Guide to code for "Optimal Epidemic Control in Equilibrium with Imperfect Testing and Enforcement"*

Thomas Phelan† Alexis Akira Toda‡

October 10, 2022

This document provides a guide to the code used to produce the figures in the paper "Optimal Epidemic Control in Equilibrium with Imperfect Testing and Enforcement". All code is written in Python 3.6.5 and is located at https://github.com/alexisakira/epidemic_equilibrium. If you spot errors or have questions please email the first author at tom.phelan@clev.frb.org. There are four scripts:

- main.py: imports classes, calibration, and PRME_PBE, and produces all of the figures in the paper.
- calibration.py: fixes the parameters used throughout the analysis.
- classes.py: contains class constructors and functions used to produce figures.
- PRME_PBE.py: computes Perfect Recall Markov Equilibrium and Perfect Bayesian Equilibrium allocations and compares both the paths of activity over time and the recursive representation of the activity function (i.e. computes differences over the state space).

Section 1 describes the class constructors and Section 2 describes functions not attached to a particular class (these are mainly used for the construction of plots).

*The views stated herein are those of the authors and are not necessarily those of the Federal Reserve Bank of Cleveland or the Board of Governors of the Federal Reserve System.

†Federal Reserve Bank of Cleveland. Email: tom.phelan@clev.frb.org.

‡Department of Economics, University of California San Diego. Email: atoda@ucsd.edu.

1 Class constructors

Class constructors are contained in classes.py:

1. SIR_PBE: Perfect Bayesian Equilibrium.
2. SIR_SPP: Social Planner's Problem.
3. SIR_ME: Perfect Recall Markov Equilibrium (only used in the appendix).

The following sections describe the methods used in each class.

1.1 SIR_PBE

In the methods that follow, "a" will refer to the actions of an individual unknown agent, and "a_tilde" will refer to the average action of all other unknown agents that each agent takes as given. The SIR_PBE class constructor contains the following methods:

1. u(self,a): flow utility function, given by $u(a) = (a^{1-\alpha} - 1)/(1-\alpha)$ for $\alpha \neq 1$ and $u(a) = \ln a$ for $\alpha = 1$.
2. u_inv(self,U): inverse of the utility function.
3. tran_func(self,ind,a_tilde): the transition probabilities for the aggregate state (S, I) divided by Δ_t. This produces a dictionary with keys (-1,0), (0,1) and (0,-1), indicating a down transition in S, an upward transition in I and a downward transition in I.
4. tran_func_id(self,ind,a,a_tilde): the transition probabilities for the individual state divided by Δ_t.
5. T(self,a,a_tilde): the linear operator in the definition of the finite-state Bellman equation for the individual unknown agent.
6. Vupdate(self,a,a_tilde): returns the value function of unknown infected agent given their action a and the actions of other agents a_tilde.
7. c_func(self,V,a_tilde): $\sigma\mu\beta r^{-1}[V_{I_k}(S,I) - V_U(S,I)]I$, where $\mu = S/(1-\sigma+\sigma S)$. This function is defined for convenience and is used in update of optimal policy.
8. opt_action(self,c,a_tilde): computes the optimal action of an unknown individual agent given the constant c in 7.

9. polupdate(self,V,a_tilde): computes optimal action of an unknown agent given continuation value function V. Calls opt_action and c_func.

10. solveV(self,a_tilde): computes value function of an unknown agent given the average action of other agents a_tilde. Repeatedly iterates on polupdate and Vupdate and returns both the value function and the optimal action of the unknown agent.

11. solve(self): computes equilibrium by iterating on solveV given an initial guess of a_tilde $\equiv \underline{a}$ everywhere. Returns both the utility function and the equilibrium activity level.

12. simul_path(self,T,K,init,V,a_rec,a): takes as given the number of days T, the subperiods within a day K, the initial condition init, the utility function V, the recommended policy function a_rec (which might be a), and the poliy function a that governs the evolution of the state. It returns paths for the population shares (S, I, R and D), the recommended activity level, the actual activity level, and the path of lifetime utility experienced by an unknown agent (in units of *activity*).

13. T_func(self,A,B,C): creates a sparse matrix of size $M := (N_0 + 1)(N_1 + 1)$, where $N = (N_0, N_1)$ is the grid over S and I.

1.2 SIR_SPP

This is the class constructor for the social planner's problem. It contains the following methods:

1. u(self,a): flow utility function, given by $u(a) = (a^{1-\alpha} - 1)/(1 - \alpha)$ for $\alpha \neq 1$ and $u(a) = \ln a$ for $\alpha = 1$.

2. u_inv(self,U): inverse of the utility function.

3. cost(self,aU): the flow objective that the planner wishes to minimize as a function of the action aU taken by the unknown infected agents.

4. tran_func_SPP(self,ind,aU): the transition probabilities for the aggregate state divided by Δ_t.

5. T(self,aU): the linear operator in the definition of the finite-state Bellman equation for the planner.

6. cand(self,c): the solution to the first-order condition in the planner's problem. This is gives a "candidate" optimal activity because one must take bounds into consideration.

7. polupdate_SPP(self,C): computes optimal policy for the planner given continuation value of the cost function C.

8. Cupdate_SPP(self,aU): computes the cost function for the planner given the policy function aU.

9. solve(self): computes cost function and optimal policy of the planner.

10. a_hat(self): the state-contingent value of the action below which the probability of leaving the I grid vanishes.

11. simul_path(self,T,K,init,V,a_rec,a): analogous to simul_path in SIR_PBE class.

12. T_func: analogous to T_func in SIR_PBE class.

1.3 SIR_ME

This is the class constructor for the social planner's problem. It is more complicated because the minimal state space for the problem of the unknown type is three-dimensional. In our numerical method, we exploit the monotonicity of the belief variable μ, and solve for the value function at the lowest μ first and iterating backwards. Write $\{\mu_1, \ldots, \mu_{N_2}\}$ for the values of μ. In all of the following a_tilde is a two-dimensional vector. This class constructor contains the following methods:

1. u(self,a): analogous to u in SIR_PBE class.

2. u_inv(self,a): analogous to u_inv in SIR_PBE class.

3. tran_func(ind, a_tilde): analogous to tran_func in SIR_PBE class.

4. tran_func_id(ind,a,a_tilde,k): transitions for the idiosyncratic variables. k here represents the value of $\mu = \mu_k$. Two components: the first is the downward transition in μ, denoted $p^{-\mu}$, and the second component is the transition probability for developing symptoms.

5. T(self,a,a_tilde,k): transition operator for a particular value of $\mu = \mu_k$.

6. simul_path(self,T,K,init,V,a_rec,a): analogous to simul_path in SIR_PBE class.

7. T_func: analogous to T_func in SIR_PBE class.

8. Vupdate(self,a,Vlow,a_tilde,k): fixed $\mu = \mu_k$, takes as given the utility function Vlow $= V(\cdot, \cdot, \mu_{k-1})$ for the agent at the lower value of μ, the behavior of the other agents a_tilde when $\mu = \mu_k$, and computes the value of the unknown agent.

9. c_func(Vlow,V,a_tilde,k): this is shorthand for $c = x\mu_k \beta r^{-1} I$, where

$$x = \sigma(V_{I_k} - V(S, I, \mu_k)) + (1 - \sigma)(V(S, I, \mu_{k-1}) - V(S, I, \mu_k))/\Delta_\mu.$$

10. opt_action(self,c,a_tilde): optimal action written as a function of c (which absorbs the dependence on k).

11. polupdate(self,Vlow,V,a_tilde,k): find the optimal policy at μ_k given the current guess V of the value function at μ_k and the value at μ_{k-1}, Vlow $= V(\cdot, \cdot, \mu_{k-1})$.

12. solveVslice(self,Vlow,a_tilde,k): compute the value function $V(\cdot, \cdot, \mu_k)$ given the actions of all other agents at μ_k and the value Vlow $= V(\cdot, \cdot, \mu_{k-1})$.

13. solveV(self,a_tilde): find the value function of the unknown type given the actions of the other agents.

14. solve(self): find the competitive equilibrium by beginning with the guess $\tilde{a} \equiv \underline{a}$ everywhere.

15. a_hat(self): the state-contingent value of the action below which the probability of leaving the I grid vanishes.

16. simul_path(self,T,K,init,V,a_rec,a): analogous to simul_path in SIR_PBE class.

17. T_func: analogous to T_func in SIR_PBE class.

2 Functions

This section describes the purpose of each function in classes.py that is not a method of one of the above class constructors. Note that the only parameters that vary in our numerical examples are the diagnostic rate σ, the expected number of years until arrival T, and the activity of the known infected agents a_I.

- expGrid: the exponential grid for the state space.
- exp_death: takes as given the arrival rate of the vaccine and the path of all population shares and computes the expected death toll (note that this is different from the cumulative death toll conditional on no vaccine arrival).
- results(sigma,T_vac,aIk) takes as given σ, T and a_I and for all three solution concepts (myopic, perfect Bayesian equilibrium, and the efficient allocation) produces:
  - paths for population shares, activity, recommended (both static efficient and efficient) activity, utility and the effective reproduction number;
  - recursive representation for the cost of the pandemic, the utility of an unknown agent, and activity; and
  - miscellaneous quantities such as herd immunity, time until convergence and a dictionary for the instances of classes.

  This function creates all of the objects we plot in the paper (apart from the perfect recall figures considered in the appendix). The tuple produced enters as an argument for the following plotting functions.
- SIRD_plots: produces figures for population shares.
- contour_plots: produces figures for contours of activity.
- activity_plots: produces figures for paths for activity over time and along the equilibrium path.
- DS_plots: produces figures for paths for deaths and susceptible shares (conditional on no arrival of vaccine).
- robust_sigma_plots: takes as argument list of diagnostic rates sigma_list, expected vaccine arrival T_vac and activity of known infected agent aIk and produces figures for welfare loss and expected death toll.

- robust_T_vac_plots: takes as argument diagnostic rate sigma, list of expected vaccine arrival dates T_vac_list, and activity of known infected agent aIk and produces figures for welfare loss and expected death toll.