Problem Set 4

1 Numerical Questions

1.1 Competitive Equilibrium in One-Sector Growth Model Under Certainty

Consider a decentralized economy with a large number of firms and a large number of households of total size normalized to one.

The state of the economy is the aggregate capital stock. We distinguish between "big K" and "little k". Households and firms see big K as the aggregate state that is beyond their control. Little k is chosen by firms and households. When making their optimization decisions, they do not see the connection between K and k. In equilibrium $K = k$.

We specify the price functions $\tilde{r}(K)$ and $w(K)$ that represent, respectively, the rental price on capital and the wage rate when the aggregate state is K. The law of motion for the aggregate capital is denoted as

$$K' = G(K).$$

A firm maximizes its profits given the factor prices $(\tilde{r}_t, w_t) = (\tilde{r}(K_t), w(K_t))$:

$$\max_{k_t, n_t} A k_t^{\alpha} n_t^{1-\alpha} - \tilde{r}_t(K_t) k_t - w(K_t) n_t,$$

where $A k_t^{\alpha} n_t^{1-\alpha}$ is a production function, k_t is the capital input and n_t is the labour input. Note that the firms' first-order conditions and market clearing implies that the equilibrium pricing functions are

$$\begin{aligned} \tilde{r}(K) &= \alpha A K^{\alpha-1} \\ w(K) &= A K^{\alpha} - \tilde{r}(K) K. \end{aligned} \tag{1}$$

Each household possesses homogeneous physical capital, denoted k, and supplies one unit of labour. Since the number of households is normalized to one, total supply of

labour in the economy is one (i.e. $n = 1$). The problem faced by the households is

$$\begin{aligned} \max_{c_t, x_t} \quad & \textstyle\sum_{t=0}^{\infty} \beta^t \log c_t \\ \text{subject to} \quad & c_t + x_t = \tilde{r}(K_t)k_t + w(K_t) \\ & k_{t+1} = (1-\delta)k_t + x_t \\ & k_0 \text{ given} \end{aligned}$$

where c_t is consumption, $x_t$is gross investment, $\tilde{r}(K_t)$ is the rental rate on capital, $w(K_t)$ is a competitive wage, and $\delta \in (0, 1)$is the depreciation rate on capital. Given the initial aggregate capital $K_0 = k_0$, the law of motion of aggregate capital, $K_{t+1} = G(K_t)$, determines a sequence of future aggregate capital stock $\{K_t\}_{t=0}^{\infty}$ and hence, through equation (1), a sequence of future prices $\{(w_t, \tilde{r}_t)\}_{t=0}^{\infty}$.

Each household solves the above maximization problem taking as given the price sequence $\{(w_t, \tilde{r}_t)\}_{t=0}^{\infty}$. The policy function for the household is

$$k_{t+1} = g(k_t, K_t).$$

In equilibrium, $k = K$, so this household maximization, in turn, determines the law of motion $K = g(K, K)$, which then determines another sequence of prices $\{(w_t, \tilde{r}_t)\}_{t=0}^{\infty}$.

Rewrite the household problem in terms of Bellman's equation as follows:

$$\begin{aligned} V(k, K) \quad = \quad & \max_{c, k'} \log(c) + \beta V(k', G(K)) \\ & \text{subject to: } c + k' \leq (1 + \tilde{r}(K) - \delta)k + w(K). \end{aligned} \tag{2}$$

Let the associated policy function be $g(k, K)$.

Define a *recursive competitive equilibrium* for this economy a s a collection of functions: $V(k, K)$, $c(k, K)$, $g(k, K)$, $\tilde{r}(K)$, $w(K)$, $G(K)$, and $C(K)$ such that

1. Given the pricing functions, $\tilde{r}(K)$ and $w(K)$, and aggregate law of motion $G(K)$, the functions $V(k, K)$, $c(k, K)$, and $g(k, K)$, satisfy the household Bellman equation.

2. Given the pricing functions, $\tilde{r}(K)$ and $w(K)$, firms maximize profits.

3. the markets for consumption good, labour, and capital clear so that $c(k, K) = C(K)$, $n = 1$, and $g(k, K) = G(K)$.

We may define a steady state of this economy with the steady state aggregate capital level K^* that satisfies $K^* = g(K^*, K^*) = G(K^*)$.

In the following you are asked to numerically solve the recursive competitive equilibrium.

1. Use the parameter values $\alpha = 0.25$, $\beta = 0.96$ $A = (\alpha\beta)^{-1}$, and $\delta = 1$. Our discretization for k and K will be $k_i = 0.8 + (i-1)\kappa$ and $K_i = 0.8 + (i-1)\kappa$, for $i = 1, 2, ..., n$. We choose $\kappa = 0.01$ with $n = 41$ so $k_1 = K_1 = 0.8$ and $k_n = K_n = 1.2$. Also, set the value of the "relaxation parameter" as $\xi = 0.99$. You should use a tolerance $\varepsilon_0 = 0.00001$ to test the convergence of the value function and a tolerance of $\varepsilon_1 = 0.01$ for the convergence of the household policy function and the aggregate transition function.
   To numerically solve the recursive equilibrium, consider the following algorithm:

   (a) Start from the initial guess of the law of motion of aggregate capital $G^0(K)$. Choose a constant function $G^0(K) = 1$ for every K as an initial guess. Note that this initial guess implies that households expect aggregate capital to be $K = 1$ next period, regardless of what aggregate capital is today. This is a reasonable guess as $K = 1$ is the steady state capital level.

   (b) After h iterations, we have the law of motion of capital $G^h(K)$. Then, using the pricing functions (1), we solve a discretized version of the household Bellman equation by backward induction:

$$V(k, K) = \max_{k'} \log\left([1 + \tilde{r}(K) - \delta]k + w(K) - k'\right) + \beta V(k', G^h(K)). \quad (3)$$

   Let the policy function be $g^h(k, K)$. Se below for how to solve (3) .

   (c) If $||g^h(K, K) - G^h(K)|| < \varepsilon_1$, then $G^h(K)$ is the fixed point and so stop. If $||g^h(K, K) - G^h(K)|| \geq \varepsilon_1$, then compute a new estimate of $G^{h+1}(K) = \xi G^h(K) + (1 - \xi)g^h(K, K)$ for a fixed "relaxation parameter" $\xi \in (0, 1)$ and go back to step (b).

2. Plot on a graph both the law of motion for the recursive equilibrium $G(K)$, the house household policy function, and the policy function of corresponding *planner's* Bellman equation.[1] Check if they are the same. Hand in your program and the output.

1.1.1 How to numerically solve (3)

Notice that the state space is now (k, K) rather than k. Thus, the number of possible states in the household problem is n^2.

First, define the utility function $U(k, K, k') \equiv \log\left([1 + \tilde{r}(K) - \delta]k + w(K) - k'\right)$. This utility function can be represented by an "n^2 by n" utility matrix, say $\mathbf{U}$, where an

[1] We computed this in homework 1.

$[(i+(j-1)n), i']$ element is equal to $\log\left(\max\left\{[1+\tilde{r}(K_j)-\delta]k_i + w(K_j) - k_{i'}, \texttt{eps}\right\}\right)$.[2]

Second, the law of motion $G^h(K)$ can be represented by an "n by n" Markov matrix, denoted by Q_K, such that the (j,j')th element is 1 if $K_{j'} = G^h(K_j)$ and 0 otherwise. Let $q_{j,j'}$ be the (j,j')th element of the Markov matrix Q_K. The law of motion is then represented by the matrix operation

$$K' = G^h(K) = Q_K \times K.$$

Now, the discretized Bellman equation is given by

$$V(i,j) = \max_{i'=1,2,...,n} U(k_i, K_j, k_{i'}) + \beta \sum_{j=1}^{n} q_{j,j'} V(i',j'). \tag{4}$$

We solve the fixed point of the Bellman equation as follows:

1. Set the initial guess $V^0 = \mathbf{0}_{n,n}$, where $\mathbf{0}_{n,n}$ is an "n by n" matrix with each element a scalar 0.

2. After l value function iterations, we have an "n by n" matrix representing a value function, V^l. Given V^l, compute

$$V^{l+1}(i,j) = \max_{i'=1,2,...,n} U(k_i, K_j, k_{i'}) + \beta \sum_{j=1}^{n} q_{j,j'} V^l(i',j'),$$

for every $(i,j) \in \{1,2,...,n\} \times \{1,2,...,n\}$.

- One way to do this is to use a "do-loop" twice. For $(i,j) = (1,1)$, compute $U(k_1, K_1, k_{i'}) + \beta \sum_{j'=1}^{\infty} q_{1,j'} V^l(i',j')$ for $i' = 1,2,...,n$ and store their values in an "n by 1" vector. Then, find the max among those values. If the $i^*(1,1)$th element is the max, then $V^{l+1}(1,1) = U\left(k_1, K_1, k_{i^*(1,1)}\right) + \beta \sum_{j'=1}^{\infty} q_{1,j'} V^l(i^*(1,1),,j')$. This is the inner loop. Repeat this for every pair of $(i,j) \in \{1,2,...,n\} \times \{1,2,...,n\}$ ("outer loop") to get $\{V^{l+1}(i,j)\}$ and store them in an "n by n" matrix; this is the function V^{l+1}.

- In practice, avoid loops as much as possible in Matlab since they slow down your program. Instead, use vector or matrix operations. In the current context, we may "vectorize" the state space. First, given the value function V^l, an "n by n" matrix with element $V^l(i,j) = V^l(k_i, K_j)$, create an "n by n" matrix, $EV^l = V^l Q^\top$ (where $\top$ is the transpose) which will represent the next period's value

[2]The max operation inside the log is to ensure we do not take the log of a non-positive number. `eps` represents "machine zero" in Matlab. This is the smallest positive number that the computer can distinguish from zero.

function under the law of motion of aggregate capital Q. The (i', j)th element of EV^l represents the *next period's* value function when $k = k_{i'}$ aggregate capital is K_j.

Second, compute an "n^2 by n" matrix $W^{l+1} = \mathbf{U} + \beta \mathbf{1}_n \otimes \left(EV^l\right)^\top$, where $\mathbf{1}_n$ is an "n by 1" vector of ones, $\top$ is a transpose, and $\otimes$ is the Kronecker product.[3] Third, for each *row*, take the max across columns to compute an "n^2 by 1" vector V^{l+1}. You can do this using just one command.[4] Finally, to get back to an "n by n" matrix, use the command `reshape(V1,n,n)`.

3. Repeat the recursive computations of step 2 until $\max_{(i,j)\in\{1,2,\ldots,n\}\times\{1,2,\ldots,n\}} |V^l(i,j) - V^{l+1}(i,j)| < 0.0001$. Let V^* be the computed fixed point; the V that solves the Bellman equation.

4. Compute the policy function represented by an "n by n" Markov matrix, denoted by P_k, such that the (i, i')th element is 1 if $k_{i'} = g(K_i, K_i)$. Note that we want the policy function implies when markets clear, that is, when $k_i = K_i$. The policy function is then represented by the matrix operation

$$k' = g^h(K, K) = P_k \times k.$$

[3] In Matlab, you can use the command `W=U+beta*kron(ones(n,1),EV')`. Alternatively, you can use the command `W=U+beta*repmat(EV',n,)`. Look at the help files for `kron` and `repmat` to see exactly what each does.

[4] In Matlab, you can use the command `[V1 I]=max(W,[],2)`. This will result in two vectors. V1 will contain the value of the maximum of W, and I will contain the position of the maximum.

1.1.2 Example with $n = 5$

Below I reproduce a sample of the matrices for an example with $n = 5$. The following are the numerical values for the "n by 1" vectors k, K, $r(K)$, and $w(K)$.

```
k =         K =         r =         w =
    0.8000      0.8000      1.2314      2.9554
    0.9000      0.9000      1.1273      3.0438
    1.0000      1.0000      1.0417      3.1250
    1.1000      1.1000      0.9698      3.2004
    1.2000      1.2000      0.9085      3.2707
```

Below I reproduce the "$n \times n$ by n" Utility matrix U, and the values of "n by n" matrices Q_K, V^0, and EV^0. The latter three are the values for the initial guess of $G(K_j) = 1$ for all k_j, and for the first iteration of the value function. Below these we have the "$n \times n$ by n" matrix W, the "n by 1" vectorized versions of the value function V^1, and the indicator of the column containing the maximum value of W for each row: I.

```
U =                                             Q =

    1.1444    1.1121    1.0786    1.0440    1.0082         0     0     1     0     0
    1.1829    1.1518    1.1196    1.0864    1.0521         0     0     1     0     0
    1.2199    1.1899    1.1590    1.1272    1.0942         0     0     1     0     0
    1.2556    1.2267    1.1970    1.1663    1.1346         0     0     1     0     0
    1.2901    1.2622    1.2335    1.2039    1.1735         0     0     1     0     0
    1.1460    1.1137    1.0803    1.0458    1.0100
    1.1812    1.1500    1.1179    1.0846    1.0502
    1.2152    1.1851    1.1541    1.1220    1.0889    V0 =
    1.2481    1.2190    1.1890    1.1581    1.1262
    1.2800    1.2518    1.2228    1.1929    1.1621         0     0     0     0     0
    1.1500    1.1179    1.0846    1.0502    1.0146         0     0     0     0     0
    1.1825    1.1514    1.1192    1.0860    1.0517         0     0     0     0     0
    1.2139    1.1838    1.1527    1.1206    1.0874         0     0     0     0     0
    1.2444    1.2152    1.1850    1.1540    1.1219         0     0     0     0     0
    1.2740    1.2456    1.2164    1.1863    1.1553
    1.1557    1.1237    1.0906    1.0565    1.0211
    1.1858    1.1547    1.1227    1.0896    1.0554    EV =
    1.2150    1.1848    1.1538    1.1217    1.0886
    1.2433    1.2141    1.1839    1.1528    1.1207         0     0     0     0     0
    1.2709    1.2425    1.2132    1.1830    1.1519         0     0     0     0     0
    1.1624    1.1306    1.0978    1.0639    1.0287         0     0     0     0     0
    1.1904    1.1595    1.1277    1.0947    1.0607         0     0     0     0     0
    1.2177    1.1876    1.1567    1.1247    1.0917         0     0     0     0     0
    1.2442    1.2150    1.1848    1.1538    1.1217
    1.2700    1.2416    1.2122    1.1820    1.1509


W =                                             V1 =        I =

    1.1444    1.1121    1.0786    1.0440    1.0082      1.1444       1
    1.1829    1.1518    1.1196    1.0864    1.0521      1.1829       1
    1.2199    1.1899    1.1590    1.1272    1.0942      1.2199       1
    1.2556    1.2267    1.1970    1.1663    1.1346      1.2556       1
    1.2901    1.2622    1.2335    1.2039    1.1735      1.2901       1
    1.1460    1.1137    1.0803    1.0458    1.0100      1.1460       1
    1.1812    1.1500    1.1179    1.0846    1.0502      1.1812       1
    1.2152    1.1851    1.1541    1.1220    1.0889      1.2152       1
    1.2481    1.2190    1.1890    1.1581    1.1262      1.2481       1
    1.2800    1.2518    1.2228    1.1929    1.1621      1.2800       1
    1.1500    1.1179    1.0846    1.0502    1.0146      1.1500       1
    1.1825    1.1514    1.1192    1.0860    1.0517      1.1825       1
    1.2139    1.1838    1.1527    1.1206    1.0874      1.2139       1
    1.2444    1.2152    1.1850    1.1540    1.1219      1.2444       1
    1.2740    1.2456    1.2164    1.1863    1.1553      1.2740       1
    1.1557    1.1237    1.0906    1.0565    1.0211      1.1557       1
    1.1858    1.1547    1.1227    1.0896    1.0554      1.1858       1
    1.2150    1.1848    1.1538    1.1217    1.0886      1.2150       1
```

```
    1.2433    1.2141    1.1839    1.1528    1.1207         1.2433         1
    1.2709    1.2425    1.2132    1.1830    1.1519         1.2709         1
    1.1624    1.1306    1.0978    1.0639    1.0287         1.1624         1
    1.1904    1.1595    1.1277    1.0947    1.0607         1.1904         1
    1.2177    1.1876    1.1567    1.1247    1.0917         1.2177         1
    1.2442    1.2150    1.1848    1.1538    1.1217         1.2442         1
    1.2700    1.2416    1.2122    1.1820    1.1509         1.2700         1
```

The following matrices are (again for the initial guess for $G(K_j) = 1$ for all k_j) The vectorized version of the converged value function V^1, The vector version of the indicator I, the matrix version of V^1, the policy function P_k^0, and the new guess for the aggregate transition function $Q_K^1 = \xi Q_K^0 + (1-\xi)P_k^0$.

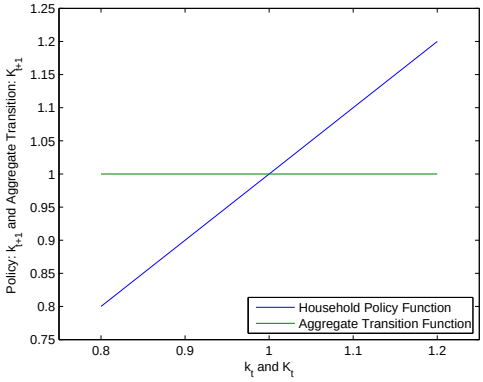
```
V1 =       I =    V1 =

   28.7452      1    28.7452   28.7468   28.7509   28.7565   28.7632
   28.7842      2    28.7842   28.7825   28.7838   28.7872   28.7920
   28.8231      3    28.8231   28.8181   28.8168   28.8179   28.8207
   28.8619      4    28.8619   28.8537   28.8496   28.8485   28.8494
   28.9006      5    28.9006   28.8892   28.8825   28.8790   28.8780
   28.7468      1
   28.7825      2
   28.8181      3
   28.8537      4  P =
   28.8892      5
   28.7509      1       1     0     0     0     0
   28.7838      2       0     1     0     0     0
   28.8168      3       0     0     1     0     0
   28.8496      4       0     0     0     1     0
   28.8825      5       0     0     0     0     1
   28.7565      1
   28.7872      2
   28.8179      3  Q =
   28.8485      4
   28.8790      5     0.0100         0    0.9900         0         0
   28.7632      1          0    0.0100    0.9900         0         0
   28.7920      2          0         0    1.0000         0         0
   28.8207      3          0         0    0.9900    0.0100         0
   28.8494      4          0         0    0.9900         0    0.0100
   28.8780      5
```

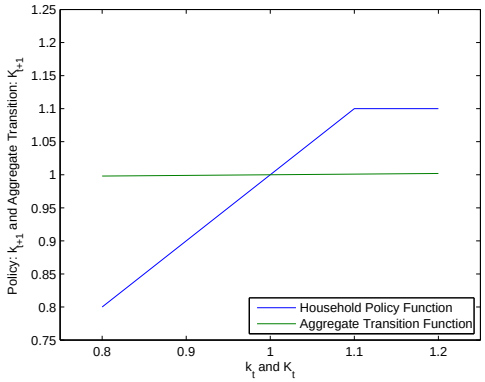
Using the new $Q_K^1 = \xi Q_K^0 + (1-\xi)P_k^0$, we repeat the value function iteration again.

Note that the household policy function is $k' = g(K, K) = P_k^0 k$ and the new aggregate transition function for capital is $K' = G(K) = Q_K^1 K$. Below I plot the first five iterations on Q_K. That is, I plot the guess of $K' = Q_K^l K$ and the implied household policy function $k' = P_k^0 k$.

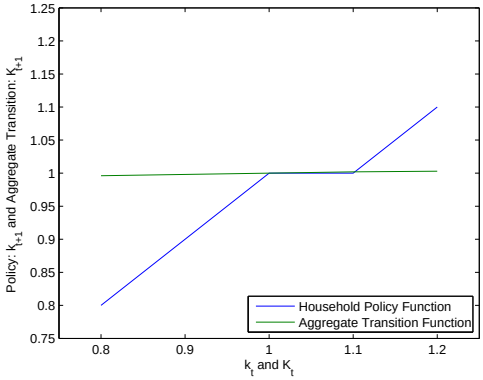
Policy functions, $k' = P_k^l k$, and Aggregate capital transitions, $K' = Q_K^l K$, for an example with grid size $n = 5$.



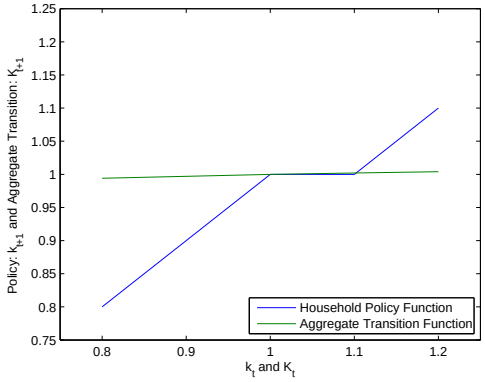
(a) $k' = P_k^1 k$ and $K' = Q_K^1 K$



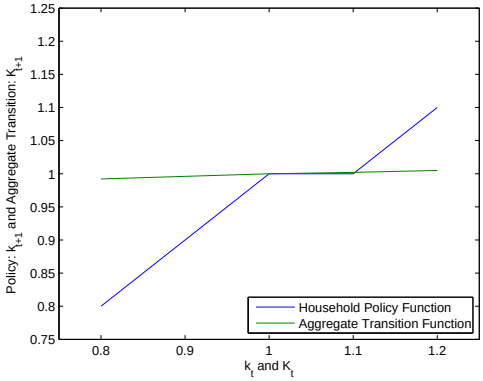
(b) $k' = P_k^2 k$ and $K' = Q_K^2 K$



(c) $k' = P_k^3 k$ and $K' = Q_K^3 K$



(d) $k' = P_k^4 k$ and $K' = Q_K^4 K$



(e) $k' = P_k^5 k$ and $K' = Q_K^5 K$