1 Purpose

We want to develop a model based on the paper “A Reassessment of the 1970s Property Tax Revolt: Was the Tax Revolt an Unintended Consequence of Good Government?” by Nathan Anderson and Andreas Pape. The purpose of this model is to use Agent-based software to evaluate the voting behavior and actions of rational agents implementing median voter theorem as described in the paper. In essence voters choose a property tax limit under uncertainty in future periods, we want to model this and investigate the collective actions of voters.

Show Theorem 1 (pg11):

1. If there is uncertainty, there is support for a binding limit.

2. If there is no uncertainty, there is no support for a binding limit.

The model will have two types of decision-makers:
1. DMs that follow the derived rule in Anderson/Pape (No learning).

2. DMs that are Case-based Software Agents (CBSAs) (Learning).

2 Project Scope

Teams are to program an Agent Based Model in NetLogo during the term of the class. The final product will be presented in the Complex Systems Seminar by the team and there will be a final composite document about the project, in the form that an academic paper follows.

3 Project Roles

These are suggested roles that people on the team can fill. The class as a team is meant to self organize and choose the roles that they feel are best suited to them and to the team.

- Subject Matter Expert and Facilitator (Professor Pape)
- Solution Architect - Designs and ensures that the different sections of code fit together and ultimately run. Organizes how the code ultimately works for the whole program.
- Coders - Provides the functional code, may be responsible for just sections of the code.
- Quality Assurance - in charge of the results section, describe what kind of results the software produces and how one can use the software to simulate social policy and evaluate it. Designs and sets up the experiments to run the model on.
- Methods Documentor - responsible for methods section in which we describe how we wrote this model and the technical specifications of what it can do. Also compiles documentation of the coders and other documentation needs.
- Literature Review - will be in charge of ultimately writing the literature review section, as well as selecting the best target journal.

4 Software Documentation

Notation in your code is essential since this is a joint project. And a small write-up describing the quality assurance tests you have performed to assure that the code operates as you claim.

5 Products of This Project

1. An Agent Based Software Program that replicates the results of the Anderson and Pape paper.

2. The same Agent Based Software Program that implements agents with Case-based Decision Making.
3. An article written for publication in an Complex Systems Journal of appropriate stature.
4. A presentation of the results of this project. (Potential)
6 Appendix: Description of Agents and Game

6.1 Agents
[excerpt from Anderson/Pape ]
Consider a continuum of agents \( i \in I = [0, 1] \) who are voting citizens in the same jurisdiction. There is a consumption good \( C \) and a local public good \( R \) equal to tax revenue. Suppose each agent has additively separable, quadratic utility:

\[
U(C_i, R) = u(C_i) + \lambda u(R), \quad \text{where } u(x) = \phi x - \frac{1}{2} x^2
\]  (1)

Prices:
Each agent \( i \) has an uncertain future tax price, a random variable \( p(i) \) with mean \( \mu(i) \) and forecast error \( \epsilon(i) \):

\[
p(i) = \mu(i) + \epsilon(i)
\]  (2)

This equation describes an agent’s expectation of the level of his or her future tax price, and the expected future tax price, \( \mu(i) \), could be higher, lower, or the same as the current tax price.

Tax payments, \( T_i = p(i) R \), under market value assessment.

Then the agent’s optimal, ex-post revenue is given by the following function \( R(p, \omega) \) when they have income \( \omega \) and their tax price is revealed to be \( p \).

\[
R(p, \omega) = \frac{\phi(\lambda - p) + p\omega}{\lambda + p^2}
\]  (3)

6.2 The Game
[excerpt from Anderson/Pape ]
There are two stages to the game: an ex-ante stage, before tax price forecast errors, \( \epsilon(i) \), are revealed; and an ex-post stage, after \( \epsilon(i) \) have been revealed. In the ex-post stage, agents vote for a level of revenues \( R_{\text{ex-post}}^* \). If no revenue limit is in effect, then \( R_{\text{ex-post}}^* \) wins the majority vote, is implemented, and enters all agents’ utility functions. If a revenue limit \( \mathcal{R} \) is in effect and \( R_{\text{ex-post}}^* > \mathcal{R} \), then \( \mathcal{R} \) is implemented and enters all agents’ utility functions. If a revenue limit \( \mathcal{R} \) is in effect but \( R_{\text{ex-post}}^* \leq \mathcal{R} \), then \( R_{\text{ex-post}}^* \) is implemented and enters all agents’ utility functions. At this point the game is concluded.

In the ex-ante stage, agents vote on whether to impose a revenue limit, where the alternative is to proceed to the ex-post stage with no limit in place. If the agents vote to impose a revenue limit, they immediately vote on the level of this revenue limit, \( \mathcal{R} \). They then proceed to the ex-post stage.\[1\] When agents choose whether they prefer to limit revenues in the ex-ante stage, their decisions depend on their beliefs about what revenue level will prevail in the ex-post stage if there is no limit. All agents face this complicated problem: forecasting the ideal \( R \) of the ex-post median voter. Median ex-post ideal \( R \) is a random variable, because it is a function of a population of random variables.

\[1\] We consider revenue limits rather than tax rate limits for two reasons. First, the vast majority of limits enacted during the tax revolt were tax revenue limits rather than tax rate limits. Second, tax rate limits are only effective if they create binding revenue limits.
6.3 Case-based Software Agents

The programming team must also make agents that are Case-based Software Agents (CBSA). In doing so you can evaluate the robustness of the analytic results of the program as well as explore interactions between agents. There are a few core ideas in Cased-based Decision Theory. The agents take their experiences of past actions and results of those actions and use that to make decisions about current problems, they judge how similar the situations are to each other and then take the course of action that is most likely to maximize utility. This is in contrast to expected utility where agents have an infinite state space and probabilities assigned to all the states.

6.4 Primitives

This decision process that an agent goes through has three types of primitives: first, the primitives of CBDT; second, the CBDT representation; third, a decision environment.

The primitives and representation of CBDT define aspects of decision making internal to the agent. The primitives of CBDT are: a finite set of actions $A$, a finite set of problems $P$, a set of results $R = \mathbb{R}$, which includes $r_0 = 0$ to be interpreted as “this action was not chosen,” and the set of cases $C = P \times A \times R$. Moreover, a set $M \subseteq C$ is memory of this agent. From the CBDT representation, the agent is also endowed with a similarity function $s : P \times P \to \mathbb{R}_+$. The similarity value is interpreted as: the higher a similarity value $s(p, q)$, the more relevant is problem $q$ to problem $p$.

The decision environment defines those aspects of the series of problems the agent faces that are external to the agent. This environment both stands apart from the decision theory and is unknown to the agent. We capture the decision environment with a function (algorithm) called the problem-result map or $PRM$. The $PRM$ is the transition function of the environment. It takes as input the current problem $p \in P$ the agent is facing, the action $a \in A$ that the agent has chosen, a well defined Social Decision Function $S$, and some vector $\theta \in \Theta$ of environmental characteristics. The $PRM$ returns the outcome of these four inputs: namely, it returns a result $r \in R$; the next problem $p' \in P$ that the agent faces; and a potentially modified vector of environmental characteristics $\theta' \in \Theta$ while the Social Function $S$ remains unchanged overtime. I.e.:

$$PRM : P \times A \times S \times \Theta \to R \times P \times \Theta$$

The Social Decision Function is a predefined function that the agents have agreed to abide by. This function $S$ maps individual actions $a$ into a Social Action $a^s$ and result $r$. We are adding one additional step into the individual CBDT introduced by GilboaSchmeidler95. In our specification the Social Decision Function does not change over time. Also, the results of the Social Decision directly map into the memory $M$ and therefore the similarity function $s(\cdots)$. In this context defining or voting on the form of the Social Decision Function is not one of the problems that the agents face. Agents are aware of the the characteristics of the Social Decision Function, but make decisions based on the similarity function and their memory as opposed to just a projection of beliefs over an infinite number of future states resulting from the collective actions of themselves and other agents in the Social Decision Function.

---

2Indeed, all the CBDT decision-maker ‘knows’ about the problems she is facing is specified in the endowed similarity function above and her memory of cases at any given point in time.


6.5 Algorithms of the implementation

Input: problem $p$, memory $M$.

1. For each $a \in A$:
   
   (a) Construct $M_a = \{(q,a,a^s,r) | \exists q \in P, r \in R \text{ s.t. } (q,a,a^s,r) \in M\}$
   
   (b) If $M_a \neq \emptyset$,
   
   $U_a = \sum_{(q,a,a^s,r) \in M_a} s(p,q)r$
   
   Else
   
   $U_a = r_0 = 0$.

2. Construct set $BEST$

   $BEST = \left\{ a \in A \mid U_a = \max_{b \in A} \{U_b\} \right\}$

3. If $#(BEST) = 1$ then let $a^*$ be the sole entry in $BEST$. Else choose one element uniformly from the set $BEST$ and assign that to $a^*$.

Output: Selected actions $a^*$.

Figure 1: The Choice Algorithm

Figure 1 describes the choice algorithm which implements the core of CBDT. The agent faces a problem $p \in P$ and has a memory $M \subseteq C$. For each action $a$ that she has available to her, she consults her memory and collects those cases, $M_a$, in which she performed this act. Then she uses this subset of her memory to construct an ‘expected utility’ of that act, called here $U_a$. This value is a similarity-weighted payoff: a sum across the cases in $M_a$, the result times the similarity between the problem faced at that time and the current problem. The agent then chooses the action which corresponds to the maximum $U$. There is an additional step, left unspecified in the original CBDT: in the case of a tie, the agent randomizes uniformly over the acts which achieve this maximum.

Figure 2 describes a single choice problem faced by the agent. It imbeds a reference to the choice algorithm described in Figure 1. While the choice algorithm describes CBDT in the sense that it describes decision making, the algorithm described in Figure 2 embeds the agent in an environment and explicitly references that environment, in the call to $PRM$. In step one, the agent selects a best act, $a^*$. In step 3, the action is performed, in the sense that the environment of the agent reacts to the agent’s choice: the PRM takes the current problem $p$, the action selected by the agent $a^*$, and the characteristics unobserved by the agent $\theta$, and constructs a result, a next problem, and a next set of characteristics. Finally, the agent’s memory is augmented by the new case which was just encountered: that is, the case that was just experienced is added to the set $M$. Note that the choice problem maps a problem, characteristic, memory vector to another vector in the same
Input: problem $p$, memory $\mathcal{M}$, characteristics $\theta$.

1. Input $p, \mathcal{M}$ into choice algorithm (Figure 1). Receive output $a^\star$.
2. Let $(r, p', \theta') = PRM(p, a^\star, \theta)$.
3. Let $\mathcal{M}' = \mathcal{M} \cup \{(p, a^\star, r)\}$

Output: problem $p'$, memory $\mathcal{M}'$, characteristics $\theta'$.

Figure 2: A single choice problem.

space, so it can be applied iteratively.

Input: act $a$, problem $p$, characteristics $\theta$.

1. Add $a$ to list of agent’s choices.
2. If all agents have chosen, then calculate $\text{evaluate}(a^\star_i)$ which returns $a^\star$.
3. For each agent, calculate result $r$ from $a^\star$.
4. Characteristics $\theta'$, next problem $p'$; not yet defined.

Output: result $r$, problem $p'$, characteristics $\theta'$.

Figure 3: The Anderson/Pape PRM

In the PRM associated with this problem, in Figure 3, the agents join their collective decisions into a predefined Social Decision Function $S$ that results in some collective action $a^\star$ and result $r$ for each agent. Note that the PRM is not completely defined.

1. Initialize $\mathcal{M} = \emptyset$. Initialize $\theta$. Initialize first problem $p$.
2. Evaluate single choice problem with input $(p, \mathcal{M}, \theta)$. Receive output $(p', \mathcal{M}', \theta')$.
3. Evaluate: Does $(p', \theta')$ achieve ending condition? If so, stop. Else, return to previous step with new vector $(p', \mathcal{M}', \theta')$

Figure 4: A complete series of choice problems.

Figure 4 describes a complete series of problems faced by an agent. Initially, the agent is assumed to have an empty memory, and some initial problem $p$. Furthermore, it is assumed that there is some initial starting condition $\theta$. Essentially, thereafter, the single choice problem is repeated iteratively. In step three $p'$ and $\theta'$ are jointly evaluated against some ending condition, and if that ending condition is achieved, the algorithm halts.

\footnote{It is simple to modify the algorithm such that the agent starts with some non-empty memory.}
6.6 Example of Case-based Decision

Here we describe an implementation of the Cased-based primitives with decision process. The primitives are defined:

1. \( \mathcal{P} \) which contains sets of situations as vectors of (Individual Parameters (i.e. Income, Risk Preference), Environmental Parameters). Some of these values maybe null.

2. \( \mathcal{A} \) which contains \( a \)

3. \( \mathcal{R} \) which contains the resulting \( U_a \) and \( a^s \)

This problem is saved in the \( \mathcal{M}_a = (q, a, a^s, r) \). In the next step we encounter the second stage of the problem and use \( \mathcal{M} \) to make decisions about the next problem. The similarity function could be of many different forms: One possibility could be a weighted Euclidean Distance. Remember that \( p \) and \( q \) are two different problems that the agent is comparing.

\[
s(p, q) = \frac{1}{\sqrt{\sum_{i=1}^{4} (p_i - q_i)^2 \times w_i}} \tag{5}
\]

We assume that there are four numbers to compare in the vector and that \( w_i \) is the weight in the function given to the \( i^{th} \) variable in the vector. We could assume that the weights are equal. For example, if the problem \( p \) is a vector of \( (1, 2, 2, 3) \) and the problem \( q \) is a vector of \( (1, 1, 1, 1) \) then the similarity function would return a value of \( s(p, q) = \frac{1}{\sqrt{(1-1)^2 + (2-1)^2 + (2-1)^2 + (3-1)^2}} = \frac{1}{6} \). This tells us how similar the problems are with a larger number meaning the problems are relatively more similar. We can then observe the Utility that the problem \( p \) resulted in and compare and choose the comparison that results in the highest Utility weighted by similarity. If we were faced with a Problem \( p \) and had in our memory set problems \( q \) and \( t \) our result would be.

If there were two actions, vote yes or no.

- \( s(p, q) = \frac{1}{6} \) and \( U_q = 4 \) and the action taken \( a_q \) was a Yes vote
- \( s(p, t) = \frac{1}{4} \) and \( U_t = 3 \) and the action taken \( a_t \) was a No vote

We would choose action No for problem \( p \), because we get a score of \( \frac{2}{3} \) for the yes vote and a score of \( \frac{3}{4} \) for the no vote.

6.7 Case-based Decision: Modeling Choices

When we implement CBDT we need to make choices about the parameters such as what the individual and environmental characteristics contain. We also make other important choices to create a memory set, and choose the number of agents, etc... In this section we put down some of those important choices and potential starting points. We may need to try options and discover patterns in the results that can inform our future decisions to calibrate the model.
• Individual parameters: income and wealth

• Environmental parameters: Statistics of all others’ wealth and income (mean, median, variance)

• Group Size: Start off with small groups to populate Memory then move agents to bigger groups

• $r_0$: Choose a value of Utility for actions not taken, Maybe the median of all possible actions?

• How many possible actions? Start with 10 uniformly distributed actions?

• What experiments to run?