].log_stats(k, logger, debug=args.
                        debug)
62              u = [agents[i].utility for i in range(num_agents)]
63              for i in range(num_agents):
64                  utility[tuple(tax_rate_arr)][tuple(
                        worker_labor_arr)][i] = agents[i].utility
65              for i in range(len(tax_rates)):
66                  tax_arr[tuple(tax_rate_arr)][tuple(
                        worker_labor_arr)][i] = tax_rates[i]
67              tax_planner.log_stats(k, logger, z=pre_tax_incomes
                    , u=u, debug=args.debug)
68              swf_arr[tuple(tax_rate_arr)][tuple(
                    worker_labor_arr)] = tax_planner.swf
69
70      # find highest utility for each tax bracket for each agent
71      tuple_utility_best = (11,) * (num_brackets)
72      tuple_utility_best += (-1,num_agents)
73      utility_arg_best = utility.reshape((tuple_utility_best)).
            argmax(axis=-2)
74      if args.debug: print('\n\n\n\nUTILITY')
75      if args.debug: print(utility_arg_best)
76      if args.debug: print(utility_arg_best.shape)
77      # recover
78      coords = []
79      for i in range(num_agents):
80          if i == num_agents - 1:
```

```
81              arr = utility_arg_best % 11
82          else:
83              arr = (utility_arg_best // int(11**(num_agents-1 -
                    i))) % 11
84          arr = arr[...,i]    # select coords for specific agent
                corresponding to their labor choice
85          coords.append(arr)
86      coords = np.array(coords)
87      if args.debug: print(coords.shape)
88      coords_tuple = ()
89      for i in range(1,len(coords.shape)):
90          coords_tuple += (i,)
91      coords_tuple += (0,)
92      if args.debug: print(coords_tuple)
93      coords  = np.transpose(coords, coords_tuple)
94
95      # get optimal swf
96      tuple_swf_best = (11,) * (num_brackets)
97      swf_opt = np.zeros(tuple_swf_best)
98      tuple_swf_best_agents = tuple_swf_best
99      tuple_swf_best_agents += (num_agents,)
100     labor_opt = np.zeros(tuple_swf_best_agents)
101     rate_opt = np.zeros(tuple_swf_best)
102     for tax_rate_arr in list(product(range(11), repeat=
           num_brackets)):
103         swf_opt[tuple(tax_rate_arr)] = swf_arr[tuple(
               tax_rate_arr)][tuple(coords[tuple(tax_rate_arr)])]
104         labor_opt[tuple(tax_rate_arr)] = coords[tuple(
               tax_rate_arr)]
```

```
105     if args.debug: print('\n\n\n\n')
106     if args.debug: print(labor_opt)
107     swf_max_arg = np.unravel_index(np.argmax(swf_opt), swf_opt
            .shape)
108     if args.debug: print(swf_max_arg)
109     swf_best = swf_opt[swf_max_arg]
110     print('s', ((swf_opt / swf_best ) * 100).tolist())
111     print('s', (swf_opt).tolist())
112     # get best rates
113     rates_best = np.array(swf_max_arg) * 10
114     # get optimal labor
115     labor_best = labor_opt[swf_max_arg] * 10
116     # np.set_printoptions(threshold=sys.maxsize)
117 
118     if args.debug: print(labor_best)
119     if args.debug: print(swf_best)
120 
121     tax_rates = np.array(rates_best)
122     tax_rates = tax_rates.tolist()
123     tax_planner.tax_rates = tax_rates
124 
125     # calculate labor based on taxes
126     for i in range(num_agents):
127         agents[i].l = int(labor_best[i])
128         agents[i].z = agents[i].l * agents[i].v
129 
130     # calculate taxes
131     pre_tax_incomes = [agents[i].z for i in range(num_agents)]
132     post_tax_incomes, total_tax = tax_planner.apply_taxes(
```

```
        tax_rates, pre_tax_incomes)
133     tax_indv = np.array(pre_tax_incomes) - np.array(
            post_tax_incomes)
134
135     # calculate agent utility
136     for i in range(num_agents):
137         agents[i].tax = tax_indv[i]
138         agents[i].update_utility(0, post_tax_incomes[i],
                total_tax / num_agents)
139     u = [agents[i].utility for i in range(num_agents)]
140     print(f'done:\nswf: {swf_best}\ntax rates: {rates_best}\
            nlabor: {labor_best}\nutility: {u}')
141     return np.array(rates_best), np.array(swf_best), np.array(
            labor_best), np.array(u)
142
143 def analyze_rates(args, worker_labor_arr):
144     assert args.num_agents == len(worker_labor_arr)
145
146     # init tax planner
147     tax_planner = FixedTaxPlanner('Joe', 'SAETZ_TWO',
            history_len=args.history_len, args=args)
148     # init N workers
149     num_agents = args.num_agents
150     num_brackets = 2
151     agents = [] # list of worker agents
152     skills = [25,50,75]
153     # skills = [87.876,114.14,96.392]
154     for i in range(num_agents):
155         name = "worker_" + str(i)
```

```
156         agent = FixedWorker(name, history_len=args.
                two_timescale//2, skill=skills[i], args=args)
157         agents.append(agent)
158 
159     tuple_rates_labor = (11,) * (num_brackets+num_agents)
160     tuple_rates_labor_agents = (11,) * (num_brackets+
            num_agents)
161     tuple_rates_labor_agents += (num_agents,)
162     utility = np.zeros(tuple_rates_labor_agents)
163     swf_arr = np.zeros(tuple_rates_labor)
164     tuple_rates_labor_rates = (11,) * (num_brackets+num_agents
            )
165     tuple_rates_labor_rates += (num_brackets,)
166     tax_arr = np.zeros(tuple_rates_labor_rates)
167     tax_rates = None
168     # for worker_labor_arr, _ in np.ndenumerate([11]*
            num_agents*2):
169     k = 0
170     for tax_rate_arr in list(product(range(11), repeat=
            num_brackets)):
171         logger = {}
172         # get new tax rates
173         tax_rates = np.array(tax_rate_arr) * 10
174         tax_rates = tax_rates.tolist()
175         tax_planner.tax_rates = tax_rates
176 
177         # calculate labor based on taxes
178         for i in range(num_agents):
179             agents[i].l = np.array(worker_labor_arr[i]) * 10
```

```
180             agents[i].z = agents[i].l * agents[i].v
181
182         # calculate taxes
183         pre_tax_incomes = [agents[i].z for i in range(
                num_agents)]
184         post_tax_incomes, total_tax = tax_planner.apply_taxes(
                tax_rates, pre_tax_incomes)
185         tax_indv = np.array(pre_tax_incomes) - np.array(
                post_tax_incomes)
186
187         # calculate agent utility
188         for i in range(num_agents):
189             agents[i].tax = tax_indv[i]
190             agents[i].update_utility(k, post_tax_incomes[i],
                    total_tax / num_agents)
191             agents[i].log_stats(k, logger, debug=args.debug)
192         u = [agents[i].utility for i in range(num_agents)]
193         for i in range(num_agents):
194             utility[tuple(tax_rate_arr)][tuple(
                    worker_labor_arr)][i] = agents[i].utility
195         for i in range(len(tax_rates)):
196             tax_arr[tuple(tax_rate_arr)][tuple(
                    worker_labor_arr)][i] = tax_rates[i]
197         tax_planner.log_stats(k, logger, z=pre_tax_incomes, u=
                u, debug=args.debug)
198         swf_arr[tuple(tax_rate_arr)][tuple(worker_labor_arr)]
                = tax_planner.swf
199
200     # find highest utility for each tax bracket for each agent
```

```
    tuple_utility_best = (11,) * (num_brackets)
    tuple_utility_best += (-1,num_agents)
    utility_arg_best = utility.reshape((tuple_utility_best)).
       argmax(axis=-2)
    if args.debug: print('\n\n\n\nUTILITY')
    if args.debug: print(utility_arg_best)
    if args.debug: print(utility_arg_best.shape)
    # recover
    coords = []
    for i in range(num_agents):
        if i == num_agents - 1:
            arr = utility_arg_best % 11
        else:
            arr = (utility_arg_best // int(11**(num_agents-1 -
                i))) % 11
        arr = arr[...,i]    # select coords for specific agent
            corresponding to their labor choice
        coords.append(arr)
    coords = np.array(coords)
    if args.debug: print(coords.shape)
    coords_tuple = ()
    for i in range(1,len(coords.shape)):
        coords_tuple += (i,)
    coords_tuple += (0,)
    if args.debug: print(coords_tuple)
    coords  = np.transpose(coords, coords_tuple)

    # get optimal swf
    tuple_swf_best = (11,) * (num_brackets)
```

```
227     swf_opt = np.zeros(tuple_swf_best)
228     tuple_swf_best_agents = tuple_swf_best
229     tuple_swf_best_agents += (num_agents,)
230     labor_opt = np.zeros(tuple_swf_best_agents)
231     rate_opt = np.zeros(tuple_swf_best)
232     for tax_rate_arr in list(product(range(11), repeat=
           num_brackets)):
233         swf_opt[tuple(tax_rate_arr)] = swf_arr[tuple(
               tax_rate_arr)][tuple(coords[tuple(tax_rate_arr)])]
234         labor_opt[tuple(tax_rate_arr)] = coords[tuple(
               tax_rate_arr)]
235     if args.debug: print('\n\n\n\n')
236     if args.debug: print(labor_opt)
237     swf_max_arg = np.unravel_index(np.argmax(swf_opt), swf_opt
           .shape)
238     if args.debug: print(swf_max_arg)
239     swf_best = swf_opt[swf_max_arg]
240     # print('s', ((swf_opt / swf_best ) * 100).tolist())
241     # get best rates
242     rates_best = np.array(swf_max_arg) * 10
243     # get optimal labor
244     labor_best = labor_opt[swf_max_arg] * 10
245     # np.set_printoptions(threshold=sys.maxsize)
246
247     if args.debug: print(labor_best)
248     if args.debug: print(swf_best)
249
250     tax_rates = np.array(rates_best) / 10
251     tax_rates = tax_rates.tolist()
```

```
252     tax_planner.tax_rates = tax_rates
253
254     # calculate labor based on taxes
255     for i in range(num_agents):
256         agents[i].l = np.array(labor_best[i])
257         agents[i].z = agents[i].l * agents[i].v
258
259     # calculate taxes
260     pre_tax_incomes = [agents[i].z for i in range(num_agents)]
261     post_tax_incomes, total_tax = tax_planner.apply_taxes(
            tax_rates, pre_tax_incomes)
262     tax_indv = np.array(pre_tax_incomes) - np.array(
            post_tax_incomes)
263
264     # calculate agent utility
265     for i in range(num_agents):
266         agents[i].tax = tax_indv[i]
267         agents[i].update_utility(k, post_tax_incomes[i],
                total_tax / num_agents)
268     u = [agents[i].utility for i in range(num_agents)]
269     # print(f'done {worker_labor_arr}:\nswf: {swf_best}\ntax
            rates: {rates_best}\nlabor: {labor_best}\nutility: {u
            }')
270     return np.array(rates_best), np.array(swf_best), np.array(
            labor_best), np.array(u)
271
272 def output_best(rates_best_equilibria, swf_max,
        labor_best_equilibria, utility_max, rates, args):
273     rates_best_fixed_l, swf_max_fixed_l, labor,
```

```
        utility_fixed_l = analyze_rates(args, rates)
    print(f"labor {labor}\nrates: {rates_best_fixed_l}\n% of
        max:\nswf: {np.around(swf_max_fixed_l/ swf_max * 100,
        2)}\nutility: {np.around(utility_fixed_l/utility_max
        *100, 2)}")
    print()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Simulation
        stats')
    parser.add_argument('--num-agents', type=int, default=2)
    parser.add_argument('--planner-type', default='LLM',
        choices=['LLM', 'US_FED', 'SAETZ', 'SAETZ_TWO'])
    parser.add_argument('--max-timesteps', type=int, default
        =100)
    parser.add_argument('--history-len', type=int, default=20)
    parser.add_argument('--two-timescale', type=int, default
        =20)
    parser.add_argument('--debug', type=bool, default=False)
    parser.add_argument('--bracket_setting', default='two',
        choices=['two', 'US_FED'])
    args = parser.parse_args()
    print(args)
    np.random.seed(0)
    rates_best_equilibria, swf_max, labor_best_equilibria,
        utility_max = main_loop(args)
    # analyze_rates(args, [])
    output_best(rates_best_equilibria, swf_max,
        labor_best_equilibria, utility_max, [4, 5, 6], args)
```

```
292     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [5,7,6], args)
293     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [7,7,6], args)
294
295     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,7,6], args)
296     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,6,6], args)
297     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,8,6], args)
298
299     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,7,6], args)
300     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,7,7], args)
301     # output_best(rates_best_equilibria, swf_max,
            labor_best_equilibria, utility_max, [6,7,5], args)
```

Code/saetz–calc.py

F.2 saez.py

For reference, here is our function that calculates a tax policy from Saez's optimal income taxation formulas.

```
1 def saez_optimal_tax_rates(skills, brackets, elasticities):
2     """
3     Calculate Saez optimal marginal tax rates for income
          brackets based on skills.
```

```
4
5       Parameters:
6       -----------
7       skills : list of float
8           List of individual skills (incomes/100).
9       brackets : list of float
10          List of income-cutoff points [min1, min2, ...,
                max_value];
11          each consecutive pair defines one bracket.
12      elasticities : float or list of float
13          If a single float: apply this elasticity to every
                bracket.
14          If a list: must have length = (number of brackets), i.
                e. len(brackets)-1,
15          giving one elasticity per bracket.
16
17      Returns:
18      --------
19      tax_rates : list of float
20          Optimal marginal tax rates for each bracket, in
                percentages
21          (e.g., [12.88, 3.23, 3.23]).
22      """
23      # Convert skills to incomes
24      incomes = np.array(skills) * 100.0
25      brackets = np.array(brackets)
26
27      # Build elasticity list
28      n_brackets = len(brackets) - 1
```

```
    if isinstance(elasticities, (int, float)):
        elasticities = [float(elasticities)] * n_brackets
    else:
        if len(elasticities) != n_brackets:
            raise ValueError(f"elasticities must be length {
                n_brackets}, got {len(elasticities)}")
        elasticities = [float(e) for e in elasticities]

    # Sort incomes and compute welfare weights
    incomes = np.sort(incomes)
    welfare_weights = 1.0 / np.maximum(incomes, 1e-10)
    welfare_weights /= welfare_weights.sum()

    # Estimate density
    kde = stats.gaussian_kde(incomes)

    tax_rates = []
    for i in range(n_brackets):
        bracket_start, bracket_end = brackets[i], brackets[i
            +1]
        # choose z at midpoint (or near start for top bracket)
        if i < n_brackets - 1:
            z = 0.5 * (bracket_start + bracket_end)
        else:
            z = bracket_start + 0.1 * (bracket_end -
                bracket_start)

        F_z = np.mean(incomes <= z)
        f_z = kde(z)[0]
```

```
55
56          # Pareto-tail parameter a(z)
57          if F_z < 1.0:
58              a_z = (z * f_z) / (1.0 - F_z)
59          else:
60              a_z = 10.0
61
62          # for the top bracket refine a(z)
63          incomes_above = incomes[incomes >= z]
64          if i == n_brackets - 1 and incomes_above.size > 0:
65              m = incomes_above.mean()
66              a_z = m / (m - bracket_start)
67
68          # G(z): average welfare weight above z, normalized
69          if incomes_above.size > 0 and F_z < 1.0:
70              G_z = welfare_weights[incomes >= z].sum() / (1.0 -
                    F_z)
71          else:
72              G_z = 0.0
73
74          # pick the right elasticity for this bracket
75          ε = elasticities[i]
76
77          # Saez optimal rate τ = (1 - G) / [1 - G + a * ε]
78          tau = (1.0 - G_z) / (1.0 - G_z + a_z * ε)
79          tau = max(0.0, min(1.0, tau))
80
81          tax_rates.append(round(tau * 100, 2))
82
```

```
83      return tax_rates
```

Code/saez.py

Bibliography

[1] K. Armstrong. The WEIRD Science of Culture, Values, and Behavior. *APS Observer*, 31, Mar. 2018.

[2] A. Atkinson and J. Stiglitz. The design of tax structure: Direct versus indirect taxation. *Journal of Public Economics*, 6(1-2):55–75, July 1976.

[3] AWS. What is Batch Processing? - Batch Processing Systems Explained - AWS. https://aws.amazon.com/what-is/batch-processing/?utm_source=chatgpt.com.

[4] X. Bai, A. Wang, I. Sucholutsky, and T. L. Griffiths. Measuring Implicit Bias in Explicitly Unbiased Large Language Models, May 2024.

[5] Tamer. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Academic Press, London ;, 2nd ed. edition, 1995.

[6] R. Bevans. Akaike Information Criterion | When & How to Use It (Example), Mar. 2020.

[7] G. Brero, A. Eden, D. Chakrabarti, M. Gerstgrasser, A. Greenwald, V. Li, and D. C. Parkes. Stackelberg POMDP: A Reinforcement Learning Approach for Economic Design, July 2024. arXiv:2210.03852 [cs].

[8] G. Brero, N. Lepore, E. Mibuari, and D. C. Parkes. Learning to Mitigate AI Collusion on Economic Platforms, June 2022. arXiv:2202.07106 [cs].

[9] D. Bunn and C. Weigel. Sources of U.S. Tax Revenue by Tax Type, 2024, Mar. 2024.

[10] S. Clarke, G. Dragotto, J. F. Fisac, and B. Stellato. Learning Rationality in Potential Games. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 4261–4266, Dec. 2023.

[11] P. A. Diamond and E. Saez. The Case for a Progressive Tax: From Basic Research to Policy Recommendations. *SSRN Electronic Journal*, 2011.

[12] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, B. Chang, X. Sun, L. Li, and Z. Sui. A Survey on In-context Learning, Oct. 2024.

[13] P. Duetting, V. Mirrokni, R. P. Leme, H. Xu, and S. Zuo. Mechanism Design for Large Language Models, July 2024. arXiv:2310.10826 [cs].

[14] C. Ford. Understanding QQ Plots | UVA Library, Aug. 2015.

[15] J. Gao, H. Xu, and L. Dao. Multi-Generative Agent Collective Decision-Making in Urban Planning: A Case Study for Kendall Square Renovation, Feb. 2024.

[16] A. Goli and A. Singh. Frontiers: Can Large Language Models Capture Human Preferences? *Marketing Science*, 43(4):709–722, July 2024.

[17] J. Gruber and E. Saez. The Elasticity of Taxable Income: Evidence and Implications. *NBER WORKING PAPER SERIES*, Jan. 2000.

[18] D. Henderson. James A. Mirrlees - Econlib.

[19] IRS. Earned Income Tax Credit (EITC) | Internal Revenue Service, 2024.

[20] M. Jain, F. Ordóñez, J. Pita, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Robust Solutions in Stackelberg Games: Addressing Boundedly Rational Human Preference Models. *Association for the Advancement of Artificial Intelligence*, 2008.

[21] R. Johari. Stackelberg Games, 2007.

[22] S. Karten, W. Li, Z. Ding, Y. Bai, and C. Jin. The LLM Economist: Optimizing Policy in Multiagent Generative Simulations.

[23] LADWP. Commercial Electric Rates | Los Angeles Department of Water and Power. https://www.ladwp.com/account/understanding-your-rates/commercial-electric-rates.

[24] S. Lee. Kolmogorov-Smirnov Test Explained: 6 Essential Facts for Data Experts, Mar. 2025.

[25] J. Lidard, H. Hu, A. Hancock, Z. Zhang, A. Contreras, V. Modi, J. DeCastro, D. Gopinath, G. Rosman, N. Leonard, M. Santos, and J. Fisac. Blending Data-Driven Priors in Dynamic Games. In *Robotics: Science and Systems XX*. Robotics: Science and Systems Foundation, July 2024.

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments, Mar. 2020.

[27] R. E. Lucas. ECONOMETRIC POEICY EVALUATION: A CRITIQUE. *Carnegie-Rochester Conference Series on Public Policy*, 1:19–46, 1976.

[28] A. Marchesi. Leadership Games: Multiple Followers, Multiple Leaders, and Perfection. In A. Geraci, editor, *Special Topics in Information Technology*, pages 107–118. Springer International Publishing, Cham, 2021.

[29] J. McCarty and L. J. Shrum. The Role of Personal Values and Demographics in Predicting Television Viewing Behavior: Implications for Theory and Application on JSTOR. *Journal of Advertising*, 22, Dec. 1993.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space, Sept. 2013.

[31] J. A. Mirrlees. An Exploration in the Theory of Optimum Income Taxation. *The Review of Economic Studies*, 38(2):175–208, Apr. 1971.

[32] G. Monea, A. Bosselut, K. Brantley, and Y. Artzi. LLMs Are In-Context Reinforcement Learners, Oct. 2024.

[33] T. Nakamura. One-leader and multiple-follower Stackelberg games with private information. *Economics Letters*, 127:27–30, Feb. 2015.

[34] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, San Francisco CA USA, Oct. 2023. ACM.

[35] T. Piketty, E. Saez, and S. Stantcheva. Optimal Taxation of Top Labor Incomes: A Tale of Three Elasticities. *NATIONAL BUREAU OF ECONOMIC RESEARCH*, Nov. 2011.

[36] A. Rees-Jones and D. Taubinsky. Taxing Humans: Pitfalls of the Mechanism Design Approach and Potential Resolutions. *Tax Policy and the Economy*, 33(1):107–133, 2018.

[37] A. Reynolds. Optimal Top Tax Rates: A Review and Critique. *Cato Journal*, 39(3), Sept. 2019.

[38] E. Saez. Using Elasticities to Derive Optimal Income Tax Rates. *The Review of Economic Studies*, 68, 2001.

[39] E. Saez and S. Stantcheva. A simpler theory of optimal capital taxation. *Journal of Public Economics*, 162:120–142, June 2018.

[40] J. Seade. On the shape of optimal tax schedules. *Journal of Public Economics*, 7(2):203–235, Apr. 1977.

[41] S. Stantcheva. Optimal Income Taxation, 2022.

[42] A. Storkey, J. Millin, and K. Geras. Isoelastic Agents and Wealth Updates in Machine Learning Markets, Sept. 2012.

[43] G. Suri, L. R. Slater, A. Ziaee, and M. Nguyen. Do Large Language Models Show Decision Heuristics Similar to Humans? A Case Study Using GPT-3.5, May 2023. arXiv:2305.04400 [cs].

[44] L. Tjuatja, V. Chen, T. Wu, A. Talwalkwar, and G. Neubig. Do LLMs Exhibit Human-like Response Biases? A Case Study in Survey Design. *Transactions of the Association for Computational Linguistics*, 12:1011–1026, Sept. 2024.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, Aug. 2023.

[46] H. Von Stackelberg. *Market Structure and Equilibrium*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[47] J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, and T. Ma. Larger language models do in-context learning differently, Mar. 2023.

[48] J. Wu, W. Shen, F. Fang, and H. Xu. Inverse Game Theory for Stackelberg Games: The Blessing of Bounded Rationality, Oct. 2022.

[49] M. Yildiz. 14.12 Game Theory Lecture Notes∗ Lectures 7-9.

[50] S. Yousefi, L. Betthauser, H. Hasanbeig, R. Millière, and I. Momennejad. Decoding In-Context Learning: Neuroscience-inspired Analysis of Representations in Large Language Models, Feb. 2024.

[51] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games, Nov. 2022.

[52] S. Zheng, A. Trott, S. Srinivasa, N. Naik, M. Gruesbeck, D. C. Parkes, and R. Socher. The AI Economist: Improving Equality and Productivity with AI-Driven Tax Policies, Apr. 2020.

[53] S. Zheng, A. Trott, S. Srinivasa, D. C. Parkes, and R. Socher. The AI Economist: Taxation policy design via two-level deep multiagent reinforcement learning. *Science Advances*, May 2022.