HW1 FNCE937

Luke Min, Antony Anyosa

October 13, 2019

Abstract

Solving the problem of the firm in both discrete and continuous time and understanding the distribution of firms in equilibrium.

1 Solving the Problem of the Firm in Discrete Time

Consider the problem os the firm of maximizing present discounted dividends with constant interest rates

$$max_{\{k_t,l_t\}_{t=0}^{\infty}} \mathbb{E}_0 \sum_{t=0}^{\infty}\left[\frac{1}{(1+r)^t}d_t\right]$$
$$s.t\ d_t = \pi_t - i_t - \phi_t$$

where d_t is the amount of gross distributions and the other variables obey the usual relationship

$$i_t = k_{t+1} - (1-\delta)k_t$$
$$\pi_t = a_t k_t^{\theta_1} l_t^{\theta_2} - Wl_t$$

The stochastic process for a_t is given by

$$ln(a_t) = (1-\rho)ln\bar{a} + \rho ln(a_{t-1}) + \sigma\varepsilon_t$$

With parameters

Table 1: Parameters

Param	Value
ρ	0.8
r	0.02
δ	0.1
θ_1	0.3
θ_2	0.6
W	2
σ	0.1

(a) Plot the profit function as a function of the capital stock. Do this for the highest and the lowest value of the shock a. This gives you an idea of what the value function should look like. First split the optimization problem into the static and dynamic. The static part can be solved from

$$\pi^*(k_t) = max_{l_t}(a_t k_t^{\theta_1} l_t^{\theta_2} - W l_t)$$

The FOC is

$$\begin{aligned}
\theta_2 a_t k_t^{\theta_1} l_t^{\theta_2 - 1} &= W \\
\implies l_t^{\theta_2 - 1} &= \frac{1}{\theta_2 a_t} W k_t^{-\theta_1} \\
\implies l_t &= \left(\frac{W k_t^{-\theta_1}}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}}
\end{aligned}$$

Plug this back in the objective to solve for the value function π^* of the static optimization problem.

$$\begin{aligned}
\pi^*(k_t) &= a_t k_t^{\theta_1} \left(\frac{W k_t^{-\theta_1}}{\theta_2 a_t}\right)^{\frac{\theta_2}{\theta_2 - 1}} - W \left(\frac{W k_t^{-\theta_1}}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}} \\
&= a_t k_t^{\theta_1} k_t^{\frac{\theta_2 \theta_1}{1-\theta_2}} \left(\frac{W}{\theta_2 a_t}\right)^{\frac{\theta_2}{\theta_2 - 1}} - W \left(\frac{W}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}} k_t^{\frac{\theta_1}{1-\theta_2}} \\
&= a_t k_t^{\frac{\theta_1}{1-\theta_2}} \left(\frac{W}{\theta_2 a_t}\right)^{\frac{\theta_2}{\theta_2 - 1}} - W \left(\frac{W}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}} k_t^{\frac{\theta_1}{1-\theta_2}} \\
&= \left(a_t \left(\frac{W}{\theta_2 a_t}\right)^{\frac{\theta_2}{\theta_2 - 1}} - W \left(\frac{W}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}}\right) k_t^{\frac{\theta_1}{1-\theta_2}}
\end{aligned}$$

Let $a_t^* = \left(a_t \left(\frac{W}{\theta_2 a_t}\right)^{\frac{\theta_2}{\theta_2 - 1}} - W \left(\frac{W}{\theta_2 a_t}\right)^{\frac{1}{\theta_2 - 1}}\right)$ and $\gamma = \frac{\theta_1}{1-\theta_2}$. Then

$$\pi^*(k_t) = a_t^* k_t^{\gamma}$$

To get values of a_t I discretize the space using quadrature methods as in Tauchen. I then do this plot on MATLAB with the parametrization given in Table 1 and two extreme values of a_H and a_L.

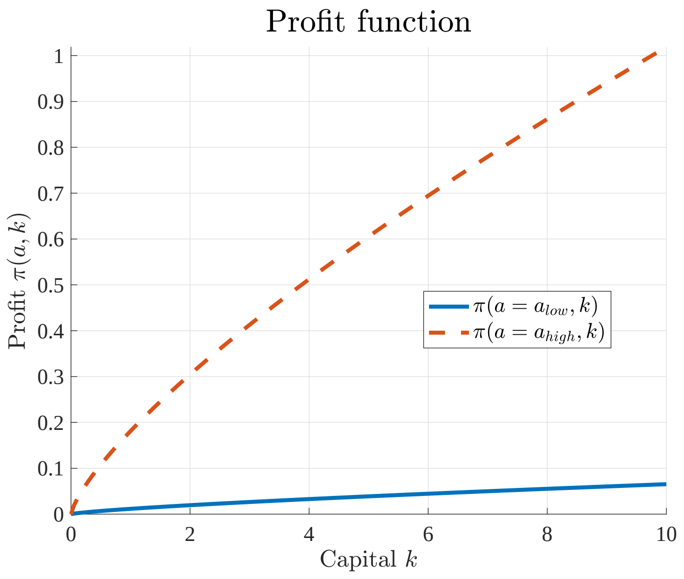


Figure 1.1:

(b) Value function iteration. The Bellman equation for the problem of the firm is

$$V\left(a,k\right)=\max_{k'}\left\{\pi\left(a,k\right)-i\left(k',k\right)-\phi\left(i,k\right)+E\left[\frac{1}{1+r}V\left(a',k'\right)\right]\right\}$$

such that

$$\begin{aligned}\pi\left(a,k\right)&=ak^{\theta_1}l^{\theta_2}-Wl^{\theta_2}\\ i\left(k',k\right)&=k'-\left(1-\delta\right)k\\ \phi\left(k',k\right)&=b_0k+b_1\left(\frac{i}{k}-\delta\right)^2k\end{aligned}$$

Value function:

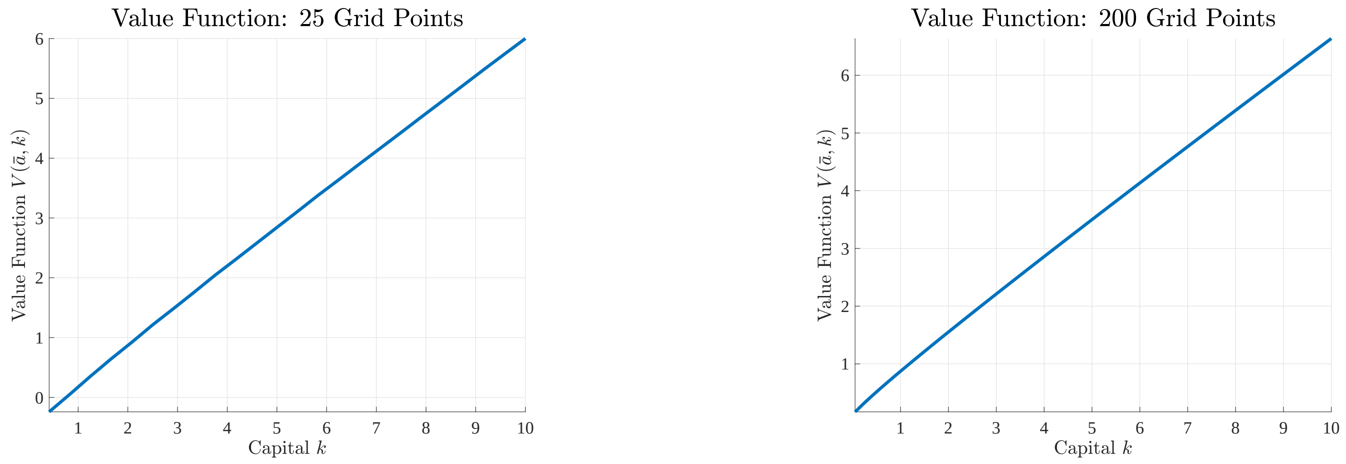


Figure 1.2:

(c) Optimal investment and financing policies

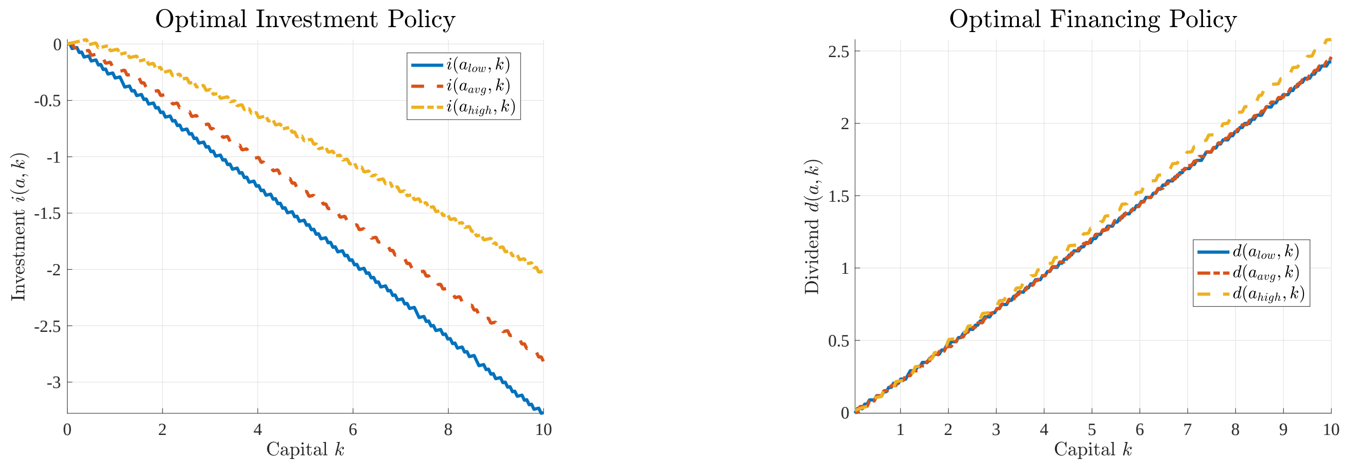


Figure 1.3:

(d) I do sensitivity analysis and solve the model under different parametrizations.

Figure 1.4: $b_0 = 0, b_1 = 0.5$

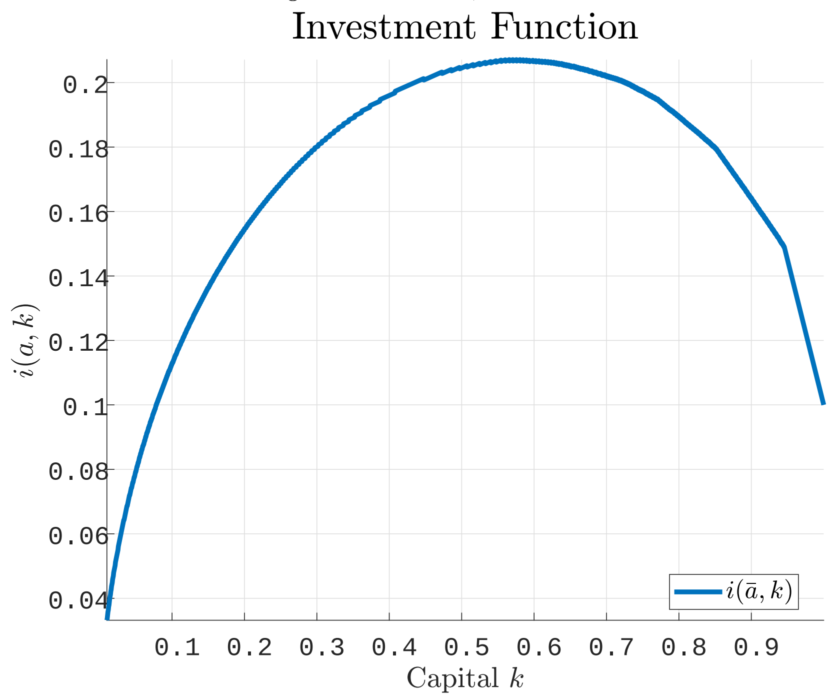


Figure 1.5: $b_0 = 0, b_1 = 10$

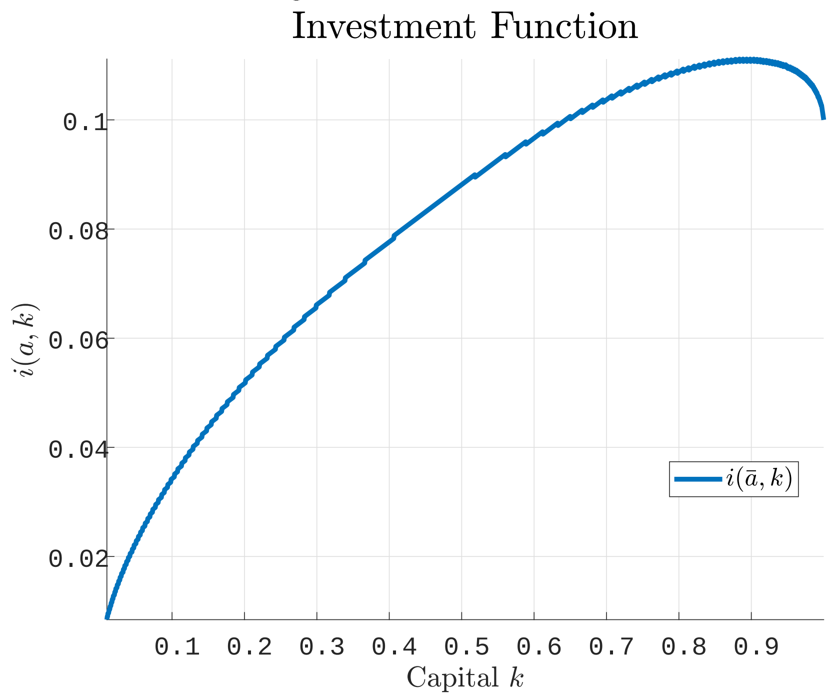


Figure 1.6: $b_0 = 0, b_1 = \begin{cases} 0.5 & i \geq \delta k \\ 10 & i \leq \delta k \end{cases}$

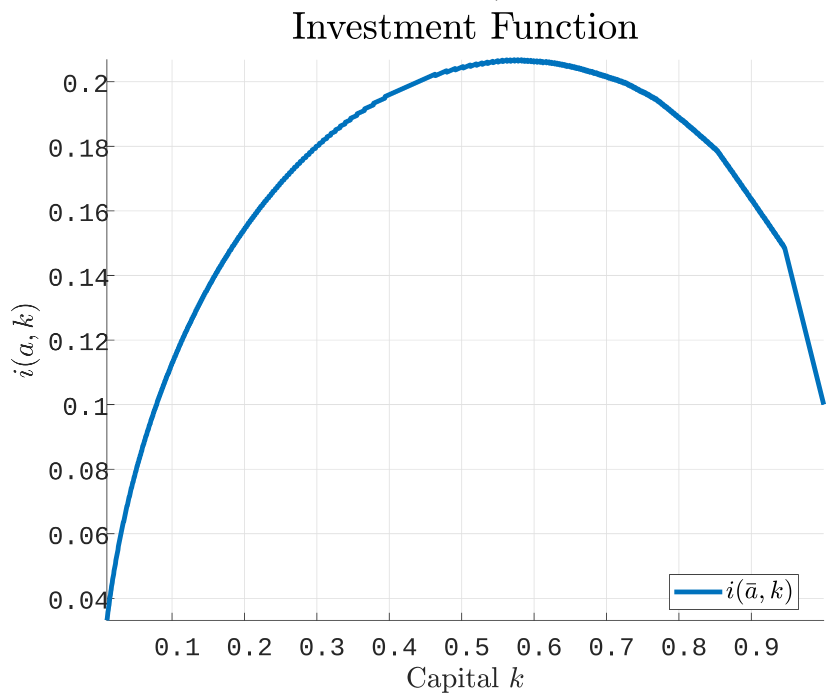
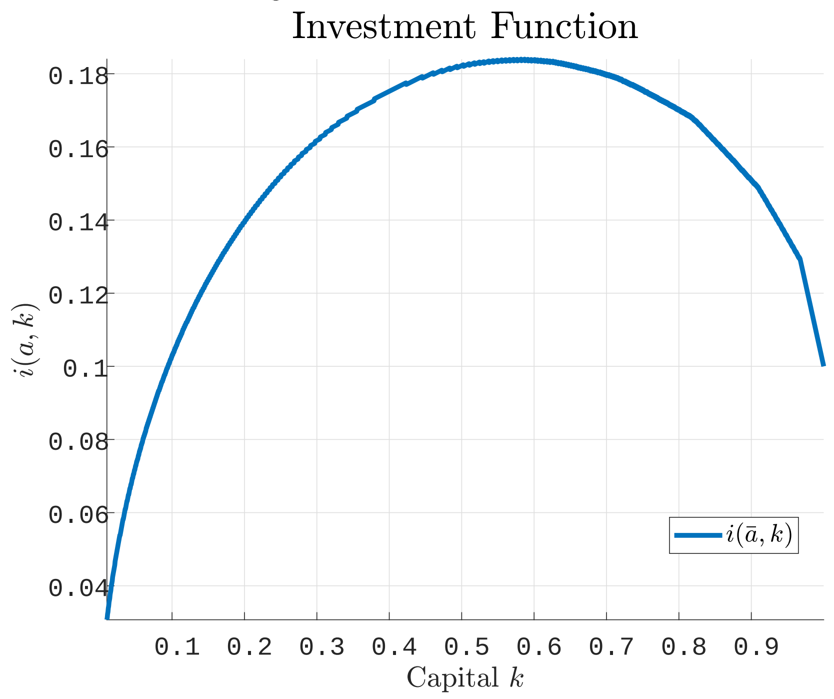


Figure 1.7: $b_0 = 0.02, b_1 = 0.5$



2 Solving the Problem of the Firm in Continuous Time

(a) No uncertainty In the first case, the Hamilton Jacoby Bellman equation becomes

$$rv(k) = \max_i \left\{ ak^\alpha - i - \frac{b}{2}\left(\frac{i}{k} - \delta\right)^2 k + v'(k)(i - \delta k) \right\}$$

where we can write $\phi(i,k) = \frac{b}{2}\left(\frac{i}{k} - \delta\right)^2 k$ and $\dot{k} = i - \delta k$. We have that $a = 1$ fixed, so a is no longer a state variable. First order condition gives

$$v'(k) = 1 + b\left(\frac{i}{k} - \delta\right) = 1 + \frac{\partial \phi}{\partial i}$$

which we use later.

Solving this equation on the computer requires us to evaluate the $v'(k)$. We do this by using the finite difference method. We approach as follows.

First, construct a grid of $k = \begin{bmatrix} k_1 & \cdots & k_j & \cdots & k_I \end{bmatrix}'$ with I equidistant points. So we can define $\Delta k = k_{j+1} - k_j$ for all $j = 1, \cdots, I-1$. Similarly, we can write $v(k_j) = v_j$ for convenience. Since we start with a guess for $v(\cdot)$, we can add a superscript to write v_j^n to refer to the value function at the n-th iteration.

Evaluating the any derivative numerically confronts us with a choice. In particulaar, we can have take the forward difference, so that we have $v'_{j,FD} = \frac{v_{j+1} - v_j}{\Delta}$, or backward difference $v'_{j,BD} = \frac{v_j - v_{j-1}}{\Delta}$. We use both for differing regions. The numerical step of updating v_j^n's becomes

$$\frac{v_j^{n+1} - v^n}{\Delta} + rv^{n+1} = d_j^n + \underbrace{\frac{v_{j+1}^{n+1} - v_j^{n+1}}{\Delta k}\left(\dot{k}_{j,FD}^n\right)^+}_{\text{forward difference}} + \underbrace{\frac{v_j^{n+1} - v_{j-1}^{n+1}}{\Delta k}\left(\dot{k}_{j,BD}^n\right)^-}_{\text{backward difference}}$$

There's a quite a bit of explaining to do. Here, Δ is the step of each iteration, Δk is the (constant) space between capital grid. Moreover,

$$\dot{k}_{j,FD}^n = i_{j,FD}^n - \delta k_j$$
$$\dot{k}_{j,BD}^n = i_{j,BD}^n - \delta k_j$$

meant to approximate the capital drift given in the problem, and investment is in turn obtained from the first order condition condition where $v'_{j,FD} = 1 + b\left(\frac{i_{j,FD}}{k_j} - \delta\right)$, rearranged to give

$$i_{j,FD} = k_j\left[\frac{1}{b}\left(v'_{j,FD} - 1\right) + \delta\right]$$
$$i_{j,BD} = k_j\left[\frac{1}{b}\left(v'_{j,BD} - 1\right) + \delta\right]$$

at each iteration. Also, we have $x^+ = \max(x, 0)$ and $x^- = \min(x, 0)$. d_j, on the other hand, already has the forward and backward differences embeded into it, in the sense that

$$d_j = ak_j^\alpha - i_j - \frac{b}{2}\left(\frac{i_j}{k_j} - \delta\right)^2 k_j$$

where $i_j = i_{j,FD}$ whenever $\dot{k}_{j,FD}^n > 0$ and $i_j = i_{j,BD}$ whenever $\dot{k}_{j,BD}^n < 0$. For now we assume this covers all cases.

Now we can group the right hand side of the numerical step by v's to get

$$\frac{v_j^{n+1} - v_j^n}{\Delta} + rv_j^{n+1} = d_j + \underbrace{\left[-\frac{\left(\dot{k}_{j,BD}^n\right)^-}{\Delta k}\right]}_{\equiv x_j} v_{j-1}^{n+1} + \underbrace{\left[-\frac{\left(\dot{k}_{j,FD}^n\right)^+}{\Delta k} + \frac{\left(\dot{k}_{j,BD}^n\right)^-}{\Delta k}\right]}_{\equiv y_j} v_j^{n+1} + \underbrace{\left[\frac{\left(\dot{k}_{j,FD}^n\right)^+}{\Delta k}\right]}_{\equiv z_j} v_{j+1}^{n+1}$$

Let's do some linear algebra. We need a way to collect across $j = 1, \cdots, I$ points to turn the above into a matrix equation. Let $v^n = \begin{bmatrix} v_1^n & \cdots & v_j^n & \cdots & v_I^n \end{bmatrix}'$ as before. To get the right hand side terms, we construct a matrix A^n at each iteration where

$$A^n = \begin{bmatrix} y_1 & z_1 & & & & & & & \\ x_2 & y_2 & z_2 & & 0 & & & & \\ & x_3 & y_3 & z_3 & & & & & \\ & & & \ddots & & & & & \\ & 0 & & x_j & y_j & z_j & 0 & & \\ & & & & & & & & \\ & & & & & & x_{I-1} & y_{I-1} & z_{I-1} \\ & & & & & & & x_I & y_I \end{bmatrix} \quad v^{n+1} = \begin{bmatrix} v_1^{n+1} \\ \vdots \\ v_{j-1}^{n+1} \\ v_j^{n+1} \\ v_{j+1}^{n+1} \\ \vdots \\ v_I^{n+1} \end{bmatrix}$$

Then,

$$\frac{v^{n+1} - v^n}{\Delta} + r v^{n+1} = d + A^n v^{n+1}$$

To solve for v^{n+1},

$$\left[\left(\frac{1}{\Delta} + r\right) I_{I \times I} - A^n\right] v^{n+1} = d + \frac{1}{\Delta} v^n$$

Value function:

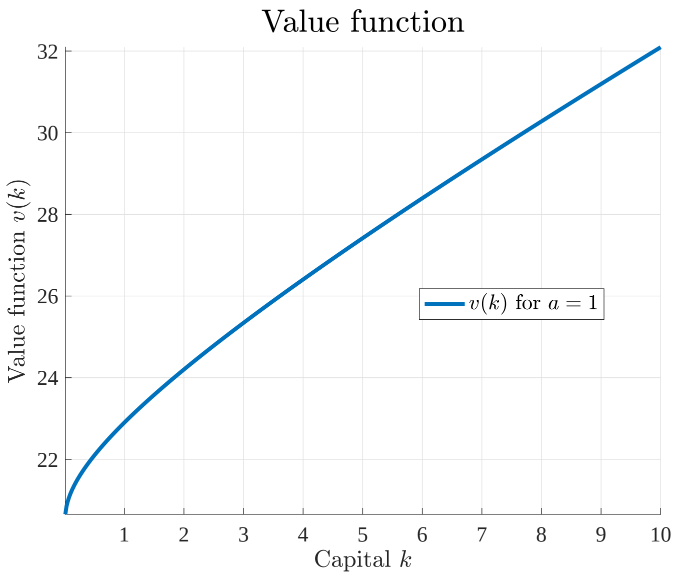


Figure 2.1:

Investment and capital drift policy:

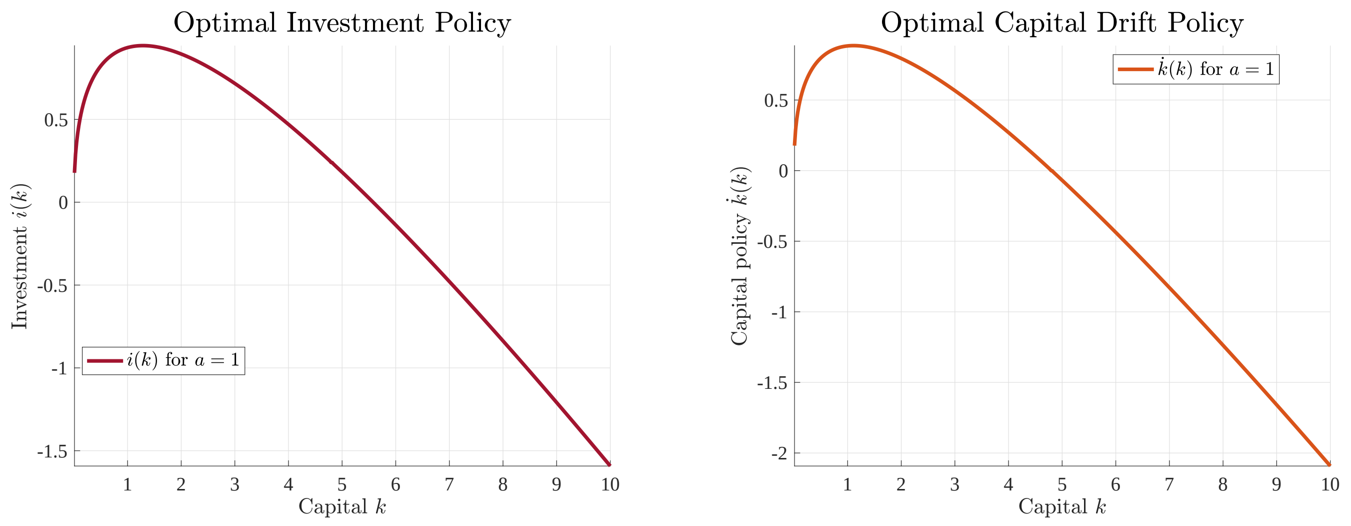


Figure 2.2:

(b) Poisson uncertainty Now we have the added layer of complexity with the Poisson uncertainty. The HJB equation is

$$rv(a,k) = \max_i \left\{ d(a,i,k) + v_k(a,k)(i-\delta k) + \sum_{a'} p(a'|a)\left[v(a',k) - v(a,k)\right] \right\}$$

Let's see how we can implement this numerically. We will have

$$v_{i,j} = v(a_j, k_i)$$

with j being the state subscript and i being the capital grid point subscript. Then, stack the v vector as

$$v = \begin{bmatrix} v_{1,1} \\ \vdots \\ v_{I,1} \\ v_{1,2} \\ \vdots \\ v_{I,2} \\ v_{1,3} \\ \vdots \\ v_{I,3} \end{bmatrix}$$

cycling through the capital grid for each $i = 1, 2, 3$.

We also need to redefine investment points $i_{i,j,FD}$, $i_{i,j,BD}$, $\dot{k}^n_{i,j}, _{FD}$, and $\dot{k}^n_{i,j}, _{FD}$:

$$i_{i,j,FD} = k_j \left[\frac{1}{b}\left(v'_{i,j,FD} - 1\right) + \delta\right]$$
$$i_{i,j,BD} = k_j \left[\frac{1}{b}\left(v'_{i,j,BD} - 1\right) + \delta\right]$$

which is used to compute

$$\dot{k}^n_{i,j,FD} = i^n_{i,j,FD} - \delta k_i$$
$$\dot{k}^n_{i,j,BD} = i^n_{i,j,BD} - \delta k_i$$

We want to construct another A matrix such that each row of the product $A^n \cdot v^{n+1}$ becomes the followng. Write $p\left(a_s \middle| a_j\right) = p_{j,s}$.

$$
\begin{aligned}
& v_k(a,k)(i-\delta k) + \sum_{a'} p\left(a' \middle| a\right)\left[v(a',k) - v(a,k)\right] \\
&= \frac{v_{i+1,j}^{n+1} - v_{i,j}^{n+1}}{\Delta k}\left(\dot{k}_{i,j,FD}^n\right)^+ + \frac{v_{i,j}^{n+1} - v_{i-1,j}^{n+1}}{\Delta k}\left(\dot{k}_{i,j,BD}^n\right)^- + \sum_{s\in\{1,2,3\}} p_{j,s}\left[v_{i,s}^{n+1} - v_{i,j}^{n+1}\right] \\
&= \underbrace{\left[-\frac{\left(\dot{k}_{i,j,BD}^n\right)^-}{\Delta k}\right]}_{\equiv x_{i,j}} v_{i-1,j}^{n+1} + \underbrace{\left[-\frac{\left(\dot{k}_{i,j,FD}^n\right)^+}{\Delta k} + \frac{\left(\dot{k}_{i,j,BD}^n\right)^-}{\Delta k} - \sum_{s\neq j} p_{j,s}\right]}_{\equiv y_{i,j}} v_{i,j}^{n+1} + \underbrace{\left[\frac{\left(\dot{k}_{i,j,FD}^n\right)^+}{\Delta k}\right]}_{\equiv z_{i,j}} v_{i+1,j}^{n+1} + \sum_{s\neq j} p_{j,s} v_{i,s}^{n+1}
\end{aligned}
$$

Take,

$$
A = \begin{bmatrix}
y_{1,1} & z_{1,1} & & & & p_{1,2} & p_{1,3} & \\
x_{2,1} & y_{2,1} & z_{2,1} & & & & & \\
 & x_{3,1} & y_{3,1} & z_{3,1} & & & & \\
 & & & & & & & \\
p_{2,1}I_{N\times N} & & & & \ddots & & & p_{2,3}I_{N\times N} \\
 & & & & & & & \\
p_{3,1}I_{N\times N} & & & p_{3,2}I_{N\times N} & & & & \\
 & & & & & & &
\end{bmatrix}, \quad v^{n+1} = \begin{bmatrix} v_{1,1} \\ \vdots \\ v_{I,1} \\ v_{1,2} \\ \vdots \\ v_{I,2} \\ v_{1,3} \\ \vdots \\ v_{I,3} \end{bmatrix}
$$

Value function:

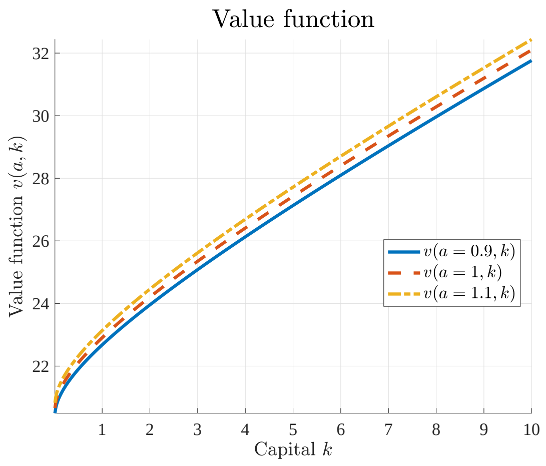


Figure 2.3:

Investment and capital drift policy:

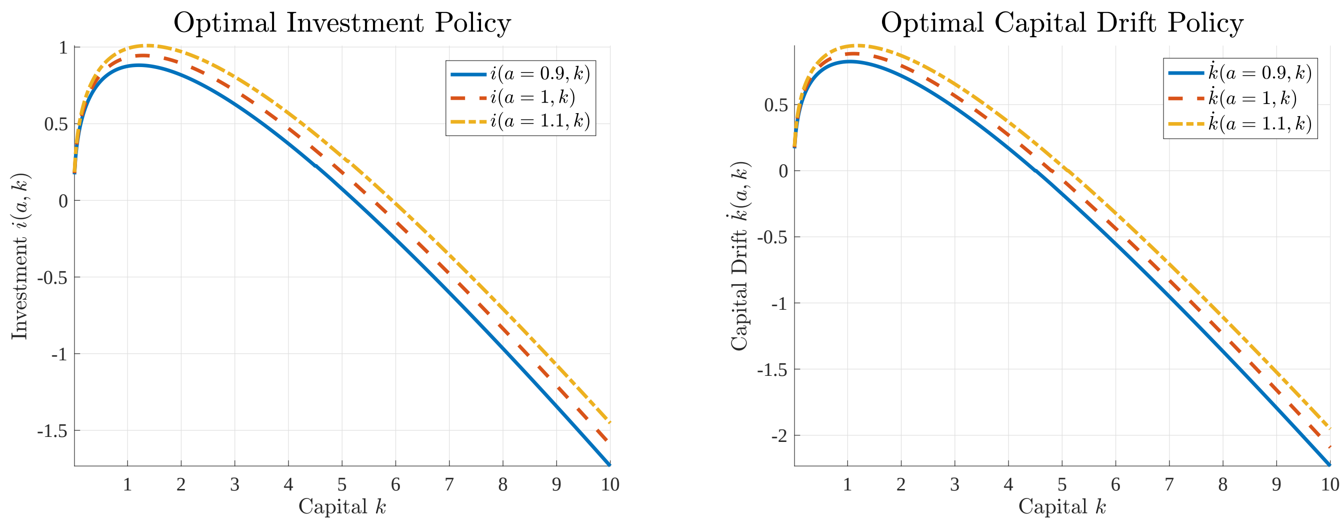


Figure 2.4:

3 Corporate Investment in Equilibrium - Gomes (2001)

We first need to set the notation straight.

$$v(z,k) = \max_{k'} \{d(z,k,k') + M \max\{E_{z'} v(z',k'), 0\}\}$$

such that

$$d(z,k,k') = \pi(z,k) - i(k,k')$$
$$i(k,k') = k' - (1-\delta)k + 0.5\left(\frac{i}{k} - \delta\right)^2 k$$

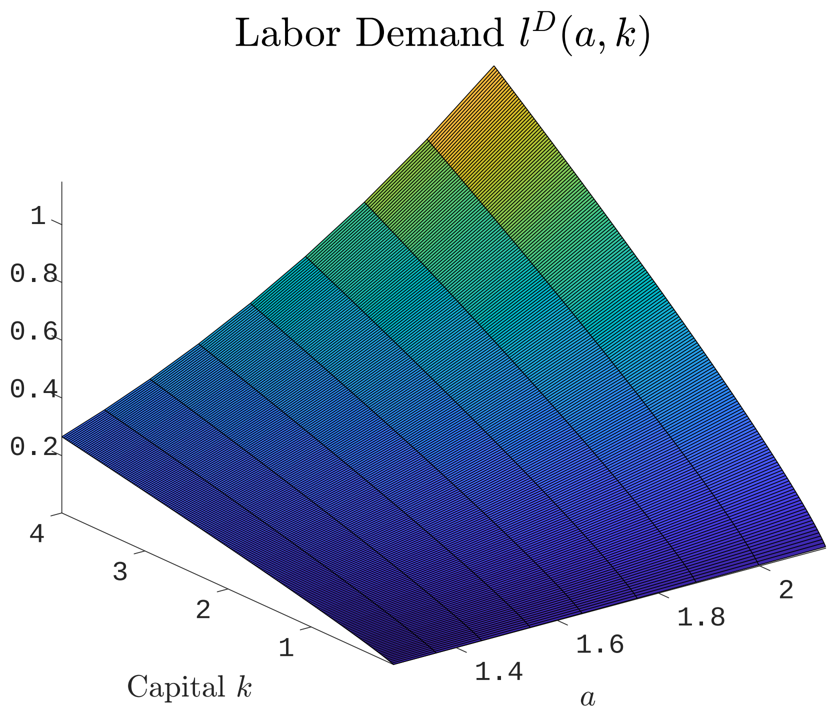
Operating profits are

$$\pi(z,k) = zk^{\alpha_k} l^{\alpha_l} - Wl - f$$

Labor choice/demand l is chosen to maximize profits

$$l(z,k) = \left(\frac{z\alpha_l k^{\alpha_k}}{W}\right)^{\frac{1}{1-\alpha_l}}$$

This is what the labor demand function looks like given the parametrization.



There is also aggregate labor supply given by

$$L^s = BW^{0.1}$$

With parameter values

Table 2: Parameters

Param	Value
M	0.95
δ	0.1
ρ	0.95
α_k	0.3
α_l	0.65
W	2
σ	0.02
$log(\bar{z})$	0.5

Law of motion for the cross-sectional distribution First we need to understand the law of motion for the cross-sectional distribution T, which is computed from (1) the law of motion for the exogenous state P; and (2) the law of motion for the endogenous state Q. Suppose we have S states and N points for the grid point of capital P is given by applying the Tauchen method on the given AR(1) process, and is an $S \times S$ matrix with the structure

$$P = \begin{bmatrix} & & \\ & p_{ij} & \\ & & \end{bmatrix}, \quad \text{where } p_{ij} = \Pr\left(a_{t+1} = a_j \middle| a_t = a_i\right)$$

Here p_{ij} denotes the entry of P in row i and column j. The law of motion for the endogenous state Q is obtained by the policy function, and is a $N \times NS$ matrix with the structure

$$Q = \begin{bmatrix} & & \\ & q_{ij} & \\ & & \end{bmatrix}, \quad \text{where } q_{ij} = \Pr\left(k_i \middle| (a_l, k_m) \text{ with } j = m + lN\right)$$

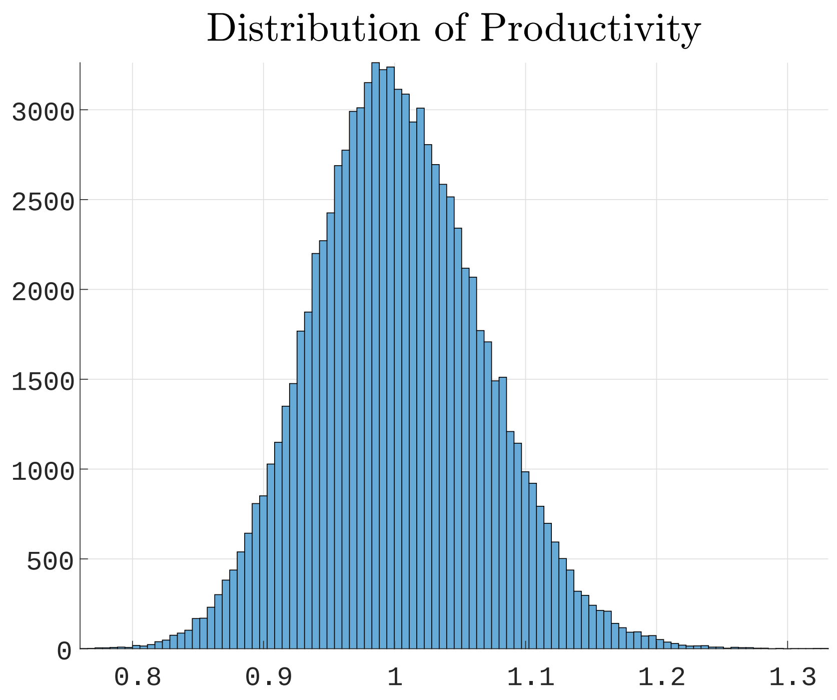
Lastly, the law of motion for the cross-section distribution T is obtained through P and Q, and is an $NS \times NS$ matrix with the structure

$$T = \begin{bmatrix} & & \\ & T_{ij} & \\ & & \\ & & \end{bmatrix}, \quad \text{where } t_{ij} = \Pr\left((a_l, k_m) \text{ with } i = m + lN | (a_n, k_s) \text{ with } j = s + nN \right)$$

Entry

We would like to compute the probability of a new entrant arriving next period with productivity shock a' which will be drawn from the invariant distribution of shocks.

Then I approximate the distribution of productivity using quadrature methods. This is the long run invariant distribution of the productivity shocks from which entrants will draw their distributions from.



With entry, the law of motion for the cross-sectional distribution then becomes

$$\mu' = T\mu + \gamma E$$

Where $\gamma(a', k = 0)$ is the probability of a firm that enters the market with draw a' and with zero capital. We know that in expectation a firm will only enter if given $k = 0$, if it draws a a high enough a' such that that its value function its positive. Given the existence of fixed costs. We can set entry costs as 0 since setting fixed costs f is isomorphic to setting entry costs.

Exit

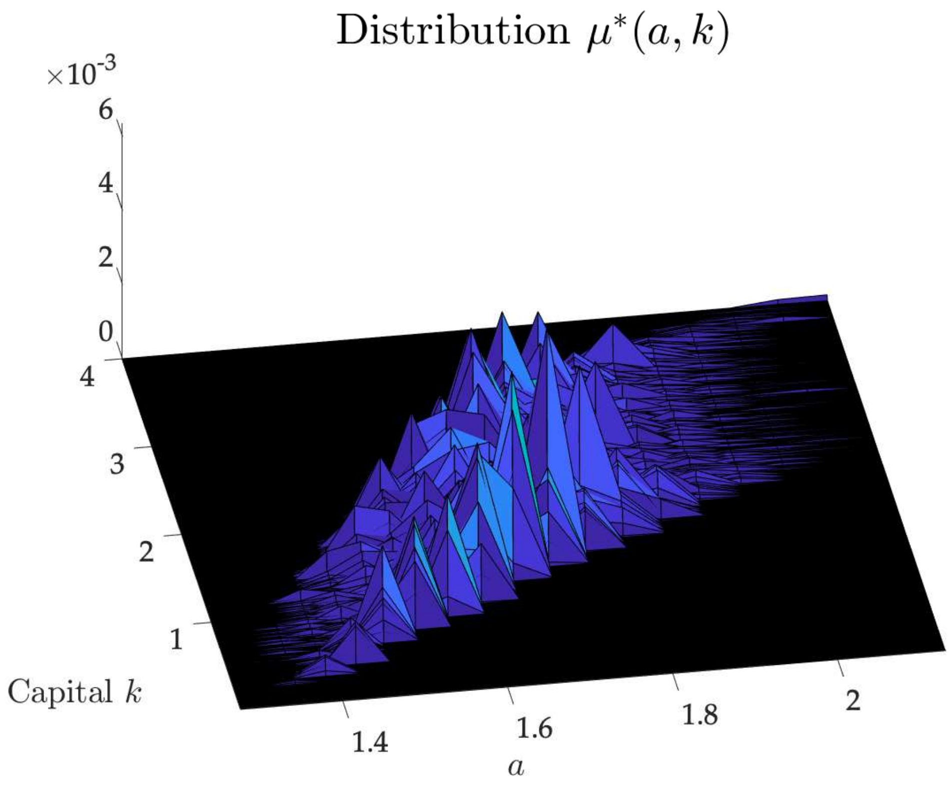
Then firm exit is incorporated by having firms who have negative value function and are better off exiting the market. This is computed using an exit threshold $a^*(k)$ such that

$$x(a,k) = \begin{cases} 1 & a > a^*(k) \\ 0 & a < a^*(k) \end{cases}$$

Now after incorporating entry and exit we can solve for the invariant distribution as follows.

$$\mu^*(a,k) = (I - T \circ X)^{-1} \gamma E$$

Where E, the mass of entrants, is set to 1. We obtain the following, which is consistent with the notion of larger firms being the most productive.



Given this distribution with measure one, we can compute aggregate demand for labor since we have solved for $l^d(a,k)$. Hence integrating gives the aggregate demand for labor

$$L^d = \int l^d(a,k) d\mu^*(a,k)$$

In equilibrium this should be equal to the aggregate supply for labor given by

$$L^s = BW^{0.1} = L^d$$

Hence solving for the constant B gives

$$B = W^{-0.1} \int l^d(a,k) d\mu^*(a,k) = 0.2884$$

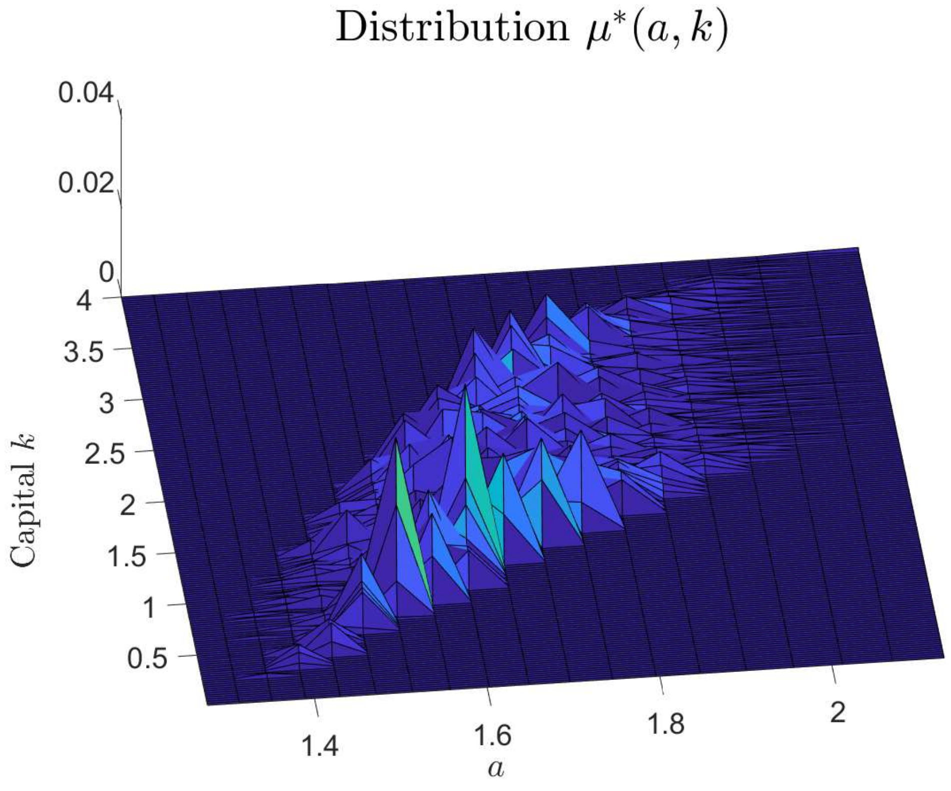
Solving for the mass of firms

Now given $B = 1$. We know that labor supply is then $L^s = W^{0.1} = 2^{0.1} = 1.07$. Now we are expected to solve for the mass of firms N such that the market for labor clears. We know that the mass of firms will be strictly related to the number of entrant firms E. Hence we can solve for E

$$\mu(a,k) = N\mu^*(a,k)$$
$$L^s = \int_0^1 l^d(a,k)d\mu(a,k)$$

Implementing this numerically implies solving for equilibrium. I implement a simple bisection method to look for E^* such that the labor market clears given prices W.

This gives the following distribution of firms:



And the number of firms in equilibrium is approximately 3.5 given our parametrization. This is intuitive since firms are choosing capital between 0 and 4 and there is only 1.07 units of labor supplied in the market so we should not expect a large number or firms.

4 Distribution of Firms in Continuous Time

The following are given in the problem:

$$\pi(z,k) = zk^\alpha, \quad k \in \{k_1, k_2\}$$
$$dz = \mu z dt + \sigma z dW$$
$$\phi = b_0 + b_1(k_2 - k_1)$$

(a) Analytical expression for the value function after exercising option Let's denote the value function after exercise of option as $v(z)$. Then we have

$$rv(z) = \pi(z) = zk_2^\alpha$$

so that

$$\boxed{v(z) = \frac{1}{r}zk_2^\alpha}$$

(b) Analytical expression for the firm's value function before it exercises its investment option Let u denote the value function before exercising the option. Given $k = k_1$, this only depends on z. So we write $u(z)$ to denote this dependence.

To obtain the HJB equation, consider the change of the value function over a infinitesimally small interval dt:

$$\begin{aligned} ru(z)\,dt &= zk_1^\alpha dt + E[du] \\ &= zk_1^\alpha dt + E\left[\frac{\partial u}{\partial z}dz + \frac{1}{2}\frac{\partial^2 u}{\partial z^2}(dz)^2\right] \\ &= zk_1^\alpha dt + \frac{\partial u}{\partial z}\mu z dt + \frac{1}{2}\frac{\partial^2 u}{\partial z^2}\sigma^2 z^2 dt \end{aligned}$$

Dividing by dt gives the equation

$$ru(z) = zk_1^\alpha + \mu z\frac{\partial u}{\partial z} + \frac{\sigma^2}{2}z^2\frac{\partial^2 u}{\partial z^2}$$

or in the usual form for differential equations,

$$0 = u'' + \frac{2\mu}{\sigma^2}\left(\frac{1}{z}\right)u' - \frac{2r}{\sigma^2}\left(\frac{1}{z^2}\right)u + \frac{2k_1^\alpha}{\sigma^2}\frac{1}{z}$$

This is a second order linear non-homogeneous ordinary differential equation, and has the following class of solutions:

$$u(z) = \frac{zk_1^\alpha}{r-\mu} + A_1 z^{\beta_1}k_1^\alpha + A_2 z^{\beta_2}k_1^\alpha$$

Let's verify and solve for β_1 and β_2 by plugging it into the differential equation:

$$\begin{aligned} \mu z u'(z) &= \mu\left[\frac{zk_1^\alpha}{r-\mu} + A_1\beta_1 z^{\beta_1}k_1^\alpha + A_2\beta_2 z^{\beta_2}k_1^\alpha\right] \\ \frac{\sigma^2}{2}z^2 u''(z) &= \frac{\sigma^2}{2}\left[A_1\beta_1(\beta_1-1)z^{\beta_1}k_1^\alpha + A_2\beta_2(\beta_2-1)z^{\beta_2}k_1^\alpha\right] \end{aligned}$$

so that

$$\begin{aligned} r\left[\frac{zk_1^\alpha}{r-\mu} + A_1 z^{\beta_1}k_1^\alpha + A_2 z^{\beta_2}k_1^\alpha\right] = zk_1^\alpha &+ \mu\left[\frac{zk_1^\alpha}{r-\mu} + A_1\beta_1 z^{\beta_1}k_1^\alpha + A_2\beta_2 z^{\beta_2}k_1^\alpha\right] \\ &+ \frac{\sigma^2}{2}\left[A_1\beta_1(\beta_1-1)z^{\beta_1}k_1^\alpha + A_2\beta_2(\beta_2-1)z^{\beta_2}k_1^\alpha\right] \end{aligned}$$

Since this has to hold for all $z > 0$, group by z, z^{β_1} and z^{β_2} respectively to obtain that the following conditions that have to be satisfied:

$$\begin{aligned} \frac{rk_1^\alpha}{r-\mu} &= z + \frac{\mu k_1^\alpha}{r-\mu} \\ rA_1k_1^\alpha &= \mu A_1\beta_1 k_1^\alpha + \frac{\sigma^2}{2}A_1\beta_1(\beta_1-1)k_1^\alpha \\ rA_2k_1^\alpha &= \mu A_2\beta_2 k_1^\alpha + \frac{\sigma^2}{2}A_2\beta_2(\beta_2-1)k_1^\alpha \end{aligned}$$

The first condition holds immediately, and we set β_1 and β_2 such that the second and the third hold. Notice that both are symmetric, so we solve

$$rA_j k_1^\alpha = \mu A_j \beta_j k_1^\alpha + \frac{\sigma^2}{2} A_j \beta_j (\beta_j - 1) k_1^\alpha \quad \text{for } j = 1, 2$$

which gets us to the same quadratic equation for both

$$0 = \frac{\sigma^2}{2}\beta_j^2 + \left(\mu - \frac{\sigma^2}{2}\right)\beta_j - r$$

assuming $k_2 > 0$ and $A_j \neq 0$. The roots are

$$\boxed{\beta_j = \frac{\left(\frac{\sigma^2}{2} - \mu\right) \pm \sqrt{\left(\mu - \frac{\sigma^2}{2}\right)^2 + 2\sigma^2 r}}{\sigma^2}}$$

where we have one positive and one negative root. Let's say $\beta_1 > 0$ and $\beta_2 < 0$.

The boundary condition requires that

$$\lim_{z\to 0} u(z) = \lim_{z\to 0}\left[\frac{zk_1^\alpha}{r-\mu} + A_1 z^{\beta_1} k_1^\alpha + A_2 z^{\beta_2} k_1^\alpha\right] = 0$$

which means that $\beta_2 < 0$ will be a problem as that term will go to infinity. Thus we impose $A_2 = 0$ as a consequence.

Now so far we have

$$\boxed{u(z) = \frac{zk_1^\alpha}{r-\mu} + A_1 z^{\beta_1} k_1^\alpha}$$

for the value function before exercising the option.

(c) Compute optimal investment boundary and optimal amount of investment Let z^* be the boundary at which the firm optimally decides to invest. Then, the **value matching condition** requires that

$$v(z^*) = u(z^*) - \phi$$

or that

$$\frac{1}{r}(z^*) k_2^\alpha = \left[\frac{(z^*) k_1^\alpha}{r-\mu} + A_1 (z^*)^{\beta_1} k_1^\alpha\right] - [b_0 + b_1(k_2 - k_1)] \tag{4.1}$$

This gives us an expression for A_1 in terms of z^* and the parameters.

In addition, the **smooth pasting condition** requires that

$$v'(z^*) = u'(z^*)$$

or

$$\frac{1}{r}k_2^\alpha = \frac{k_1^\alpha}{r-\mu} + A_1\beta_1 (z^*)^{\beta_1 - 1} k_1^\alpha \tag{4.2}$$

Combining 4.2 and 4.1 gets us to

$$\boxed{z^* = \phi\left(\frac{\beta_1}{1-\beta_1}\right)\left[\frac{k_2^\alpha}{r} - \frac{k_1^\alpha}{r-\mu}\right]^{-1}}$$

and

$$\boxed{A_1 = \frac{\phi}{(z^*)^{\beta_1} k_1^\alpha (1-\beta_1)}}$$

Let $V(\cdot)$ be the value function of the whole problem. Then,

$$V(z) = \begin{cases} \frac{zk_1^\alpha}{r-\mu} + A_1 z^{\beta_1} k_1^\alpha & \text{if } z < z^* \\ \frac{1}{r} z k_2^\alpha - \phi & \text{if } z \geq z^* \end{cases}$$

When picking k_2, it should be chosen such that it maximizes

$$v\left(z^*;k_2\right)=\frac{1}{r}\left(z^*\right)k_2^{\alpha}=\frac{k_2^{\alpha}}{r}\left[b_0+b_1\left(k_2-k_1\right)\right]\left(\frac{\beta_1}{1-\beta_1}\right)\left[\frac{k_2^{\alpha}}{r}-\frac{k_1^{\alpha}}{r-\mu}\right]^{-1}$$

which is given by the k_2 that solves the first order condition

$$\frac{\partial v\left(z^*;k_2\right)}{\partial k_2}=\left(\frac{\beta_1}{1-\beta_1}\right)\left\{b_1\left(\frac{r-\mu}{r\left(1-\frac{k_1^{\alpha}}{k_2^{\alpha}}\right)-\mu}\right)-\left(b_0+b_1\left(k_2-k_1\right)\right)\left[\frac{\left(r-\mu\right)\left(\alpha r k_1^{\alpha}k_2^{-\alpha-1}\right)}{\left(r\left(1-\frac{k_1^{\alpha}}{k_2^{\alpha}}\right)-\mu\right)^2}\right]\right\}=0$$

MATLAB Code to Numerically Solve the GE Model

Main Script Firm Problem Discrete Time

```
% Main script for homework 1 for fnce 937

% Initialization:

clc; clear all; close all;
addpath('./Tauchen', './output', './functions')

%% Problem 1a

r      = 0.02;
delta  = 0.1;
theta1 = 0.3;
theta2 = 0.6;
W      = 2;

% Parameters for Tauchen approximation:
rho       = 0.8;
sigma     = 0.1;
n_tauchen = 9;
a_bar     = 1;
num_std   = 3;    % Number of standard deviations from mean



% Simulate 5-point discrete Markov Chain
[a_states,P,d1] = tauchen1(n_tauchen,a_bar,rho,sigma,num_std); % Output = [states; transition matrix; stationary dis
% mc1        = dtmc(p1);                           % declare as a Markov Chain object
% z1         = exp(z1);                             % exponentiate to convert log(y) to y
% sim_paths1 = z1(simulate(mc1, T));

% Set capital grid
k_grid = linspace(0,10,200);
a_states = a_states'; % Turn into a column vector

% Compute profit
profit_a_low  = compute_profit(k_grid,a_states(1),  W,theta1,theta2);
profit_a_high = compute_profit(k_grid,a_states(end),W,theta1,theta2);

fig_Q1_profit_func(k_grid,profit_a_low,profit_a_high,'output/Q1_profit_func.pdf')

%% Problem 1b: Value function iteration

% Adjustment costs parameter
b0 = 0;
b1 = 0.5;

% Convergence parameter
epsil = 1e-3;

[value_fine,   kp_fine]   = compute_value_func(k_grid,a_states,P,r,W,theta1,theta2,delta,b0,b1,epsil);

k_coarse_grid = linspace(0,10,25);

[value_coarse, kp_coarse] = compute_value_func(k_coarse_grid,a_states,P,r,W,theta1,theta2,delta,b0,b1,epsil);

%% plot
fig_Q1_value_func(k_grid,value_fine(5,:),  'output/Q1_value_func_fine.pdf')
fig_Q1_value_func_coarse(k_coarse_grid,value_coarse(5,:),'output/Q1_value_func_coarse.pdf')

%% Problem 1c: Plot optimal investment and financing

% Use capital policy to plot investment and financing
```

```
N = size(k_grid,2);
inv_policy = zeros(3,N); div_policy = zeros(3,N);
idx_vec = [1,5,9];

for j = [1, 2, 3]
    idx = idx_vec(j);
    inv_policy(j,:) = compute_inv(k_grid,kp_fine(idx,:),delta);
    div_policy(j,:) = compute_div(k_grid,kp_fine(idx,:),a_states(idx),W,theta1,theta2,delta,b0,b1);
end

fig_Q1_inv_policy(k_grid,inv_policy,'output/Q1_inv_policy.pdf') % inv_policy should be 3xN matrix

fig_Q1_div_policy(k_grid,div_policy,'output/Q1_div_policy.pdf')

%% Problem 1d: Show optimal investment policy for various adjustment costs:

% Store investment policy at the mean shock for different adjustment costs
inv_policy_adj_costs = zeros(4,N);

% 1. Standard: b0 = 0, b1 = 0.5
b0 = 0; b1 = 0.5;
[~,k_policy1]              = compute_value_func(k_grid,a_states,P,r,W,theta1,theta2,delta,b0,b1,epsil);
inv_policy_adj_costs(1,:) = compute_inv(k_grid,k_policy1(5,:),delta);

% 2. Standard: b0 = 0, b1 = 10
b0 = 0; b1 = 10;
[~,k_policy2]              = compute_value_func(k_grid,a_states,P,r,W,theta1,theta2,delta,b0,b1,epsil);
inv_policy_adj_costs(2,:) = compute_inv(k_grid,k_policy2(5,:),delta);

% 3. Asymmetric: b0=0, b1+=0.5, b1-=10
b0 = 0; b1_down = 10; b1_up = 0.5;
[~, kp]                    = compute_value_func_asym_adj_costs(k_grid,a_states,P,r,W,theta1,theta2,delta,b0,b1_down,b
inv_policy_adj_costs(3,:) = compute_inv(k_grid,kp(5,:),delta);

% 4. Discontinuous: b0 = 0.02, b1 = 0.5 (unless i = delta*k)
```

Main Script Firm Problem in Continuous Time

```
%% HW 1 Question 2

clc; clear all; close all;
addpath('./Tauchen', './output', './functions')

%%
% Declare parameter values
alpha = 0.3;
a     = 1;
b     = 0.5;
delta = 0.05;
r     = 0.05;

% Declare capital grid parameters
I     = 500;
kmin  = 0.01;
kmax  = 10;

% Create capital grid
k  = linspace(kmin,kmax,I)'; % Capital grid column vec
dk = (kmax-kmin) / (I-1);    % Step size

% Iteration parameters
maxit  = 100;
crit   = 10^(-6);
step   = 1000;

% Setup blank vectors
dV_f = zeros(I,1); % column vec
```

```
dV_b = zeros(I,1);
inv  = zeros(I,1);


% INITIAL GUESS
v0 = a * k.^(alpha);
v = v0;




%% Main loop for Part a


% maxit = 10; % Check with lower values first
for n = 1:maxit
    V = v;

    % Compute forward difference of va lue function
    dV_f(1:I-1) = (V(2:I) - V(1:I-1))/dk;
    dV_f(I)     = 1; % state constraint

    % Compute backward difference of value function
    dV_b(2:I) = (V(2:I) - V(1:I-1))/dk;
    dV_b(1)   = 1; % state constraint

    % Investment and capital policy with forward difference
    inv_f    = ((dV_f - 1)/b + delta) .* k;
    kDrift_f = inv_f - delta*k;

    % Investment and capital policy with backward difference
    inv_b    = ((1/b) * (dV_b - 1) + delta) .* k;
    kDrift_b = inv_b - delta*k;

    % Steady state:
    inv_0 = delta * k;
    dV_0  = 1 + b*(inv_0 ./ k - delta);

    % Indicator vectors for f/b difference
    If = kDrift_f > 0;
    Ib = kDrift_b < 0;
    I0 = (1-If-Ib); % 1 if neither of the above is true

    % Note: dV_upwind makes a choice of forward or backward difference depending
    % on the sign of the drift.

    % Grab forward backward diff's for the relavant regions
    dV_upwind = dV_f.*If + dV_b.*Ib + dV_0.*I0;
    inv       = ((1/b) * (dV_upwind - 1) + delta) .* k;
    d         = a*k.^(alpha) - inv - phi(inv,k,b,delta);
    kDrift    = kDrift_f.*If + kDrift_b.*Ib + 0.*I0; % Let zeros populate where neither backward or forward applies

    % X,Y,Z are block vectors that form A
    X = -min(kDrift_b,0)/dk;
    Y = -max(kDrift_f,0)/dk + min(kDrift_b,0)/dk;
    Z = max(kDrift_f,0)/dk;

    % Create sparse matrix A
    A = spdiags(Y,0,I,I) + spdiags(X(2:I), -1, I,I) + spdiags([0;Z(1:I-1)],1,I,I);

    % Setup the iterative system to be solved at each nth iteration:
    %   B(n) v(n+1) = C(n)
    % where
    %   B(n)    = 1/step + r - A(n)
    %   v(n+1) = next step value function
    %   C(n)    = d(n) + v(n) / step
    B = (1/step + r)*speye(I) - A;
    C = d + V/step;
    % Solve for v(n+1)
    V = B\C;

    % Compute update error
    err = max(abs(V-v));
    v   = V;

    % Show iteration
    disp(strcat('Iteration = ',num2str(n)))

    % Convergence criteria
    if err < crit
        disp('Value Function Converged, Iteration = ')
        disp(n)
        break
    end
end


%% Plot resulting graphs for Part (a)


fig_Q2_value_func(k,V,  'output/fig_Q2_value_func.pdf')
fig_Q2_inv_policy(k,inv,'output/fig_Q2_inv_policy.pdf')
fig_Q2_kDrift(k,kDrift, 'output/fig_Q2_kDrift.pdf')


%% Part (b): Add Poisson Uncertainty

% Structure for Poisson process of a
a = [0.9; 1.0; 1.1];
P = [0.5 0.5 0; 0.25 0.5 0.25; 0 0.5 0.5];
S = 3; % number of states

% Otherwise same parameters

% Declare capital grid parameters
I     = 500;
kmin  = 0.01;
kmax  = 10;

% Create capital grid
k  = linspace(kmin,kmax,I)';
dk = (kmax-kmin) / (I-1);   % Step size

% Construct k grid MATRIX
k_mat = k*ones(1,3); % Each column is k(a)

% Iteration parameters
maxit  = 100;
crit   = 10^(-6);
step   = 1000;

% Setup blank vectors
dV_f = zeros(I,S); % NOW A MATRIX
dV_b = zeros(I,S);
inv  = zeros(I,S);

% Setup constant sparse matrix Aswitch. Used to get probabilities in A
% (IS)x(IS) matrix of blocks.
Aswitch = [zeros(I)        P(1,2)*speye(I) P(1,3)*speye(I); ...
           P(2,1)*speye(I) zeros(I)        P(2,3)*speye(I); ...
           P(3,1)*speye(I) P(3,2)*speye(I) zeros(I)];

% INITIAL GUESS
v0 = k.^(alpha) * a'; % column x row .= matrix
v = v0;
```

```matlab
%% Main loop Part (b)

% maxit = 10; % Check with lower values first
for n = 1:maxit
    V = v;

    % Compute forward difference of va lue function
    dV_f(1:(I-1),:) = (V(2:I,:) - V(1:(I-1),:))/dk;
    dV_f(I,:)     = 1; % state constraint for the last row

    % Compute backward difference of value function
    dV_b(2:I,:) = (V(2:I,:) - V(1:I-1,:))/dk;
    dV_b(1,:)   = 1; % state constraint for the first row

    % Investment and capital policy with forward difference
    inv_f    = ((dV_f - 1)/b + delta) .* k_mat;
    kDrift_f = inv_f - delta*k_mat;

    % Investment and capital policy with backward difference
    inv_b    = ((1/b) * (dV_b - 1) + delta) .* k_mat;
    kDrift_b = inv_b - delta*k_mat;

    % Steady state:
    inv_0 = delta * k_mat;
    dV_0  = 1 + b*(inv_0 ./ k_mat - delta);

    % Indicator vectors for f/b difference
    If = kDrift_f > 0;
    Ib = kDrift_b < 0;
    I0 = (1-If-Ib); % 1 if neither of the above is true

    % Note: dV_upwind makes a choice of forward or backward difference depending
    % on the sign of the drift.

    % Grab forward backward diff's for the relavant regions
    dV_upwind = dV_f.*If + dV_b.*Ib + dV_0.*I0;
    inv       = ((1/b) * (dV_upwind - 1) + delta) .* k_mat;
    d         = k.^(alpha) * a' - inv - phi(inv,k_mat,b,delta);
    kDrift    = kDrift_f.*If + kDrift_b.*Ib + 0.*I0; % Let zeros populate where neither backward or forward applies

    % Sparse matrix A now looks different. Includes transition matrix P
    % Following notes, construct X,Y,Z matrices of size (IxS)
    X        = -min(kDrift_b,0)/dk;
    Z        = max(kDrift_f,0)/dk;

    % Constructing Y takes care. Manipulate P
    non_diag = P-diag(diag(P));             % same matrix as P with zeros on the diagonal
    P_other  = ones(I,1)*sum(non_diag,2)'; % sum of probability of exiting current state. same value across rows.
    Y        = -max(kDrift_f,0)/dk + min(kDrift_b,0)/dk - P_other;

    % Create block matrices for sparse matrix A
    A1=spdiags(Y(:,1),0,I,I)+spdiags(X(2:I,1),-1,I,I)+spdiags([0;Z(1:I-1,1)],1,I,I);
    A2=spdiags(Y(:,2),0,I,I)+spdiags(X(2:I,2),-1,I,I)+spdiags([0;Z(1:I-1,2)],1,I,I);
    A3=spdiags(Y(:,3),0,I,I)+spdiags(X(2:I,3),-1,I,I)+spdiags([0;Z(1:I-1,3)],1,I,I);

    % Create sparse matrix A using blocks and Aswitch
    A = [A1,          sparse(I,I), sparse(I,I); ...
         sparse(I,I), A2,          sparse(I,I); ...
         sparse(I,I), sparse(I,I), A3         ] + Aswitch;

    % Setup the linear system to be solved at each nth iteration:
    %   B(n) v(n+1) = C(n)
    % where
    %   B(n)    = (1/step + r) - A(n)
    %   v(n+1) = next step value function
    %   C(n)    = d(n) + v(n) / step
    B = (1/step + r)*speye(3*I) - A;
    % Stack columns to get a vector
    d_stack = d(:);
    V_stack = v(:);

    % Define C
    C = d_stack + V_stack/step;

    % Solve for v(n+1)
    V_stack = B\C;

    % Convert back to matrix
    V = reshape(V_stack, [I,3]);

    % Compute update error
    err = max(abs(V-v));
    v   = V;

    % Show iteration
    disp(strcat('Iteration = ',num2str(n)))

    % Convergence criteria
    if err < crit
        disp('Value Function Converged, Iteration = ')
        disp(n)
        break
    end
end

%% Plot functions

fig_Q2b_value_func(k,V,  'output/fig_Q2b_value_func.pdf')
fig_Q2b_inv_policy(k,inv,'output/fig_Q2b_inv_policy.pdf')
fig_Q2b_kDrift(k,kDrift, 'output/fig_Q2b_kDrift.pdf')


%% Misc functions

function adj_cost = phi(i,k,b,delta)
% Compute adjustment costs


adj_cost = b/2 * (i./k -delta).^2 .* k;
end
```

Main Script Discrete Equilibrium Firms

```
clc; clear all;
close all;
rng(199)
% FY Paper Heterogeneous Agents Models
% Initialization:

%cd 'C:\Users\anyosae\OneDrive - PennO365\WHARTON\CLASSES\SECOND-YEAR\FALL19\FNCE_937\PSETS\PS1\CODE'
%cd 'D:\OneDrive - PennO365\WHARTON\CLASSES\SECOND-YEAR\FALL19/FNCE_937/PSETS/PS1/CODE'
%cd '/Users/antonyanyosa/OneDrive - PennO365/WHARTON/CLASSES/SECOND-YEAR/FALL19/FNCE_937/PSETS/PS1/CODE'
% cd '/Users/antonyanyosa/Dropbox/FNCE937_PSETS/PS1/02 Matlab'
addpath 'output'
addpath 'functions'
addpath 'sample code'
addpath 'Tauchen'
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 1: The Problem of the firm in discrete time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Parameters
```

```
theta1  = 0.3;
theta2  = 0.65;
delta   = 0.1;
W       = 2;
sigma   = 0.02;
%sigma   = 0.2;
r       = 0.02;
rho     = .95;
b0      = 0;
b1      = 0.5;
ln_abar = 0.5; % Set High Enough so that there is Entry
M       = 0.95;
n2      = 50; % Length of Capital Grid
n       = 19; % States Grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NOW DO THE INCOME PROCESS
% Assume that the log(a) follows: log(a') = (1-rho)ln_abar+rho*log(a) + sigma*eps
% where eps is a truncated standardized normal on [-4,4].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T = 1000;       % Number of periods

% Simulate the loga process based on the exact distribution:
eps = randn(T+1,1);

% Preallocate
log_a    = zeros(1, T+1);   % vector of log(y)
log_a(1) = 1;

% Create log_a process
for h = 1:T
    log_a(h+1) = (1-rho)*ln_abar+rho*log_a(h) + sigma*eps(h+1);
end

% Convert to a process for y
a = exp(log_a);

%%%
% Approximating AR(1) with a Markov Chain
%%%
% Parameters for Tauchen approximation:

mu      = ln_abar;
num_std = 4;    % Number of standard deviations from mean

% Simulate 5-point discrete Markov Chain
[a_g,P,d] = tauchen1(n,mu,rho,sigma,num_std);   % Output = [states; transition matrix; stationary dist]
mc           = dtmc(P);                        % declare as a Markov Chain object
a_grid       = exp(a_g);                         % exponentiate to convert log(a) to a
sim_paths    = a_grid(simulate(mc, T));

% Histogram of distribution of technology
%hist_prod(a(500:end),100,'output/productivity.eps')

% Show results:
for t = [1000 T]
    % Number of periods for computing sample stats
    disp(['Number of periods = ', num2str(t)])

    % [Actual distribution; 5-point approx; 9-point approx]
    means        = [mean(a(1:t)) mean(sim_paths(1:t))];
    volatility   = [std(a(1:t))  std(sim_paths(1:t))];
    corr         = [autocorr(a(1:t),1); autocorr(sim_paths(1:t),1)];
    corr         = corr(:,2)';

    % Save to file
    a_stats      = [means; volatility; corr]
    filename     = ['output/a_stats_' num2str(t)];
    save(filename, 'a_stats')
end

%%%%%%%%%%%%%%%%%%%%%%%%
% Do Plot of Profit function
%%%%%%%%%%%%%%%%%%%%%%%%

R       = 1/M; % Gross Interest rate
kmin    = 0.01;
kmax    = 4;
k       = linspace(kmin,kmax,n2)';

% CALIBRATED FIXED Costs
f  = 0.07;

a_star      = (a_grid).*(W./(theta2.*a_grid)).^(theta2/(theta2-1))- (W).*(W./(theta2.*a_grid)).^(1/(theta2-1));
gamma       = theta1/(1-theta2);

profit_mat  = zeros(n,n2);

for i=1:n;
    profit_mat(i,:) = a_star(i).*k.^gamma-f;
end

fig_value(k,profit_mat','output/profit_function_EQ.eps')

l_star_mat=zeros(n2,n);

for i=1:n2
  for j=1:n
    l_star_mat(i,j)  = ((W.*k(i).^(-1*theta1))./(theta2*a_grid(j))).^(1/(theta2-1));
  end
end

%fig_surf_ldemand(a_grid,k,l_star_mat,'output/ldemand.eps') % Labor Demand Plot


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solve the firm problem through compute_value_func_profit
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[V1, kp,investment,financing,idx] = compute_value_func_profit_eq(n2,n,gamma,delta,R,k,a_star,b0,b1,P,f,1);

% What Shock does a firm need to draw in order to enter the market?
V1_k0    = V1(1,:)>0;
entrance = sum(V1_k0.*d);

% What is the probability of entering with zero capital
gamma = repelem((V1_k0.*d),n2)';


% kp denotes the optimal policy function. We want to obtain the
[Qmat,Tmat]=transition_cs(idx,n2,n,P);

% Get the X matrix
V1_k_pos = V1>0;
X        = repmat(V1_k_pos(:),1,n*n2);

E       = 1;

mu_star = inv(eye(n2*n)-Tmat.*X)*gamma*E;

mu = mu_star./sum(mu_star);

% Re arrange mu
mu_mat=zeros(n2,n);

for j=1:n;
```

```
  %mu_mat(:,n-j+1)=mu((j-1)*n2+1:(j*n2));
  mu_mat(:,j)=mu((j-1)*n2+1:(j*n2));
end

% Compute Exit Rates (For Calibration)
V1_exit = V1<=0;
exit_rate=sum(sum(V1_exit.*mu_mat))

V1_entry = V1>=0;
entry_rate=sum(sum(V1_entry.*mu_mat))

% Labor Demand in Equilibrium
fig_surf_dist(a_grid,k,mu_mat,'output/dist.eps') % Labor Demand Plot

labor_demand=sum(sum((l_star_mat.*mu_mat)));
B=labor_demand/W^(0.1);
```

Main Script Discrete Equilibrium Firms Mass of Firms

```
clc; clear all;
close all;
rng(199)
% FY Paper Heterogeneous Agents Models
% Initialization:

%cd 'C:\Users\anyosae\OneDrive - PennO365\WHARTON\CLASSES\SECOND-YEAR\FALL19\FNCE_937\PSETS\PS1\CODE'
%cd 'D:\OneDrive - PennO365\WHARTON\CLASSES\SECOND-YEAR\FALL19/FNCE_937/PSETS/PS1/CODE'
%cd '/Users/antonyanyosa/OneDrive - PennO365/WHARTON/CLASSES/SECOND-YEAR/FALL19/FNCE_937/PSETS/PS1/CODE'
%cd '/Users/antonyanyosa/Dropbox/FNCE937_PSETS/PS1/Matlab'
cd 'D:\Dropbox\FNCE937_PSETS\PS1\Matlab'

addpath 'output'
addpath 'functions'
addpath 'sample code'
addpath 'Tauchen'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 1: The Problem of the firm in discrete time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Parameters
theta1  = 0.3;
theta2  = 0.65;
delta   = 0.1;
W       = 2;
sigma   = 0.02;
%sigma   = 0.2;
r       = 0.02;
rho     = .95;
b0      = 0;
b1      = 0.5;
ln_abar = 0.5; % Set High Enough so that there is Entry
M       = 0.95;
n2      = 200; % Length of Capital Grid
n       = 19; % States Grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NOW DO THE INCOME PROCESS
% Assume that the log(a) follows: log(a') = (1-rho)ln_abar+rho*log(a) + sigma*eps
% where eps is a truncated standardized normal on [-4,4].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T = 1000;       % Number of periods


%%%
% Approximating AR(1) with a Markov Chain
%%%
```

```
% Parameters for Tauchen approximation:

mu      = ln_abar;
num_std = 4;    % Number of standard deviations from mean

% Simulate 5-point discrete Markov Chain
[a_g,P,d] = tauchen1(n,mu,rho,sigma,num_std);   % Output = [states; transition matrix; stationary dist]
mc           = dtmc(P);                        % declare as a Markov Chain object
a_grid       = exp(a_g);                         % exponentiate to convert log(a) to a
sim_paths    = a_grid(simulate(mc, T));

% Histogram of distribution of technology
%hist_prod(a(500:end),100,'output/productivity.eps')

%%%%%%%%%%%%%%%%%%%%%%%%
% Do Plot of Profit function
%%%%%%%%%%%%%%%%%%%%%%%%

R       = 1/M; % Gross Interest rate
kmin    = 0.01;
kmax    = 4;
k       = linspace(kmin,kmax,n2)';
% CALIBRATED FIXED Costs
f  = 0.07;
a_star      = (a_grid).*(W./(theta2.*a_grid)).^(theta2/(theta2-1))- (W).*(W./(theta2.*a_grid)).^(1/(theta2-1));
gamma       = theta1/(1-theta2);

l_star_mat=zeros(n2,n);

for i=1:n2
  for j=1:n
    l_star_mat(i,j)  = ((W.*k(i).^(-1*theta1))./(theta2*a_grid(j))).^(1/(theta2-1));
  end
end

%fig_surf_ldemand(a_grid,k,l_star_mat,'output/ldemand.eps') % Labor Demand Plot

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solve the firm problem such that markets clear.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[V1, kp,investment,financing,idx] = compute_value_func_profit_eq(n2,n,gamma,delta,R,k,a_star,b0,b1,P,f,1);

% What Shock does a firm need to draw in order to enter the market?
V1_k0    = V1(1,:)>0;
entrance = sum(V1_k0.*d);

% What is the probability of entering with zero capital
gamma = repelem((V1_k0.*d),n2)';

% kp denotes the optimal policy function. We want to obtain the
[Qmat,Tmat]=transition_cs(idx,n2,n,P);

% Get the X matrix
V1_k_pos = V1>0;
X        = repmat(V1_k_pos(:),1,n*n2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% General Equilibrium.
% We want to find THE ENTRY MASS E
% We start with a guess
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Emax        = 4;
Emin        = 0.00001;
E_guess     = .5*(Emax+Emin);   % interest rate
epsil       = 1e-4; % Convergence parameter
AD_labor     = 1; % initialize while loop
```

```
% Store Guess Values and AD
ct=1;
store_vec=zeros(10,2);

tic;

while abs(AD_labor)>epsil

E       = E_guess;

mu_star = inv(eye(n2*n)-Tmat.*X)*gamma*E;

measure_firms=sum(mu_star)
%mu = mu_star./sum(mu_star);
mu = mu_star;
% Re arrange mu
mu_mat=zeros(n2,n);

for j=1:n;
  %mu_mat(:,n-j+1)=mu((j-1)*n2+1:(j*n2));
  mu_mat(:,j)=mu((j-1)*n2+1:(j*n2));
end

% Compute Exit Rates (For Calibration)
V1_exit = V1<=0;
exit_rate=sum(sum(V1_exit.*mu_mat));

% Labor Demand in Equilibrium

labor_demand=sum(sum((l_star_mat.*mu_mat)));
labor_supply=W^(0.1);
excess_demand=(labor_demand-labor_supply); % We want this to be zero.

AD_labor=excess_demand;
disp(strcat("Average Aggregate Excess Demand for L = ", num2str(AD_labor)))


% Bisection Method
if AD_labor>0

    Emax       = E_guess;
    E_guess    = 1/2*(Emin+Emax);

elseif AD_labor<0

    Emin       = E_guess;
    E_guess    = 1/2*(Emin+Emax);

end

fprintf('\n')
disp(strcat("New Guess E = ", num2str(E_guess)))

if abs(E_guess)<0.00001
    disp('No Convergence, Corner Solution')
    av_AD_k=0;
end


if abs(AD_labor)<epsil
    fprintf('************* \n' )
    disp(strcat("Converged at E = ", num2str(E_guess)))
    E_star=E_guess;
    toc
    fprintf('************* \n' )
```

```
end

end

fig_surf_dist(a_grid,k,mu_mat,'output/dist_EQ.eps') % Labor Demand Plot
```