Problem Set 3

1 Numerical Questions

It is very important to add comments generously throughout your code so I (and so your future self) can understand what you are doing. Unless something is completely obvious add comments to each small section of code, perhaps even to each line.

1.1 Search Model with Layoffs

Consider the job search model discussed in class and in section 6.3.4 of Ljungqvist and Sargent in which employed workers face an exogenous probability α of being fired. Redo question 2.1 from problem set 1 with the addition of this firing probability. Assume $\alpha = 0.1$.

1.2 Stochastic Neoclassical Growth Model

1. Consider the Bellman equation corresponding to the one-sector stochastic optimal growth problem:

$$V(k,z) = \max_{0 \leq k' \leq e^z f(k)} U\left(e^z f(k) - k'\right) + \beta E[V(k',z')]. \tag{1}$$

We will choose the functional forms:

$$\begin{aligned} U(c) &= \log c \\ f(k) &= Ak^{\alpha} \end{aligned}$$

so that we have a closed form solution to the policy function: $k' = e^z \alpha \beta A k^{\alpha}$. Assume that z is distributed according to a normal distribution with mean zero and variance σ^2.

Approximate the continuous state spaces of k and z by a finite set $K = \{k_1, k_2, ..., k_n\}$ and $Z = \{z_1, z_2, ..., z_m\}$. Given the finite set of Z, let q_j represent $\Pr(z = z_j)$

$j = 1, 2, \ldots, m$. Then the discretized Bellman equation is given by

$$V(i,j) = \max_{i'=1,2,\ldots,n} U\left(\exp(z_j)f(k_i) - k_{i'}\right) + \beta \sum_{j'=1}^{m} q_{j'} V(i', j'), \tag{2}$$

where $V(i,j)$ represents the (i,j)th element of an "n by m" matrix. Specifically, the continuous function $V(k,z)$ that maps $[0, \overline{k}] \times \mathbb{R}$ into $\mathbb{R}$ is approximated by a function $V(i,j)$ which maps $\{1, 2, ..., n\} \times \{1, 2, ..., m\}$ into $\mathbb{R}$.
We are going to solve this discretized Bellman equation numerically (on a computer) using value function iteration.

Set the parameter values to: $\beta = 0.96$, $\alpha = 0.25$. Also, set $A = (\alpha\beta)^{-1}$ implying that the steady state capital level is e^z.
Our discretization will be $k_i = \kappa + (i-1)\kappa$. We choose $\kappa = 3.5/(n-1)$ with $n = 41$ so that $k_1 = 0.0875$ and $k_n = 3.5875$ (We don't really want to include $k = 0$ since this would lead to zero output and infinite marginal utility in the next period. 3.5 seems to be a reasonable upper bound.).
For z, which is normally distributed around zero with variance σ^2, we use a uniform grid consisting of equi-spaced points between -3σ and 3σ. By choosing $\Delta z = 6\sigma/(m-1)$, our discritization for z will be $z_j = -3\sigma + (j-1) \times \Delta z$ for $j = 1, 2, \ldots, m$. We set $m = 41$. Accordingly, the normal distribution is approximated by multinomial distribution as follows:

$$q_j = \frac{\phi(z_j/\sigma)}{\sum_{j'=1}^{m} \phi(z_{j'}/\sigma)},$$

where $\phi(\cdot)$ is the standard normal density function. The normalization insures that q_j is a well defined probability density. Let $\mathbf{q}$ be an m by 1 vector with the jth element equal to q_j. We set $\sigma = 0.5$.

(a) Write a Matlab program that numerically solves the fixed point of the discretized Bellman equation 2 using the value function iteration algorithm:

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

   ii. Given V^l (starting with V^0), compute the next guess guess

$$V^{l+1}(i,j) = \max_{i'=1,2,\ldots,n} U\left(\exp(z_j)f(k_i) - k_{i'}\right) + \beta \sum_{j'=1}^{m} q_{j'} V^l(i', j'),$$

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

- One way to do this is to use a "do-loop" twice. For $(i,j) = (1,1)$, compute $U\left(\exp(z_1)f(k_1) - k_{i'}\right) + \beta \sum_{j'=1}^{m} q_{j'} V^l(i', j')$ for $i' = 1, 2, ..., n$ and store their values in an "n by 1" vector.[1] Then, find the max among those values. If the i^*th element is the max, then $V^{l+1}(1,1) = U\left(\exp(z_1)f(k_1) - k_{i*}\right) + \beta \sum_{j'=1}^{m} q_{j'} V^l(i^*, j')$. This is the inner loop. Repeat this for every pair $(i,j) \in \{1,2,3,...,n\} \times \{1,2,3,\ldots,m\}$ ("outer loop") to get
  $\{V^{l+1}(i,j)\}$ and store them in an "n by m" 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. For this it is convenient to "vectorized" the state space. In the current context, first create an "nm by n" matrix we will call the utility matrix $\mathbf{U}$, with the $(i + (j-1)m, i')$ element is equal to $U\left(\exp(z_j)f(k_i) - k_{i'}\right)$. The $(i + (j-1)m)$th row represents the current state (k_i, z_j) while the i'th column represents a choice of the next period's capital stock, $k_{i'}$. Second, compute an n by 1 vector $EV^l = V^l\mathbf{q}$, which represents the expected value function; the i'th element represents the expected value of the next period's value function when $k = k_{i'}$ is chosen. Third, given the utility matrix and the expected value function matrix, compute $W^{l+1} = \mathbf{U} + \beta \mathbf{1}_{nm} \otimes \left(EV^l\right)'$, where $\mathbf{1}_{nm}$ is an "nm by 1" vector of ones, $'$ is a transpose, and $\otimes$ is the Kronecker product.[2] Fourth, for each *row*, take the max across column to compute an *"nm by 1"* vector V^{l+1}. You can do this using just one command.[3] Finally, to get an n by m matrix, use the command reshape(V,n,m).

iii. Repeat the recursive computations of Step (ii) until $\left(V^l(i,j) - V^{l+1}(i',j')\right)$ is close to zero, specifically, until $\max_{(i,j)\in\{1,2,...,n\}\times\{1,2,...,m\}} \left|V^l(i,j) - V^{l+1}(i,j)\right| < \epsilon/(1-\beta)$, where $\epsilon = 10^{-5}$. Let V^* be the converged value function (the fixed point).

iv. Compute the policy function, P, which is an "nm by n" matrix of which the $(i+(j-1)m, i^*)$th element is 1 if $i^* = \arg\max_{i'\in\{1,2,...,n\}} U\left(\exp(z_j)f(k_i) - k_{i'}\right) +$

[1] Some combinations of (z, k, k') lead to negative consumption. Since $u(c) = \log c$, you cannot evaluate the utility if c is negative. Note that strictly negative consumption is infeasible and also that a representative agent never chooses zero consumption since zero consumption leads her utility to minus infinity. One way to deal with this issue in numerical programming is to take $c = \max\{\exp(z)f(k) - k', eps\}$ where *eps* is the "machine epsilon," so that you can evaluate $\log c$ for every choice of (z, k, k') and yet you never choose the tomorrow's capital stock k' that leads to negative or zero consumption.

[2] In Matlab, you can use the command `W=U+beta*kron(ones(n,1),EV')`.

[3] In Matlab, you can use the command `V1=max(W,[],2)`.

$\beta V^*(i', j')$, 0 otherwise. [4]

(b) The policy function together with the distribution of z determines a transition function of the state space (k_i, z_j). Specifically, an nm matrix $\Pi \equiv \mathbf{q}' \otimes P$ defines a transition function.

i. Compute an invariant distribution of (k, z). [Hint. Compute the "mn by 1" vector ξ_N by iterating on

$$\xi'_{j+1} = \xi'_j \Pi \tag{3}$$

until $||\xi_{j+1} - \xi_j|| < \varepsilon$. Start with a uniform distribution for ξ_1 (ie, $\xi_1(i) = 1/mn$ for each element i).[5][6]

(c) To see if the value function is converged or not (or if your program is working or not), first compute the utility function under the policy function P, say $\mathbf{U}^P$, which is an nm by 1 vector with the $(i + (j-1)m)$th element equal to $U\left(\exp(z_j)f(k_i) - k_{i*(i)}\right)$ where $k_{i*(i)}$ is the optimal choice for the next capital stock given the current state is k_i. Next, compute $\mathbf{U}^P + \beta\Pi V^*$ and check whether it is equal to V^* or not.

(d) Suppose that this one-sector growth model is a good approximation of reality. Furthermore, suppose that productivity shocks are independent across countries. Derive the model's prediction on the invariant distribution of capital stock per worker and output per worker across countries. Plot the steady state distribution on a graph using the `bar` command in Matlab [Hint. For output per worker, use the invariant distribution of (k, z).]

2. Extra Question [Not required, but it might be fun.] Suppose z follows a first order autoregressive process: $z_{t+1} = \rho z_t + \epsilon$ where $\rho = 0.9$ and $\epsilon$is normally distributed with $\sigma = 0.5$ as before. Can you do the similar programming exercise with this assumption? Hint: The transition function for z can be approximated by multinomial

[4] In Matlab, look at the help for the `max` command to see how to do this in one step.

[5] Here we are using the notation ξ_N instead of ξ_∞ just because we will take N steps toward the invariant distribution, not an infinite number of steps. Also note, it is not necessary to start with a uniform guess for ξ_1, any starting guess is fine, as long as it is a proper distribution (elements are non-negative and sum to one).

[6] The original hint is also fine, but the notation is poor. It should have read: Compute Π^{jN} by choosing a large j and N so that $\left|\left|\Pi^{jN} - \Pi^{(j+1)N}\right|\right| < 0.0001/\left(1 - \varepsilon^N\right)$, where $||\cdot||$ is a norm you pick. Note that now each row of Π^{jN} will be an invariant distribution of (k, z). The reason this works is that multiplying Π by itself jN times does exactly the same thing as the iteration in (3) repeated mn times, where the first row is the result when we start with the guess $\xi'_1 = \Pi(1, :)$, the second row is the result when we start with the guess $\xi'_1 = \Pi(2, :)$, etc.

distribution as follows:

$$\Pr(z_{t+1} = z_j | z_t = z_i) = q_{ij} = \frac{\phi((z_j - \rho z_i)/\sigma)}{\sum_{j'=1}^{m} \phi((z_{j'} - \rho z_i)/\sigma)}.$$

Hint for question 2.2.1 Below I have printed, for the case of $n = 5$, the discrete apporximation to the shocks, z_j, the probability of these shocks, q_j , the numerical values of the utility matrix $\mathbf{U}$, the first three iterations of W^{l+1} and $\tilde{V}^{l+1}$, and the approximate policy function P (along with k and $P \times k$).

```
z =         exp(z) =            q =
   -1.5000         0.2231         0.0066
   -0.7500         0.4724         0.1942
         0         1.0000         0.5983
    0.7500         2.1170         0.1942
    1.5000         4.4817         0.0066

U =
   -3.7220      -Inf      -Inf      -Inf      -Inf
   -1.6383      -Inf      -Inf      -Inf      -Inf
   -1.1764      -Inf      -Inf      -Inf      -Inf
   -0.9247      -Inf      -Inf      -Inf      -Inf
   -0.7559      -Inf      -Inf      -Inf      -Inf
    0.0282   -1.8736      -Inf      -Inf      -Inf
    0.3284   -0.6660      -Inf      -Inf      -Inf
    0.4887   -0.2807      -Inf      -Inf      -Inf
    0.5972   -0.0597   -2.7022      -Inf      -Inf
    0.6788    0.0921   -1.5073      -Inf      -Inf
    1.1489    0.8241    0.3399   -0.6351      -Inf
    1.3654    1.1126    0.7735    0.2565   -0.8738
    1.4881    1.2680    0.9853    0.5898   -0.0741
    1.5736    1.3735    1.1230    0.7880    0.2807
    1.6392    1.4530    1.2241    0.9267    0.5014
    2.0355    1.9142    1.7760    1.6157    1.4246
    2.2268    2.1277    2.0176    1.8939    1.7527
    2.3373    2.2489    2.1521    2.0448    1.9246
    2.4150    2.3336    2.2450    2.1477    2.0399
    2.4750    2.3985    2.3157    2.2254    2.1261
    2.8441    2.7918    2.7367    2.6783    2.6163
    3.0254    2.9820    2.9367    2.8891    2.8392
    3.1309    3.0919    3.0514    3.0091    2.9650
    3.2054    3.1693    3.1319    3.0929    3.0524
    3.2632    3.2291    3.1938    3.1573    3.1194
```

First Iteration:

```
W =
   -3.7220      -Inf      -Inf      -Inf      -Inf
   -1.6383      -Inf      -Inf      -Inf      -Inf
   -1.1764      -Inf      -Inf      -Inf      -Inf
   -0.9247      -Inf      -Inf      -Inf      -Inf
   -0.7559      -Inf      -Inf      -Inf      -Inf
    0.0282   -1.8736      -Inf      -Inf      -Inf
    0.3284   -0.6660      -Inf      -Inf      -Inf
    0.4887   -0.2807      -Inf      -Inf      -Inf
    0.5972   -0.0597   -2.7022      -Inf      -Inf
    0.6788    0.0921   -1.5073      -Inf      -Inf
    1.1489    0.8241    0.3399   -0.6351      -Inf
    1.3654    1.1126    0.7735    0.2565   -0.8738
    1.4881    1.2680    0.9853    0.5898   -0.0741
    1.5736    1.3735    1.1230    0.7880    0.2807
    1.6392    1.4530    1.2241    0.9267    0.5014
    2.0355    1.9142    1.7760    1.6157    1.4246
    2.2268    2.1277    2.0176    1.8939    1.7527
    2.3373    2.2489    2.1521    2.0448    1.9246
    2.4150    2.3336    2.2450    2.1477    2.0399
    2.4750    2.3985    2.3157    2.2254    2.1261
    2.8441    2.7918    2.7367    2.6783    2.6163
    3.0254    2.9820    2.9367    2.8891    2.8392
    3.1309    3.0919    3.0514    3.0091    2.9650
    3.2054    3.1693    3.1319    3.0929    3.0524
    3.2632    3.2291    3.1938    3.1573    3.1194

V1 =
   -3.7220    0.0282    1.1489    2.0355    2.8441
   -1.6383    0.3284    1.3654    2.2268    3.0254
   -1.1764    0.4887    1.4881    2.3373    3.1309
   -0.9247    0.5972    1.5736    2.4150    3.2054
   -0.7559    0.6788    1.6392    2.4750    3.2632
```

Second iteration:

```
W =
   -2.6830      -Inf      -Inf      -Inf      -Inf
   -0.5992      -Inf      -Inf      -Inf      -Inf
   -0.1373      -Inf      -Inf      -Inf      -Inf
    0.1143      -Inf      -Inf      -Inf      -Inf
    0.2832      -Inf      -Inf      -Inf      -Inf
    1.0672   -0.6041      -Inf      -Inf      -Inf
    1.3675    0.6035      -Inf      -Inf      -Inf
    1.5278    0.9888      -Inf      -Inf      -Inf
    1.6363    1.2098   -1.3081      -Inf      -Inf
    1.7179    1.3616   -0.1133      -Inf      -Inf
    2.1880    2.0936    1.7340    0.8448      -Inf
    2.4045    2.3821    2.1675    1.7364    0.6717
    2.5271    2.5374    2.3793    2.0698    1.4714
    2.6127    2.6430    2.5170    2.2680    1.8262
    2.6783    2.7225    2.6181    2.4066    2.0469
    3.0746    3.1836    3.1700    3.0956    2.9701
    3.2659    3.3972    3.4117    3.3739    3.2982
    3.3763    3.5184    3.5461    3.5248    3.4701
    3.4541    3.6031    3.6390    3.6277    3.5854
    3.5141    3.6680    3.7098    3.7054    3.6716
    3.8831    4.0613    4.1307    4.1583    4.1618
    4.0645    4.2515    4.3307    4.3691    4.3847
    4.1699    4.3614    4.4454    4.4891    4.5105
    4.2445    4.4388    4.5259    4.5729    4.5979
    4.3022    4.4986    4.5879    4.6373    4.6648

V1 =
   -2.6830    1.0672    2.1880    3.1836    4.1618
   -0.5992    1.3675    2.4045    3.4117    4.3847
   -0.1373    1.5278    2.5374    3.5461    4.5105
    0.1143    1.6363    2.6430    3.6390    4.5979
    0.2832    1.7179    2.7225    3.7098    4.6648
```

Third iteration:

```
W =
   -1.6634      -Inf      -Inf      -Inf      -Inf
    0.4204      -Inf      -Inf      -Inf      -Inf
    0.8823      -Inf      -Inf      -Inf      -Inf
    1.1339      -Inf      -Inf      -Inf      -Inf
    1.3028      -Inf      -Inf      -Inf      -Inf
    2.0868    0.4226      -Inf      -Inf      -Inf
    2.3871    1.6302      -Inf      -Inf      -Inf
    2.5474    2.0155      -Inf      -Inf      -Inf
    2.6559    2.2365   -0.2709      -Inf      -Inf
    2.7375    2.3883    0.9240      -Inf      -Inf
    3.2076    3.1203    2.7712    1.8965      -Inf
    3.4241    3.4088    3.2048    2.7881    1.7334
    3.5468    3.5642    3.4166    3.1214    2.5331
    3.6323    3.6697    3.5543    3.3196    2.8879
    3.6979    3.7492    3.6554    3.4583    3.1086
    4.0942    4.2104    4.2073    4.1473    4.0318
    4.2855    4.4239    4.4489    4.4255    4.3599
    4.3959    4.5451    4.5834    4.5764    4.5318
    4.4737    4.6298    4.6762    4.6793    4.6471
    4.5337    4.6947    4.7470    4.7570    4.7333
    4.9027    5.0880    5.1680    5.2099    5.2235
    5.0841    5.2782    5.3679    5.4207    5.4464
    5.1895    5.3881    5.4826    5.5407    5.5722
    5.2641    5.4655    5.5631    5.6245    5.6596
    5.3218    5.5253    5.6251    5.6889    5.7266

V1 =
   -1.6634    2.0868    3.2076    4.2104    5.2235
    0.4204    2.3871    3.4241    4.4489    5.4464
    0.8823    2.5474    3.5642    4.5834    5.5722
    1.1339    2.6559    3.6697    4.6793    5.6596
    1.3028    2.7375    3.7492    4.7570    5.7266
```

Grid for capital k, policy function P and $k' = Pk$:

```
k =
    0.8750
    1.7500
    2.6250
    3.5000
    4.3750

P =                                         P*k =
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     1     0     0     0     0              0.8750
     0     1     0     0     0              1.7500
     0     1     0     0     0              1.7500
     0     1     0     0     0              1.7500
     0     0     1     0     0              2.6250
     0     0     1     0     0              2.6250
     0     0     1     0     0              2.6250
     0     0     0     1     0              3.5000
     0     0     0     1     0              3.5000
     0     0     0     0     1              4.3750
     0     0     0     0     1              4.3750
     0     0     0     0     1              4.3750
     0     0     0     0     1              4.3750
     0     0     0     0     1              4.3750
```