ECON 937 Assignment I

Paweł Bednarek & Rodrigo Morales Mendoza

14 October 2019

Question 1

1.a)
To solve this problem, the first determined is the grid for productivity, and its transition matrix is given by the Tauchen method.
Then the optimal labor is found as a function of capital, which is given by the function:

$$l(k_t) = \left[\frac{\theta_2 a_t k_t^{\theta_1}}{w}\right]^{\frac{1}{1-\theta_2}},$$

which follows from the FOC of the profits.
The steady state capital is determined numerically, solving the Euler condition for the investment equal to δK_{ss}, which also fixes the investment cost equal to $b_0 K_{ss}$.
Plot of the profit function of the capital stock, highest and lowest value of productivity shock:

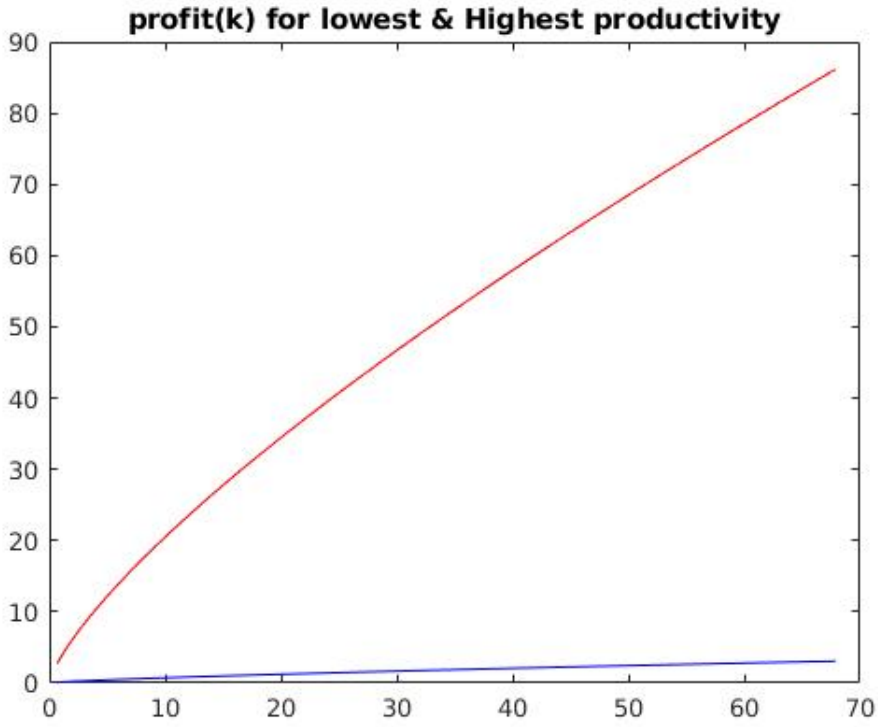
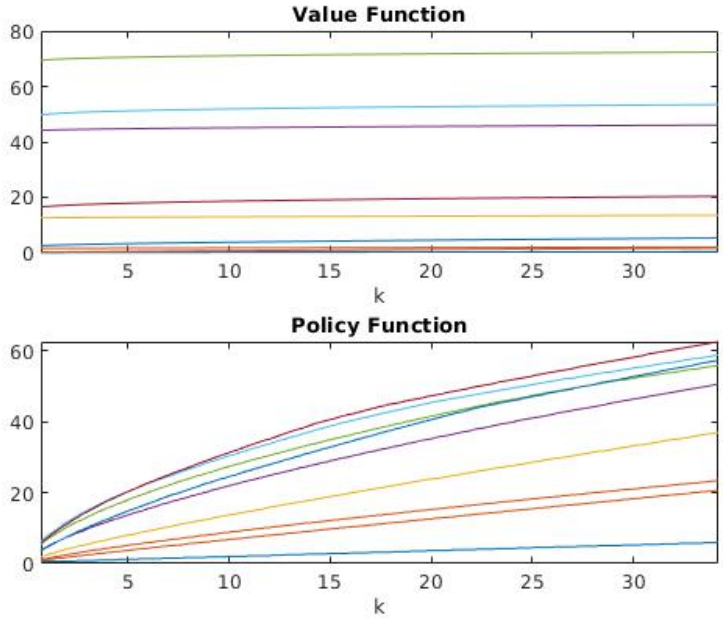
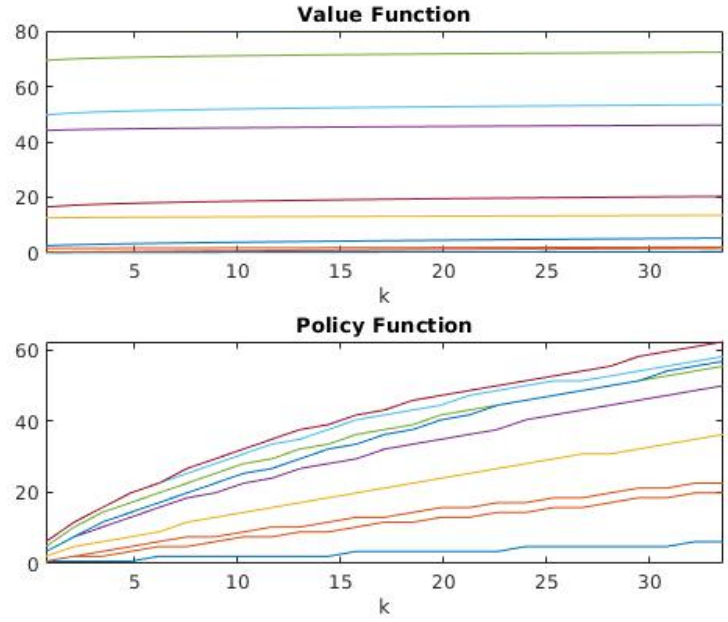


Figure 1: Exercise 1.a).



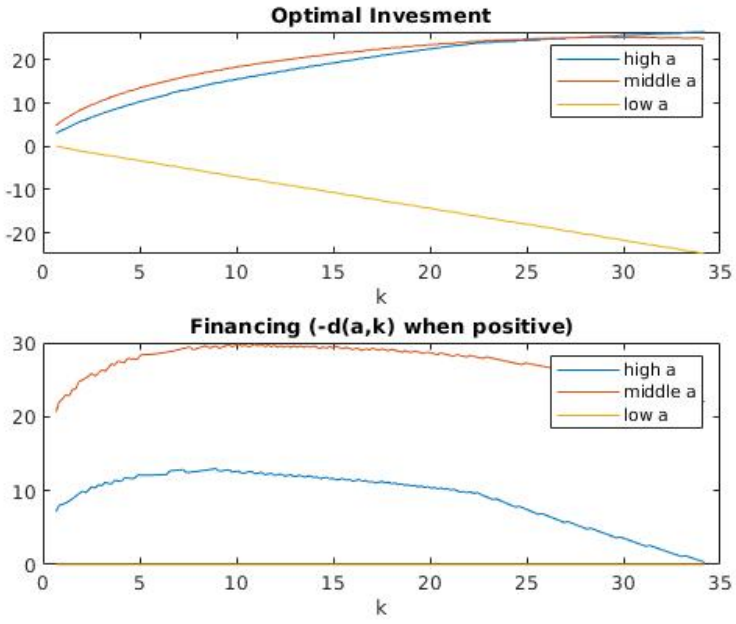
(a) 01.b) Grid size 200.



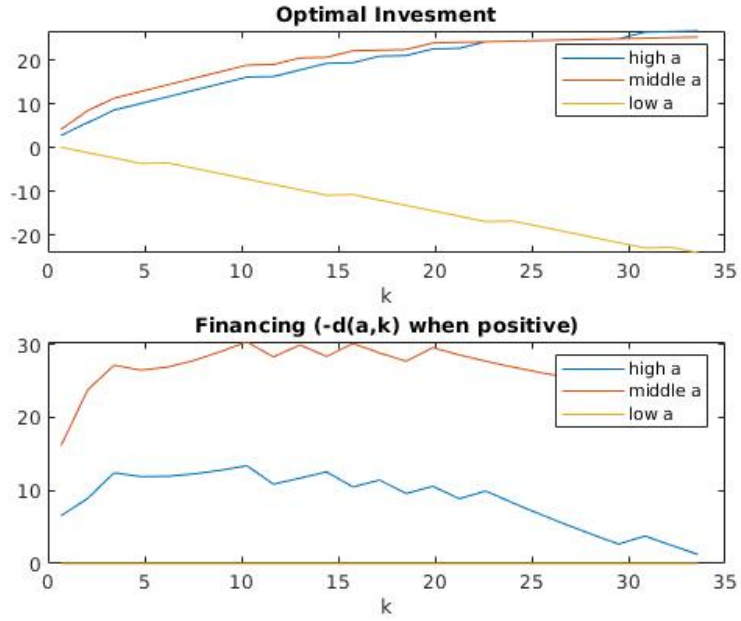
(b) 01.b) Grid size 25.

Figure 2: Exercise 1.b).

1.b) Value function iteration. Plot of the value of the firm against stock capital, using grids of 200 and 25.
1.c) Plot of the investment and financing policies against capital (highest, average and lowest productivity), using grids of 200 and 25
1.d) Plot of the investment at the mean shock only, against capital, using a grid of 200:

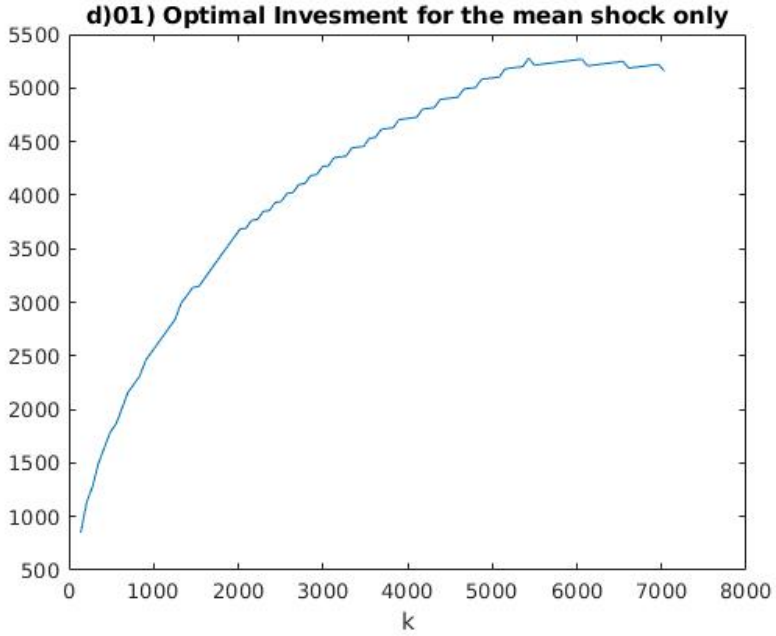


(a) 01.c) Grid size 200.

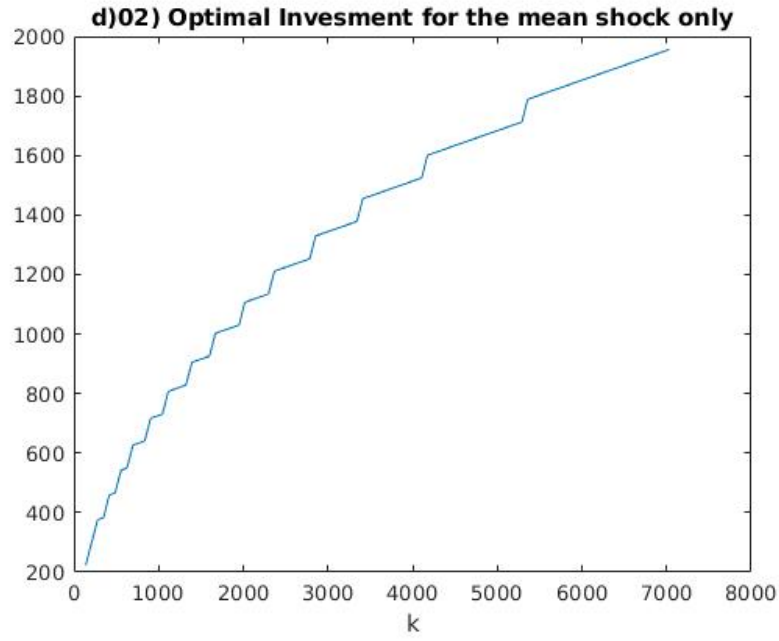


(b) 01.c) Grid size 25.

Figure 3: Exercise 1.c).

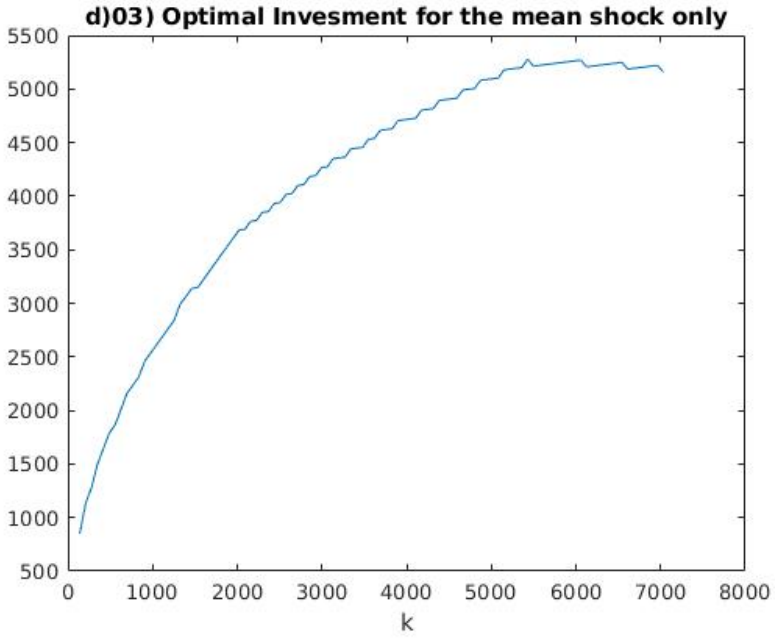


(a) 01.d.1) b0 = 0, b1 = 0.5.

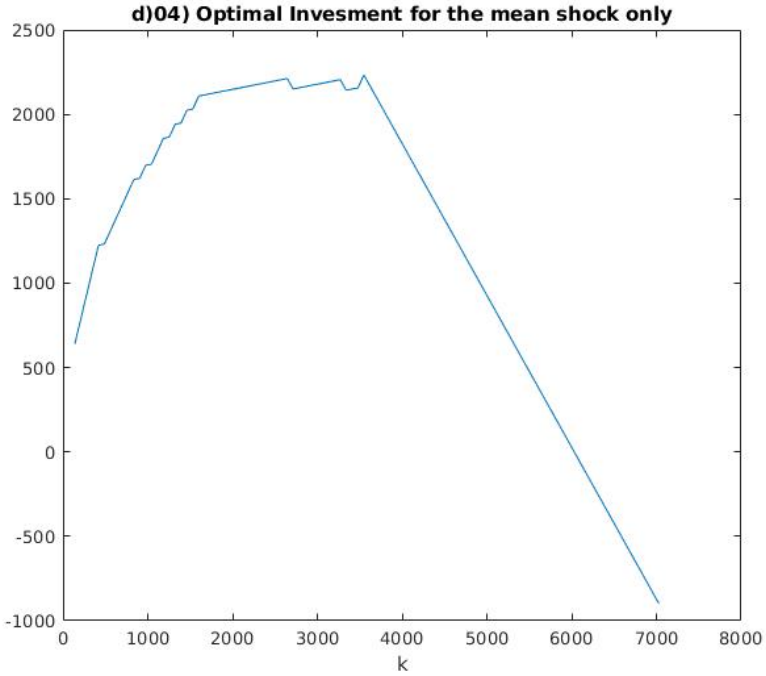


(b) 01.d.2) b0 = 0, b1 = 10.

Figure 4: Exericse 1.d.1-2))



(a) 01.d.3) b0 = 0, b1+ = 0.5, b1- = 10.



(b) 01.d.4) b0 = 0.02, b1 = 0.5, unless SS.

Figure 5: Exericse 1.d.3-4))

Question 2

2.a) No Uncertainty.
The HJB equation is:

$$rV(k) = \max_i \{\bar{a}k^\alpha - i - \Phi(i,k) + V'(k')(i - \delta k)\},$$

which results in a FOC:

$$V'(k) = 1 + \phi_i(i/k).$$

Following the slides of the class, as well as Moll's code for such model, we use the following pseudo-code:

1. Use $v_0(a,k) = ak^\alpha$ as an initial guess.
2. Compute the optimal policy i_n^*, V_k^n, with the upwind approach.
3. Solve the equation:
$$\frac{V_{n+1} - V_n}{\Delta V} + rV^{n+1} = d^n + BV^{n+1}$$
4. Stop when $||V_{n+1} - V_n||$ is small.

Implementing such algorithm in Matlab, results in the following plot of the resulting policies (investment and drift of capital), with productivity = 1.

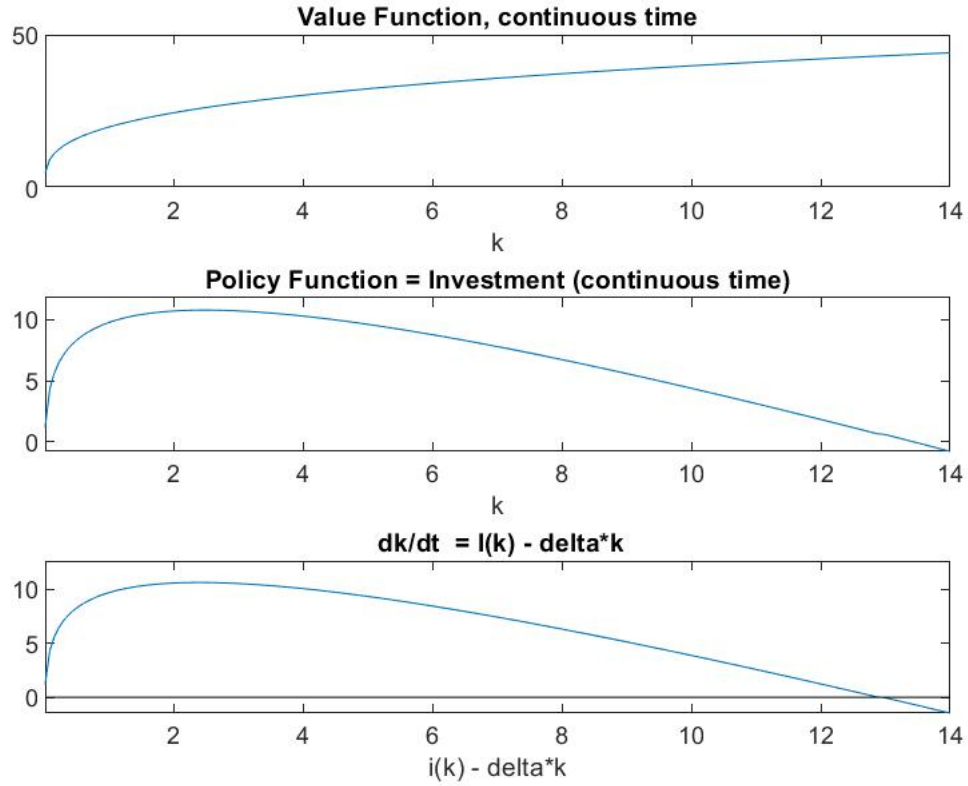


Figure 6: Exercise 2.a).

2.b) Poisson uncertainty.
We now have that the HJB equation looks as follows:

$$rV(a,k) = \max_i \{d(a,i,k) + V_k(a,k)(i - \delta k) + P(:|a) * V(:,k) - V(a,k)\}.$$

Note that in this case, we have the term $P(:|a) * V(:,k) - V(a,k) = \sum_{a'} P(a'|a)V(a',k) - V(a,k)$, which adds an extra layer to the difficulty of the calculation of matrix B. The upwind scheme will basically remain intact, but these new terms will basically make us add the transition probabilities instead of zeroes in the lower left and upper right sides of B's terms.
Implementing this new algorithm, results in the following plot of the policies with the grid of productivity:

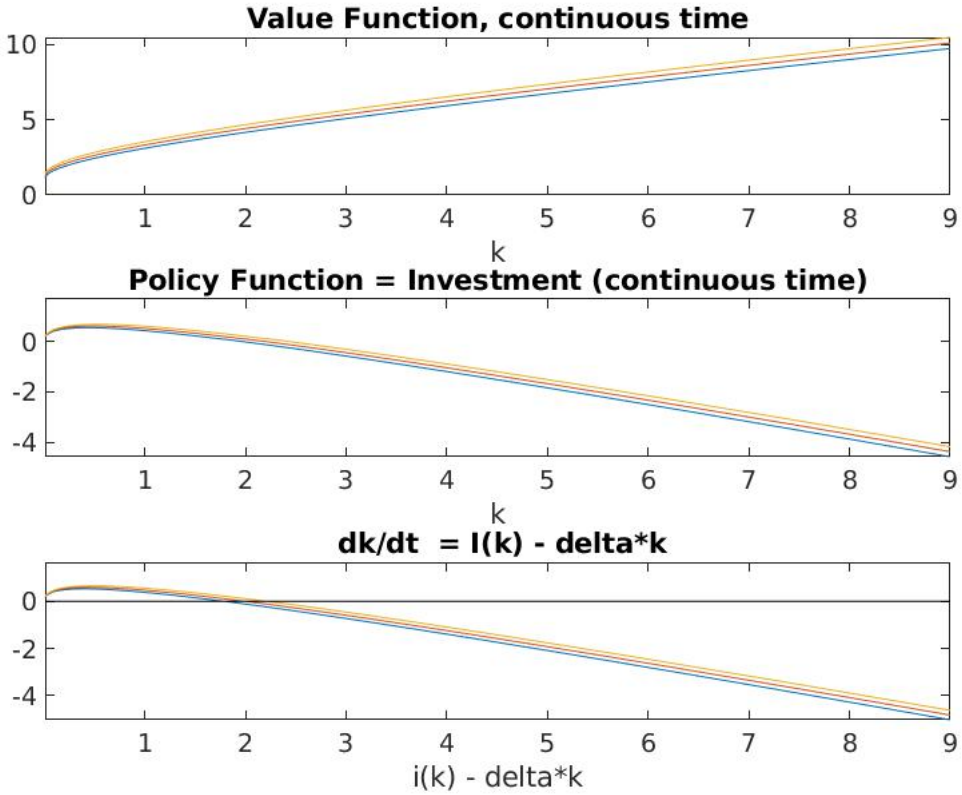


Figure 7: Exercise 2.b).

Question 3

We consider a dynamic problem of a firm, where the value function is expressed as

$$v(z,k) = \max_{k'} \Big(d(z,k,k') + M \max(\mathbb{E}_z v(z',k'), 0) \Big)$$

. The value of gross distributions is given by

$$d(z,k,k') = \pi(\,k) - \phi(k,k')$$

and the adjustment costs

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

We notice that this formulation does not allow us to solve the problem because new entrants with zero capital would face infinite costs of investment due to the division by k in the function $\phi(k, k')$. There is a few ways to overcome this issue but we choose to replace the adjustment costs function with

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

which prevents us from division by zero. One could also assume that new entrants enter with a tiny amount of capital allowing them to invest at non-infinite cost.
Furthermore, the operating profits obey

$$\pi = zk^{\alpha_k} l^{\alpha_l} - Wl - f.$$

We notice that the labor decision depends on the current values of state variables, and after solving the maximization problem, we obtain

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

The mean of the shocks is chosen to be 1.5; we create a grid of k from 0 to 2 with 0.01 increments and discretize the continuous stochastic process with the Tauchen method.

```
# Initiate parameters.
W <- 2
alpha_k <- 0.3
alpha_l <- 0.65
M <- 0.95
delta <- 0.1
ro <- 0.95
sigma <- 0.02
# Grid for capital (k).
gridk <- seq(0,2,by=0.01)
nk <- length(gridk)
# Grid for shocks (z).
z_bar <- 1.5
nz <- 9
m <- (nz-1)/2
z_1 <- z_bar-m*sigma/(sqrt(1-ro^2))
z_na <- z_bar+m*sigma/(sqrt(1-ro^2))
delta_z <- (z_na-z_1)/8
gridz <- seq(z_1,z_nz,length.out=9)
```

We proceed with formulating an algorithm performing value function iteration. Here, we solve for a particular value of f that is equal to 0.01. It will be soon callibrated to match the desired exit rate of 2.5%.

```
# Create matrices for value and policy functions - initialize with zeros.
final <- matrix(0,nz,nk)
final_k <- matrix(0,nz,nk)
# Pick a value of f for the first test run.
f <- 0.01
```

```
# Iterate over all possible shocks in the grid.
for(zi in 1:nz){
  z <- gridz[zi]
  # Initiate the value and policy functions.
  if(zi==1){
    vfun <- c(-f,rep(-100, nk-1))
    gfun <- c(0,rep(-100, nk-1))
    gfun0 <- c(0,rep(-100, nk-1))
    vfun0 <- c(-f,rep(-100, nk-1))
  }else{
    vfun <- final[zi-1,]
    gfun <- final_k[zi-1,]
    vfun0 <- final[zi-1,]
  }
  # Iterate the value function until convergence
  it=1
  diff=10^10
  while(it<=maxit & diff>tol){
    it=it+1
    for(kc in 1:nk){
      k0 <- gridk[kc]
      vfun[kc] <- -10000
      for(kcc in 1:nk){
        k0 <- gridk[kc]
        k <- gridk[kcc]
        if(k0==0){
          l <- 0
        }else{
          l <- (W/(alpha_l*z*k0^alpha_k))^(1/(alpha_l-1))
        }
        pi <- z*k0^alpha_k*l^alpha_l - W*l - f
        i <- k - (1-delta)*k0
        if(k == 0){
          phi <- 0
        }else{
          phi <- k - (1-delta)*k0  + 0.5*(i/k-delta)^2*k
        }
        d <- pi - phi
        vhelp=d+M*vfun0[kcc]
        if(vhelp>=vfun[kc]){
          optimal_k_prime <- kcc
          vfun[kc]=vhelp
          gfun[kc]=gridk[kcc]
        }
      }
      # If the expectation of v(z',k')<0 then exit the market.
      if(M*vfun0[optimal_k_prime]<0){
        if(k0==0){
          l <- 0
        }else{
          l <- (W/(alpha_l*z*k0^alpha_k))^(1/(alpha_l-1))
        }
        pi <- z*k0^alpha_k*l^alpha_l - W*l - f
        i <- 0
        d <- pi
        vfun[kc] <- d
        gfun[kc] <- 0
      }
    }
```

```
    diff=max(abs(vfun-vfun0))
    vfun0 <- vfun # Update the previous value of V.
  }
  # Save the value and policy functions.
  final[zi,] <- vfun
  final_k[zi,] <- gfun
}
```

We now create the matrices Q and T in order to obtain the joint distribution of firms with respect to both the productivity and capital.

```
# Obtaining matrix Q.
Q <- matrix(0,nk,nk*nz)
for(i in 1:nk){
  for(iz in 1:nz){
    Q[which(gridk==final_k[iz,i]),(i-1)*nz+iz] <- 1
  }
}
# Obtaining matrix T (denoted as TT in order to avoid computational problems as T
    in R stands for Boolean True).
TT <- matrix(0,nk*nz,nk*nz)
for(i in 1:(nk)){
  for(k in 1:nz){
    for(j in 1:(nk)){
      for(l in 1:nz){
        TT[(i-1)*nz+k,(j-1)*nz+l] <- Q[i,(j-1)*nz+l]*P[k,l]
      }
    }
  }
}
# Obtaining the exit matrix X.
X <- matrix(NA,nz,nk)
for(i in 1:9){
  for(j in 1:nk){
    if(final_k[i,j]==0){
      X[i,j] <- 0
    }else{
      X[i,j] <- 1
    }
  }
}
X <- t(X)
X <- matrix(NA,nk*nz,nk*nz)
X_vec <- c()
for(i in 1:nk){
  for(j in 1:nz){
    X_vec[(i-1)*nz+j] <- X[i,j]
  }
}
for(i in 1:(nk*nz)){
  X[i,] <- X_vec
}
# Saving as sparse matrices and obtaining
library(Matrix)
X <- Matrix(X, sparse=T)
TT <- Matrix(TT, sparse=T)
```

What we have to do now is to calculate $\gamma(z, k = 0)$, which is the distribution of firms entering the market. For this purpose, we need to first find the long-run distribution of z and hence we proceed as follows.

```
# We apply the Tauchen discretization.
P <- matrix(0,nz,nz)
for(i in 1:nz){
  for(j in 1:nz){
    P[i,j] <- pnorm((gridz[i]+0.5*delta_z-(1-ro)*z_bar-ro*gridz[j])/sigma) -
      pnorm((gridz[i]-0.5*delta_z-(1-ro)*z_bar-ro*gridz[j])/sigma)
  }
}
# We apply the boundary conditions in order to normalize the distribution such as
    it sums up to 1.
for(j in 1:nz){
  P[1,j] <- pnorm((gridz[1]+0.5*delta_a-(1-ro)*a_bar-ro*gridz[j])/sigma)
  P[nz,j] <- 1-pnorm((gridz[nz]-0.5*delta_a-(1-ro)*a_bar-ro*gridz[j])/sigma)
}
P <- t(P)
# We raise P to a high power in order to find the stationary distribution.
P_star <- (PP%^%10000)
stationary_dist_z <- P_star[1,]
```

We proceed with finding the distribution of entering firms and use the formula

$$\mu(z, k) = (I - T \cdot X)^{-1}\gamma(z, 0)$$

to find the joint stationary distribution of firms.

```
gamma_0 <- rep(0, nk*nz)
gamma_0[1:nz] <- stationary_dist_z
mu <- solve(diag(ncol(XX))-TT*XX)%*%(gamma_0)
```

Finally, we find the constant E that denotes the mass of entering firms so that the joint distribution μ sums up to 1; and we multiply our distribution by its value afterwards.

```
E <- 1/sum(goal)
mu <- mu*E
# We calculate the exit rate.
t(mu)%*%(1-X_vec)
[1] 0.129525
```

Therefore, we found here that the exit rate is equal to 0.1295 or 12.95%. Our target rate is 2.5%, so we create a grid for f in order achieve our goal (The procedure is shown in the Appendix C). We find that f supporting this rate is approximately equal to 0.0049. We repeat the procedure described above for $f = 0.0049$ and obtain the joint distribution of firms.

```
capital <- c()
productivity <- c()
density <- c()
for(i in 1:nk){
  for(j in 1:nz){
    capital[(i-1)*nz+j] <- gridk[i]
    productivity[(i-1)*nz+j] <- gridz[j]
    density[(i-1)*nz+j] <- density[(i-1)*nz+j]
```

```
  }
}
library(lattice)
wireframe(density ~ capital*productivity,screen = list(z = -60, x = -60), xlab="
    capital", ylab="productivity", zlab="density", scales = list(x=list(at=seq(0,
    2, by=0.5), labels=seq(0, 2, by=0.5)), arrows=FALSE, col="black", font=3, tck
    =1))
```

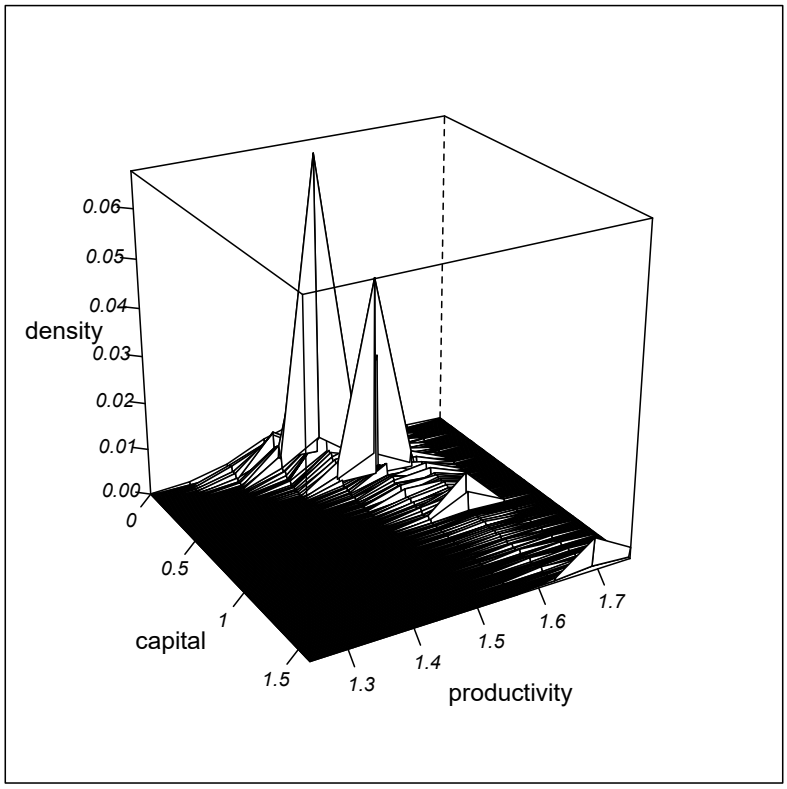


Figure 8: Stationary distribution of firms (with measure 1).

We now obtain the value of B that is consistent with the competitive equilibrium given the aggregate labor supply $L^S = BW^{0.1}$.

```
labour_vec <- c()
for(i in 1:length(capital)){
  z <- productivity[i]
  k <- capital[i]
  # Obtain the labour decision and scale by density.
  labour_vec[i] <- (W/(alpha_l*z*k^alpha_k))^(1/(alpha_l-1)) * density[i]
}
labour_supply <- sum(labour_vec)
B <- labour_supply/W^(0.1)
B
[1] 0.0576892
```

Therefore, we obtain $B = 0.0577$ and the labor supply $L^S = 0.0618$. Also, the entry that is given by E is equal precisely to the exit rate (2.5%), which is consistent with this being a stationary

distribution.
We now suppose that $B = 1$, and we must find the mass of firms that is consistent with the new labor supply. We calculate the new labor supply as follows.

$$\hat{L}^S = BW^{0.1} = 1 \cdot 2^{0.1} \approx 1.0718$$

To find the new mass of firms that is consistent with this supply, we simply need to scale our distribution of firms by a factor tat scales the labor supply. In particular, we obtain

$$\hat{L}^S / L^S \approx 17.334,$$

which is precisely the mass of firms under $B = 1$.

Question 4

4.a) Analytical expression for the value function after exercising option.
The value function after exercise of option at time 0 (time when exercised is irrelevant as time is infinite) is:

$$E\left[\int_0^\infty e^{-rs} z(s) k'^{\alpha} ds\right] = \frac{k'^{\alpha}}{\mu - r},$$

4.b)Analytical expression for the value function before exercising option.
The profits obey

$$\pi(z, \hat{k}) = z\hat{k}^{\alpha}, \quad \hat{k} \in \{k, k'\}$$

with investment costs

$$\phi = b_0 + b_1(k' - k).$$

We also have a differential equation describing the evolution of productivity shocks, which is given by

$$dz = \mu z dt + \sigma z dW.$$

We start with calculating the expected value of infinite-horizon profits irrespective of the investment decision. We formulate the HJB equation for that problem, which is given by

$$rV(z) = zk^{\alpha}\mu V_z(z)z + \frac{\sigma^2}{2} V_{zz}(z)z^2.$$

Hamiltonian to this problem yields

$$V(z) = \frac{zk^{\alpha}}{r - \mu} + A_1 z^{\eta_1} k^{\alpha} + A_2 z^{\eta_2} k^{\alpha}.$$

As usual, boundary conditions yield $A_2 = 0$, so it pins down to:

$$V(z) = \frac{zk^{\alpha}}{r - \mu} + A_1 z^{\eta_1} k^{\alpha}.$$

4.c) Optimal investment boundary and optimal amount of investment.
Note that we are unable to compute the decision boundary when $\mu = r$ because profits in both cases diverge to infinity. Therefore, we choose $\mu = 0.05$ and $r = 0.10$.
Then essentially, given a productivity realization z(t), the problem of the firm consists in comparing two payoffs, the one it gets when exercising the investment option, given by part a), but including

the payoff of ϕ it would cost, or not exercising and remaining with the continuation value, leaving the decision for a future time, with the last expression found.

We perform the following analysis in order to find the regions in which the firm would exercise its investment decision.

We present the decision boundary for the grid of $z \in [0, 1]$ and $k \in [0, 10]$. We pick three different values of k' to illustrate the decision boundary under different scenarios, and show the regions in which the firm actually decides to invest vs those where the firm remains as it is.

```
z_vec <- seq(0.001,1,by=0.001)
k_vec <- seq(0.01,10,by=0.01)
kappa <- 5
alpha <- 0.9
r <- 0.1
mu <- 0.05
sigma <- 0.02
b0 <- 0.1
b1 <- 1.2
MAT <- matrix(NA, length(z_vec), length(k_vec))
for(i in 1:length(z_vec)){
  for(j in 1:length(k_vec)){
    zeta <- z_vec[i]
    k0 <- k_vec[j]
    V_invest <- zeta*kappa^alpha/(r-mu)-b0-b1*abs(kappa-k0)
    V_notinvest <- zeta*k0^alpha/(r-mu)
    if(V_invest>V_notinvest){
      MAT[i,j] <- 1
    }else{
      MAT[i,j] <- 0
    }
  }
}
y.scale <- list(at=seq(1,1000,length.out=11), labels=seq(0,10,by=1))
x.scale <- list(at=seq(1,1000,length.out=11), labels=seq(0,1,by=0.1))
par(mfrow=c(1,2))
levelplot(MAT, main="k' = 5", ylab="capital", xlab="productivity", scales=list(x=x
    .scale, y=y.scale), colorkey=FALSE)
```

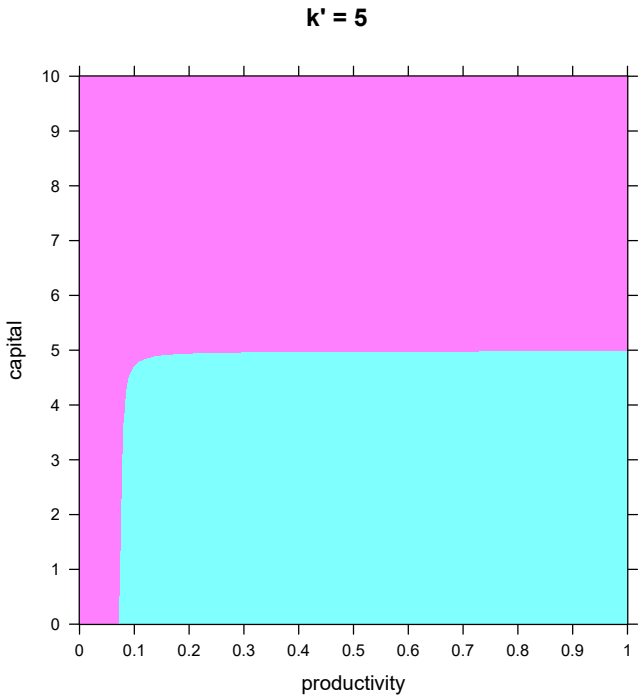


Figure 9: Decision boundary for $k' = 5$ - Blue represents exercising the option.

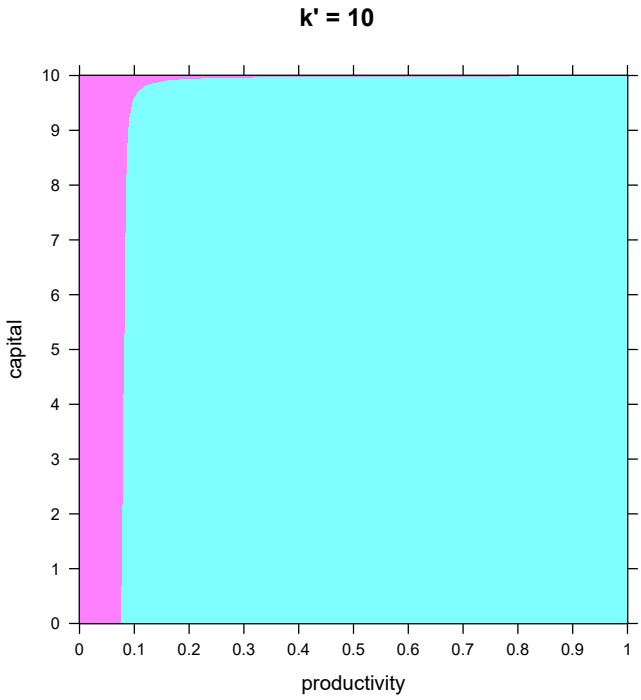


Figure 10: Decision boundary for $k' = 10$ - Blue represents exercising the option.

Appendix A

MATLAB code for Question 1.

```
%% 0. Housekeeping

clear all
close all
clc

tic

%%  1. Calibration
clear
clc

rho = 0.8;   % ar(1) parameter of log(a) (productivity)
r = 0.02;    % 1/(1+r) discount rate of firm
bbeta = 1/(1+r);   % Discount factor of the firm
delta = 0.1; % depreciation
theta1 = 0.3; % kapital elasticity (cobbdouglas)
aalpha =theta1;     % Elasticity of output w.r.t. capital
theta2 = 0.6; % labor elasticity (cobbdouglas)
W = 2;   % wage
sigma = 0.1;  % std dev of eps (ar(1)) of log(a) (productivity)
abar = 3.1; % log(abar) is the mean of log(a), which is ar(1)
b0 = 0;    %parameter0 for cots of investing
b1 = 0.5;   %parameter1 for cots of investing
options = optimset('Display', 'off');  %when solves for kss

% Productivity values
na = 9;  %number of points for logaGrid = [loga0, loga1... loga9]
logabar = log(abar);
deltaLoga = sigma/sqrt(1-rho^2);  %change between each log(a);

vProductivity = (logabar - deltaLoga*floor(na/2)):deltaLoga:(logabar + deltaLoga*
    floor(na/2));

% Transition matrix
P   = eye(na);
a_1 = vProductivity(1);
a_n = vProductivity(na);
for j = 1:na
    aj = vProductivity(j);
    upperBoundA = (a_1 - rho*aj - (1-rho)*logabar +deltaLoga/2)/sigma;
    P(1,j) = normcdf(upperBoundA);
    lowerboundA = (a_n - rho*aj - (1-rho)*logabar -deltaLoga/2)/sigma;
    P(na,j) = 1-normcdf(lowerboundA);
end
for i = 2:(na-1)
    for j = 1:(na)
        ai = vProductivity(i);
        aj = vProductivity(j);
        upperBoundA = (ai - rho*aj - (1-rho)*logabar +deltaLoga/2)/sigma;
        lowerboundA = (ai - rho*aj - (1-rho)*logabar -deltaLoga/2)/sigma;
        P(i,j) = normcdf(upperBoundA)-normcdf(lowerboundA);
    end
end
P'
mTransition␣␣␣=␣P;


```

```
% 2. Steady State
%define the functions of labor, profit and investment...
labor = @(a,k) (theta1*(k.^theta1)'*a/W).^(1/(1-theta2));
profit = @(a,k) ((k'.^theta1)*a).*(labor(a,k).^theta2) -W*labor(a,k);
investment = @(a,k,kprime) kprime -(1-delta)*k'*ones(size(a,1));
phi = @(a,k,kprime) b0 * k'*ones(1,size(a,1)) +b1*(( investment(a,k,kprime)./ (k
    '*ones(1,size(a,1)) )- delta).^2).*k'*ones(1,size(a,1));

%find the values of the steady state:
fss =@(a,kss) delta +b0 -(theta1*a*kss^(theta1-1)*labor(a,kss)^theta2)/(1+r);
[capitalSteadyState,fval] =fsolve(@(k)fss(abar,k),30,options);
outputSteadyState =abar*capitalSteadyState^theta1*labor(abar,capitalSteadyState)^
    theta2;

fprintf(' Output = %2.6f, Capital = %2.6f\n', outputSteadyState, 
    capitalSteadyState);
fprintf('\n')

% with kss, generate the grid of capital
capMiddle =(aalpha*abar/(r+delta))^(1/(1-aalpha));
kstep =0.1; %0.00001
%vGridCapital =0.5*capitalSteadyState:kstep:1.5*capitalSteadyState;
vGridCapital =0.5*capMiddle:kstep:1.5*capMiddle;
kapitalMax =(abar/(delta))^(1/(1-aalpha));
vGridCapital =(0):kstep:(0.6*kapitalMax);
vGridCapital =(0):kstep:(capitalSteadyState);
vGridCapital =linspace(0.01*capitalSteadyState,capitalSteadyState,200);




nGridCapital =length(vGridCapital);
nGridProductivity =length(vProductivity);

%% Plot of the profit function

figure(1)
plot(vGridCapital, profit(exp(vProductivity(1)),vGridCapital),'-b')
hold on
plot(vGridCapital, profit(exp(vProductivity(nGridProductivity)),vGridCapital),'-r'
    )
hold off
title('profit(k) for lowest & Highest productivity')

%% 3. Required matrices and vectors

mOutput           =zeros(nGridCapital,nGridProductivity);
mValueFunction    =zeros(nGridCapital,nGridProductivity);
mValueFunctionNew =zeros(nGridCapital,nGridProductivity);
mPolicyFunction   =zeros(nGridCapital,nGridProductivity);
expectedValueFunction =zeros(nGridCapital,nGridProductivity);

%% 4. We pre-build output for each point in the grid

profitMatrix =profit(exp(vProductivity),vGridCapital);

%% 5. Main iteration

maxDifference =10.0;
tolerance =0.00001;%0.0000001;
```

```
iteration = 0;

while (maxDifference>tolerance)

    expectedValueFunction = mValueFunction*mTransition';

    for nProductivity = 1:nGridProductivity

        % We start from previous choice (monotonicity of policy function)
        gridCapitalNextPeriod = 1;

        for nCapital = 1:nGridCapital

            valueHighSoFar = -1000.0;
            capitalChoice  = vGridCapital(1);

            for nCapitalNextPeriod = gridCapitalNextPeriod:nGridCapital

                %consumption = mOutput(nCapital,nProductivity)-vGridCapital(
                    nCapitalNextPeriod);
                prodctvt = exp(vProductivity(nProductivity));
                currentCapital = vGridCapital(nCapital);
                kprimeTomorow = vGridCapital(nCapitalNextPeriod);
                dividend = profitMatrix(nCapital,nProductivity) - investment(
                    prodctvt,currentCapital,kprimeTomorow) - phi(prodctvt,
                    currentCapital,kprimeTomorow);
                valueProvisional = (1-bbeta)*dividend+bbeta*expectedValueFunction(
                    nCapitalNextPeriod,nProductivity);

                if (valueProvisional>valueHighSoFar)
                    valueHighSoFar = valueProvisional;
                    capitalChoice = vGridCapital(nCapitalNextPeriod);
                    gridCapitalNextPeriod = nCapitalNextPeriod;
                else
                    break; % We break when we have achieved the max
                end

            end

            mValueFunctionNew(nCapital,nProductivity) = valueHighSoFar;
            mPolicyFunction(nCapital,nProductivity) = capitalChoice;

        end

    end

    maxDifference = max(max(abs(mValueFunctionNew-mValueFunction)));
    mValueFunction = mValueFunctionNew;

    iteration = iteration+1;
    if (mod(iteration,10)==0 || iteration ==1)
        fprintf(' Iteration = %d, Sup Diff = %2.8f\n', iteration, maxDifference);
    end

end

fprintf(' Iteration = %d, Sup Diff = %2.8f\n', iteration, maxDifference);
fprintf('\n')
%fprintf(' My check = %2.6f\n', mPolicyFunction(1000,3));
```

```
fprintf('\n')

toc

%% 6. Plotting results

figure(2)

subplot(2,1,1)
xkap = vGridCapital(1:nGridCapital/2);
plot(xkap,mValueFunction(1:nGridCapital/2,:))
xlim([vGridCapital(1) vGridCapital(nGridCapital/2)])
xlabel('k')
title('Value Function')

subplot(2,1,2)
plot(xkap,mPolicyFunction(1:nGridCapital/2,:))
xlim([vGridCapital(1) vGridCapital(nGridCapital/2)])
xlabel('k')
title('Policy Function')

% vExactPolicyFunction = aalpha*bbeta.*(vGridCapital.^aalpha);
%
% subplot(3,1,3)
% plot((100.*(vExactPolicyFunction'-mPolicyFunction(:,3))./mPolicyFunction(:,3)))
% title('Comparison of Exact and Approximated Policy Function')

%set(gcf,'PaperOrientation','landscape','PaperPosition',...
%[-0.9 -0.5 12.75 9])
%print('-dpdf','Figure1.pdf')

%% Plotting optimal investment and financing for lowest, avg and highest
mOptimalInvestment = zeros(nGridProductivity, nGridCapital);
mFinancing = zeros(nGridProductivity, nGridCapital);
for nProductivity = 1:nGridProductivity
    for nCapital = 1:nGridCapital
        prodctvt = exp(vProductivity(nProductivity));
        currentCapital = vGridCapital(nCapital);
        captomorrow = mPolicyFunction(nCapital,nProductivity);
        invstmnt = investment(prodctvt,currentCapital,captomorrow);
        mOptimalInvestment(nCapital,nProductivity) = invstmnt;
        profits = profitMatrix(nCapital,nProductivity);
        costOfInvstmnt = phi(prodctvt,currentCapital,captomorrow);
        dividend = profits - invstmnt - costOfInvstmnt;
        mFinancing(nCapital,nProductivity) = max(-dividend,0);
    end
end
%shock a
figure(3)
subplot(2,1,1)
xkap = vGridCapital(1:nGridCapital/2);
plot(xkap,mOptimalInvestment(1:nGridCapital/2,8))
xlabel('k')
title('Optimal Invesment')
hold on
plot(xkap,mOptimalInvestment(1:nGridCapital/2,5))
plot(xkap,mOptimalInvestment(1:nGridCapital/2,1))
legend('high a', 'middle a', 'low a')
hold off
```

```
227
228 subplot(2,1,2)
229 plot(xkap,mFinancing(1:nGridCapital/2,8))
230 xlabel('k')
231 title('Financing (-d(a,k) when positive)')
232 hold␣on
233 plot(xkap,mFinancing(1:nGridCapital/2,5))
234 plot(xkap,mFinancing(1:nGridCapital/2,1))
235 legend('high a',␣'middle a',␣'low a')
236 hold␣off
237
238 figure(4)
239 plot(xkap,mOptimalInvestment(1:nGridCapital/2,8))
240 xlabel('k')
241 title('Optimal Invesment')
242 hold␣on
243 plot(xkap,mOptimalInvestment(1:nGridCapital/2,5))
244 plot(xkap,mOptimalInvestment(1:nGridCapital/2,1))
245 legend('high a',␣'middle a',␣'low a')
246 hold␣off
```

Appendix B

MATLAB code for Question 2.

```
1  %% 0. Housekeeping
2  clear all
3  close all
4  clc
5  tic
6  %%  1. Calibration
7  aalpha = 0.3;   % Elasticity of output w.r.t. capital
8  bb = 0.5;       % cost of investment
9  delta = 0.05;  % start with a very small one
10 r = 0.05;       % interest rate, return of capital
11 % Productivity values
12 vProductivity = [0.9 1.0 1.1];
13 % vProductivity = 1;
14 nGridProductivity = length(vProductivity);
15 na = nGridProductivity;  %number of points of Productivyt
16 if na >1
17     mTransition = [1/2 1/2 0; 1/4 1/2 1/4; 0 1/2 1/2]';
18 else
19 ␣␣␣␣mTransition␣=␣1;
20 end
21
22 %%␣2.␣Steady␣State
23 %define␣the␣functions␣of␣labor,␣profit␣and␣investment...
24 abar␣=␣1.0;
25 capitalSteadyState␣=␣(aalpha*1/(r+delta))^(1/(1-aalpha));
26 outputSteadyState␣=␣abar*capitalSteadyState^aalpha;
27 %fprintf(' Output = %2.6f, Capital = %2.6f\n',␣outputSteadyState,␣
       capitalSteadyState);
28 fprintf('Code Began running...\n')
29 %␣Generate␣the␣grid␣of␣capital
30 kapitalMax␣=␣(abar/(delta))^(1/(1-aalpha));
31 kmin␣=␣0.01;
```

```
kmax = 10;
vGridCapital = linspace(kmin,kmax,200);
% Get sizes of capital and prodctvt grid:
nGridCapital = length(vGridCapital);
nGridProductivity = length(vProductivity);
I = nGridCapital;
dk = (kmax-kmin)/(I-1);

%% 3. Required matrices and vectors
%mValueFunction     = linspace(1,vGridCapital(nGridCapital),nGridCapital)'*
    vProductivity;
mValueFunction    = (vGridCapital'.^aalpha)*vProductivity;
%mValueFunction(1,:) = 6*ones(1,nGridProductivity);
%mValueFunction(end,:) = (vGridCapital(end)^aalpha./r)*vProductivity;
%mValueFunctionNew = zeros(nGridCapital,nGridProductivity);
mValueFunctionNew = mValueFunction;
mPolicyFunction    = zeros(nGridCapital,nGridProductivity); %investment
% Matrices for the Implicit iteration in each productivity:
mdVf = zeros(nGridCapital,nGridProductivity);
mdVb = zeros(nGridCapital,nGridProductivity);
%% 4. We pre-build output for each point in the grid
%profitMatrix = profit(exp(vProductivity),vGridCapital);
%% 5. Main iteration
maxDifference = 10.0;
maxite = 100000;
tolerance = 0.00099;%0.0000001;
iteration = 0;
deltaV = 0.00001;
explicit = 1; % 1 - to run the explicit, 2 - implicit

while (maxDifference>tolerance || iteration > maxite)
%      iteration
%      maxDifference
    for nProductivity = 1:nGridProductivity
        dVf = mdVf(:,nProductivity);
        dVb = mdVb(:,nProductivity);
        prodctvt = vProductivity(nProductivity);
        V = mValueFunction(:,nProductivity);
        % forward difference
        dVf(1:I-1) = (V(2:I)-V(1:I-1))/dk;
        %dVf(I) = (aalpha*prodctvt.*kmax.^(aalpha-1) - delta)/r; %state constraint
            , for stability
        dVf(I) = 1;
        % backward difference
        dVb(2:I) = (V(2:I)-V(1:I-1))/dk;
        %dVb(1) = (aalpha*prodctvt.*kmin.^(aalpha-1) - delta)/r; %state constraint
            , for stability
        dVb(1) = 1; %state constraint, for stability
        %investment with forward difference
        investmentVectorF = ((dVf-1)/bb + delta).*vGridCapital';
        muf = investmentVectorF - delta*vGridCapital';
        %investment with backward difference
        investmentVectorb = ((dVb-1)/bb + delta).*vGridCapital';
        mub = investmentVectorb - delta*vGridCapital';
        %investment and derivative of value function at steady state
        investmentVector0 = delta*vGridCapital';
        dV0 = 1+bb*(investmentVector0./vGridCapital' - delta);%-bb/2*delta^2+1;
            %%%%%%%%%% NOT SURE %%%%%%%%%%%%%%%%%%
        % dV_upwind makes a choice of forward or backward differences based on
```

```
        % the sign of the drift
        If = muf > 0; % below steady state
        Ib = mub < 0; % above steady state
        I0 = (1-If-Ib); % at steady state
        dV_Upwind = dVf.*If + dVb.*Ib + 0.*I0; % important to include third term
        invest = ((dV_Upwind-1)/bb + delta).*vGridCapital';
            %Get vectors and do implicit Euler...
            diffPoisson = mValueFunction*mTransition(:,nProductivity);
            dn = prodctvt*vGridCapital'.^aalpha - invest - ...
                bb/2*(invest./vGridCapital' -delta).^2.*vGridCapital' + ...
                diffPoisson;
        %CONSTRUCT MATRIX
        X = -min(mub,0)/dk;
        Y = -max(muf,0)/dk + min(mub,0)/dk;
        Z = max(muf,0)/dk;
        %sparse matrix: faster
        AAA = spdiags(Y,0,I,I)+spdiags(X(2:I),-1,I,I)+spdiags([0;Z(1:I-1)],1,I,I);
        B = (r + 1/delta)*speye(I) - AAA + speye(I);
        if explicit==1 %Explicit Euler
            Vnew = deltaV*(dn + AAA*V + V - r*V)+V;
        else
            vecb = dn + 1/deltaV*V;
            Vnew = B\vecb; %SOLVE SYSTEM OF EQUATIONS
        end
        mValueFunctionNew(:,nProductivity) = Vnew;
        mdVf(:,nProductivity) = dVf;
        mdVb(:,nProductivity) = dVb;
        mPolicyFunction(:,nProductivity) = invest;
        %pause
    end
    maxDifference = max(max(abs(mValueFunctionNew-mValueFunction)));
    mValueFunction = mValueFunctionNew;
    iteration = iteration+1;
    if (mod(iteration,10)==0 || iteration==1)
        fprintf(' Iteration = %d, Sup Diff = %2.8f\n', iteration, maxDifference);
    end

end

fprintf(' Iteration = %d, Sup Diff = %2.8f\n', iteration, maxDifference);
fprintf('\n')
%fprintf(' My check = %2.6f\n', mPolicyFunction(1000,3));
fprintf('\n')

toc

%% 6. Plotting results

figure(2)
endPlotN = nGridCapital;
subplot(3,1,1)
plot(vGridCapital(1:endPlotN),mValueFunction(1:endPlotN,:))
xlim([vGridCapital(1) vGridCapital(endPlotN)])
xlabel('k')
title('Value Function, continuous time')

subplot(3,1,2)
plot(vGridCapital(1:endPlotN),mPolicyFunction(1:endPlotN,:))
xlim([vGridCapital(1) vGridCapital(endPlotN)])
```

```
ylim([min(mPolicyFunction(:,1)) (max(mPolicyFunction(:,nGridProductivity))+1)])
xlabel('k')
title('Policy Function = Investment (continuous time)')

% vExactPolicyFunction = aalpha*bbeta.*(vGridCapital.^aalpha);
%
subplot(3,1,3)
plot(vGridCapital(1:endPlotN),mPolicyFunction(1:endPlotN,:)-delta*vGridCapital(1:
    endPlotN)')
hold on
plot(vGridCapital(1:endPlotN),zeros(endPlotN,1),'-k')
xlim([vGridCapital(1) vGridCapital(endPlotN)])
ylim([min(mPolicyFunction(:,1)-delta*vGridCapital(1:endPlotN)') (max(
    mPolicyFunction(:,nGridProductivity)-delta*vGridCapital(1:endPlotN)')+2)])
xlabel('i(k) - delta*k')
title(' dk/dt   = I(k) - delta*k')
hold off

%set(gcf,'PaperOrientation','landscape','PaperPosition',...
%[-0.9 -0.5 12.75 9])
versionp = 'v06_implicit';
if na == 1
    name =  ['Plots/PS01_02_a_',versionp];
    name2 =  ['Plots/PS01_02_a_',versionp,'.fig'];
else
    name =  ['Plots/PS01_02_b_',versionp];
    name2 =  ['Plots/PS01_02_a_',versionp,'.fig'];
end
print('-djpeg',name)
```

Appendix C

R code for finding the optimal value of f.

```
exit_percentage <- c()
ffs <- seq(0.001,0.01,by=0.0001)
for(jj in 1:length(ffs)){
  f <- ffs[jj]
  nk <- length(gridk)
  final <- matrix(0,length(gridz),length(gridk))
  final_k <- matrix(0,length(gridz),length(gridk))
  for(zz in 1:length(gridz)){
    z <- gridz[zz]
    if(zz==1){
      vfun <- c(-f,rep(-100, nk-1))
      gfun <- c(0,rep(-100, nk-1))
      gfun0 <- c(0,rep(-100, nk-1))
      vfun0 <- c(-f,rep(-100, nk-1))
    }else{
      vfun <- final[zz-1,]
      gfun <- final_k[zz-1,]
      vfun0 <- final[zz-1,]
    }

    it=1
    diff=10^10
    while(it<=maxit & diff>tol){
```

```
      it=it+1
      for(kc in 1:nk){
        k0 <- gridk[kc]
        vfun[kc] <- -10000
        for(kcc in 1:nk){
          k0 <- gridk[kc]
          k <- gridk[kcc]
          if(k0==0){
            l <- 0
          }else{
            l <- (W/(alpha_l*z*k0^alpha_k))^(1/(alpha_l-1))
          }
          pi <- z*k0^alpha_k*l^alpha_l - W*l - f
          i <- k - (1-delta)*k0
          if(k == 0){
            phi <- 0
          }else{
            phi <- k - (1-delta)*k0  + 0.5*(i/k-delta)^2*k
          }
          d <- pi - phi
          vhelp=d+M*vfun0[kcc]
          if(vhelp>=vfun[kc]){
            optimal_k_prime <- kcc
            vfun[kc]=vhelp
            gfun[kc]=gridk[kcc]
          }
        }
        if(M*vfun0[optimal_k_prime]<0){
          if(k0==0){
            l <- 0
          }else{
            l <- (W/(alpha_l*z*k0^alpha_k))^(1/(alpha_l-1))
          }
          pi <- z*k0^alpha_k*l^alpha_l - W*l - f
          i <- 0
          d <- pi
          vfun[kc] <- d
          gfun[kc] <- 0
        }
      }
      diff=max(abs(vfun-vfun0))
      vfun0 <- vfun
    }
    final[zz,] <- vfun
    final_k[zz,] <- gfun
  }
  Q <- matrix(0,nk,nk*nz)
  for(i in 1:nk){
    for(zzz in 1:nz){
      Q[which(gridk==final_k[zzz,i]),(i-1)*nz+zzz] <- 1
    }
  }
  TT <- matrix(0,nk*nz,nk*nz)
  for(i in 1:(nk)){
    for(k in 1:nz){
      for(j in 1:(nk)){
        for(l in 1:nz){
          TT[(i-1)*nz+k,(j-1)*nz+l] <- Q[i,(j-1)*nz+l]*P[k,l]
        }
```

```
      }
    }
  }
  X <- matrix(NA,nz,nk)
  for(i in 1:nz){
    for(j in 1:nk){
      if(final_k[i,j]==0){
        X[i,j] <- 0
      }else{
        X[i,j] <- 1
      }
    }
  }
  X <- t(X)
  XX <- matrix(NA,nk*nz,nk*nz)
  X_vec <- c()
  for(i in 1:nk){
    for(j in 1:nz){
      X_vec[(i-1)*nz+j] <- X[i,j]
    }
  }
  for(i in 1:(nk*nz)){
    XX[i,] <- X_vec
  }
  library(Matrix)
  XX <- Matrix(XX, sparse=T)
  TT <- Matrix(TT, sparse=T)
  gamma_0 <- rep(0, nk*nz)
  gamma_0[1:nz] <- stationary_dist_z
  goal <- solve(diag(ncol(XX))-TT*XX)%*%(gamma_0)
  plot(goal)
  E <- 1/sum(goal)
  goal <- goal*E
  exit_percentage[jj] <- as.numeric(t(goal)%*%(1-X_vec))
}

optimal_f <- ffs[which(min(abs(exit_percentage-0.025))==abs(exit_percentage-0.025)
    )]
```