Algorithm for the Hopenhayn-Rogerson (1993) Model: Python Implementation

Suraj Kumar

November 12, 2025

[1]

1 Overview

The following algorithm outlines the steps to solve the Hopenhayn-Rogerson model using the provided Python code. The model computes the stationary recursive competitive equilibrium with heterogeneous firms, firing costs, and endogenous entry/exit.

2 Initialization and Discretization

1. Set model parameters: α, β, c_f, c_e, θ, τ, productivity process ($\bar{z}_{\log}$, ρ, σ_ϵ), and numerical parameters (grid sizes, tolerances).
2. Discretize the productivity process using Tauchen's method:
   - Compute the state space grid for $\log z$ and transition matrix P.
   - Compute the stationary distribution g of productivity for entrants.
3. Create the employment grid: $\{n_j\}_{j=1}^{N_n} = [0, k \cdot n^*(\max z, p)]$, where $n^*(z,p) = (\alpha p z)^{1/(1-\alpha)}$ and k is a multiplier.

3 Value Function Iteration (VFI) for Incumbent Firms

Solve the incumbent firm's problem via VFI to find the value function $V(z,n)$ and optimal policy $n'(z,n)$.

1. Initialize $V^{(0)}(z,n) = 0$ for all states.
2. For iteration $t = 1$ to max iterations:
   - Compute expected continuation values: $E[V(z',n')|z] = P \cdot V^{(t-1)}$.
   - For each state (z_i, n_j):
     - Compute exit value: $V_{exit} = -\tau \cdot \max(0, n_j - 0)$.
     - For each possible $n' \in \{n_k\}_{k=1}^{N_n}$:
       - Compute profit: $\pi(z_i, n_j, n_k) = p z_i n_k^\alpha - n_k - \tau \max(0, n_j - n_k) - c_f$.
       - Compute objective: $\pi + \beta E[V(z', n_k)|z_i]$.
     - Choose $n' = \arg\max$ objective, or 0 if exit is better.

[1] This algorithm description was generated with the assistance of an AI language model (Grok by xAI). The AI was provided with the context of the Hopenhayn-Rogerson (1993) model and the associated Python code implementation. The final content was reviewed and edited by the author to ensure accuracy and clarity. The author also used the AI Language model to help facilitate the writing process and the speed optimization of the python code.

- Set $V^{(t)}(z_i, n_j) = \max(V_{exit}, \text{max objective})$.
- Check convergence: if $\max |V^{(t)} - V^{(t-1)}| < \epsilon_{VFI}$, stop.

3. Output: $V(z, n)$, $n'(z, n)$.

4 Finding the Equilibrium Price p^*

Find p^* such that the entry value equals the entry cost: $\beta E[V(z, 0)] = c_e$.

1. Coarse grid search:
   - Evaluate entry value $V_e(p) = \beta \sum_i V(z_i, 0) g_i$ over a grid of p.
   - Find bounds where $V_e(p) \geq c_e$.
2. Bisection on p:
   - While not converged:
     - Set $p = (p_L + p_H)/2$.
     - Solve VFI to get $V(z, n)$.
     - Compute $V_e(p)$.
     - If $|V_e(p) - c_e| < \epsilon_p$, stop.
     - Else, adjust bounds: if $V_e(p) > c_e$, set $p_H = p$; else $p_L = p$.
3. Output: p^*, $V^*(z, n)$, $n'^*(z, n)$.

5 Computing the Stationary Distribution

Find the stationary distribution $\mu(z, n)$ for $m = 1$ entrant per period.

1. Initialize $\mu^{(0)}(z, n) = 1/(N_z N_n)$.
2. For iteration $t = 1$ to max iterations:
   - Apply transition operator:
     - For incumbents: For each $n' > 0$, $\mu^{(t)}(z', n') = \sum_{z,n} P(z'|z) \mu^{(t-1)}(z, n) \cdot 1\{n'(z, n) = n'\}$.
     - For entrants: $\mu^{(t)}(z, 0) = g(z)$.
   - Check convergence: if $\max |\mu^{(t)} - \mu^{(t-1)}| < \epsilon_\mu$, stop.
3. Output: $\mu(z, n)$ (normalized for $m = 1$).

6 Market Clearing and Equilibrium Mass

Find m^* such that aggregate supply equals demand.

1. Compute aggregate production for $m = 1$: $Y = \sum_{z,n} z n^\alpha \mu(z, n)$.
2. Compute aggregate consumption: $C = \theta / p^*$.
3. Set $m^* = C/Y$.
4. Scale distribution: $\mu^*(z, n) = m^* \mu(z, n)$.
5. Compute aggregates: total employment $N = \sum_{z,n} n \mu^*(z, n)$, etc.
6. Output: m^*, $\mu^*(z, n)$, aggregates.

7 Overall Execution

1. Instantiate the model with parameters.
2. Call price-finding routine to get p^* and policies.
3. Compute stationary distribution.
4. Perform market clearing to get m^*.
5. Report equilibrium outcomes (price, mass, employment, etc.).

This algorithm assumes convergence and uses the specified numerical methods (e.g., Tauchen discretization, VFI, bisection). Adjust tolerances and grid sizes for accuracy vs. speed. For full details, refer to the code comments.

8 Equilibrium outcomes by firing tax

Below we report the model equilibrium outcomes for three values of the firing tax (Tau). Numbers are rounded to three decimal places.

8.1 Core equilibrium outcomes

Table 1: Core equilibrium outcomes by firing tax (3 decimals)

Tau	Equil. Price	Mass of Entrants	Total Mass of Firms	Total Employment	Avg. Firm Size
0.0	1.009	0.234	0.459	77.254	189.885
0.2	1.050	0.155	0.389	80.032	205.907
0.5	1.084	0.207	0.411	77.675	171.654

8.2 Flows and transition rates

Table 2: Exit/entry and job flow rates by firing tax (3 decimals)

Tau	Exit Rate	Entry Rate	Job Creation	Job Destruction
0.0	0.510	0.510	0.266	0.266
0.2	0.398	0.398	0.173	0.173
0.5	0.503	0.503	0.125	0.125

9 Aggregate outcomes by firing tax

Table 3: Aggregate outcomes for different firing tax levels (3 decimal places)

Tau	Aggregate Production	Total Employment	Aggregate Labor Productivity
0.0	99.137	77.254	1.283
0.2	95.251	80.032	1.190
0.5	92.225	77.675	1.187

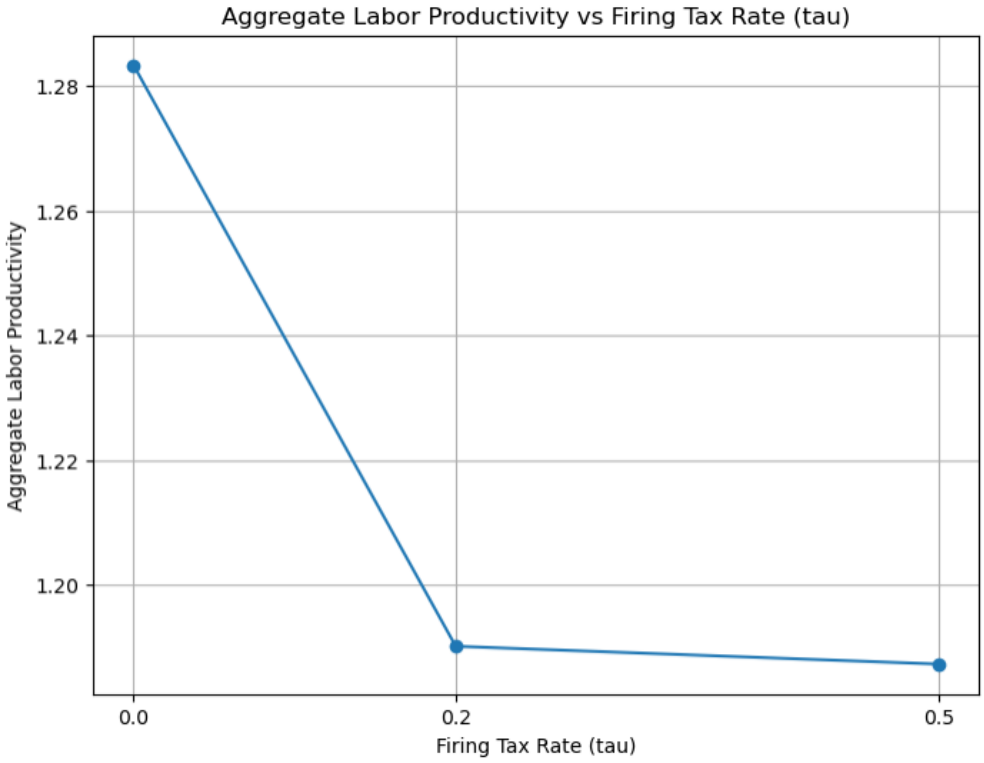

Figure 1: Aggregate Labor Productivity by Firing Tax

The misallocation increases with higher firing taxes, leading to lower aggregate labor productivity as shown in Figure 1. This is consistent with the intuition that firing costs discourage optimal firm adjustments, resulting in inefficiencies in the allocation of labor across firms.