Advanced Macroeconomic Theory
Alan Olivi

Problem Set 1

1 Numerical Solution for the Neoclassical Growth Model

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.

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

$$V(k) = \max_{k' \leq f(k)} U\left(f(k) - k'\right) + \beta V(k'). \tag{1}$$

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: $k' = \alpha\beta A k^{\alpha}$.

By approximating the continuous state space of k by a finite set $K = \{k_1, k_2, ..., k_n\}$ we approximate the Bellman equation by

$$\tilde{V}(i) = \max_{j=1,2,...,n} U\left(f(k_i) - k_j\right) + \beta\tilde{V}(j), \tag{2}$$

where $\tilde{V}(i)$ represents the ith element of an “n by 1” vector. Specifically, the continuous function $V(k)$ that maps $[0, \overline{k}]$ into $\mathbb{R}$ is approximated by a function $\tilde{V}(i)$ which maps $\{1, 2, ..., n\}$ into $\mathbb{R}$.
We are going to solve this discretized Bellman equation numerically 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 1.

Our discretization will be $k_i = 0.8 + (i-1)\kappa$. We choose $\kappa = 0.01$ with $n = 41$ so that $k_1 = 0.8$ and $k_n = 1.2$ (We know that the steady state value of k is 1, so we can the problem for k_i around 1).

1. Write a Matlab program that numerically solves the fixed point of the discretized Bellman equation **??** using the value function iteration algorithm:

   (a) Set the initial guess $\tilde{V}^0 = \mathbf{0}_n$, where $\mathbf{0}_n$ is an "n by 1" vector where each element is a scalar 0.

      i. Given $\tilde{V}^l$ (starting with $\tilde{V}^0$), compute the next guess guess

$$\tilde{V}^{l+1}(i) = \max_{j=1,2,\ldots,n} U\left(f(k_i) - k_j\right) + \beta\tilde{V}^l(j),$$

      for $i = 1, 2, ..., n$.

      - One way to do this is to use a "do-loop" twice. For $i = 1$, compute $U\left(f(k_i) - k_j\right) + \beta\tilde{V}^l(j)$ for $j = 1, 2, ..., n$ and store their values in an "n by 1" vector. Then, find the max among those values. If the mth element is the max, then $\tilde{V}^{l+1}(1) = U\left(f(k_1) - k_m\right) + \beta\tilde{V}^l(m)$. This is the inner loop. Repeat this for $i = 2, 3, ..., n$("outer loop") to get $\{\tilde{V}^{l+1}(1), \tilde{V}^{l+1}(2), \tilde{V}^{l+1}(3), ..., \tilde{V}^{l+1}(n)\}$ and store them in an "n by 1" vector; this is the function $\tilde{V}^{l+1}$.
      - 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, first create an "n by n" matrix we will call the utility matrix $\mathbf{U}$, with the $(i, j)th$ element given by $\mathbf{U}(i, j) \equiv U\left(f(k_i) - k_j\right)$. That is, the $(i, j)th$ element contains the utility associated with having current capital k_i and choosing next period capital k_j. Second, compute an "n by n" matrix $W^{l+1} = \mathbf{U} + \beta\mathbf{1}_n \otimes \left(\tilde{V}^l\right)'$, where $\mathbf{1}_n$ is an "n by 1" vector of ones, $'$ is a transpose, and $\otimes$ is the Kronecker product.[1] Finally, for each *row*, take the max across column to compute an "m by 1" vector $\tilde{V}^{l+1}$. You can do this using just one command.[2]

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

      iii. Compute the policy function, P, which is an "n by n" matrix of which the (i, j)th element is 1 if $j = j^*(i) \equiv \arg\max_{j' \in \{1,2,\ldots,n\}} U\left(f(k_i) - k'_{j'}\right) + \beta\tilde{V}^*(j')$, 0 otherwise. [3] This will be a policy function because when multiplied by the vector of current capital the result is the optimal capital for next period: $k_{j*(i)} = P \times k_i$.

[1]In Matlab, you can use the command `W=U+beta*kron(ones(n,1),V0')`.
[2]In Matlab, you can use the command `V1=max(W,[],2)`.
[3]In Matlab, look at the help for the `max` command to see how to do this in one step.

2. Plot a graph of the computed policy function with k_i on the x-axis and $k_{j*(i)}$ on the y-axis.[4] On the same graph plot k_i against the true policy function ($k_i = \alpha\beta A k_i^{\alpha}$) to see whether the discretized Bellman equation (**??**) approximates the Bellman equation (**??**) well.

3. Try the same program with (i) $\kappa = 0.1$ and $n = 5$ and (ii) $\kappa = 0.001$ and $n = 401$. Plot all of the policy functions together on the same graph with k_i on the x-axis and $k_{j*(i)}$ on the y-axis.

Hint Below I have printed, for the case of $\kappa = 0.1$ and $n = 5$, the numerical values of the utility matrix $\mathbf{U}$, the initial guess $\tilde{V}^0 = \mathbf{0}_n$, the first five iterations of W^{l+1} and $\tilde{V}^{l+1}$, and the approximate policy function P (along with k and $P \times k$).

```
U =                                                                   V0 =

    1.1444    1.1121    1.0786    1.0440    1.0082                       0
    1.1812    1.1500    1.1179    1.0846    1.0502                       0
    1.2139    1.1838    1.1527    1.1206    1.0874                       0
    1.2433    1.2141    1.1839    1.1528    1.1207                       0
    1.2700    1.2416    1.2122    1.1820    1.1509                       0
```

First Iteration:

```
W =                                                                   V1 =

    1.1444    1.1121    1.0786    1.0440    1.0082                  1.1444
    1.1812    1.1500    1.1179    1.0846    1.0502                  1.1812
    1.2139    1.1838    1.1527    1.1206    1.0874                  1.2139
    1.2433    1.2141    1.1839    1.1528    1.1207                  1.2433
    1.2700    1.2416    1.2122    1.1820    1.1509                  1.2700
```

Second Iteration:

```
W =                                                                   V1 =

    2.2430    2.2460    2.2440    2.2376    2.2274                  2.2460
    2.2799    2.2840    2.2832    2.2782    2.2695                  2.2840
    2.3126    2.3177    2.3180    2.3142    2.3067                  2.3180
    2.3420    2.3480    2.3493    2.3464    2.3400                  2.3493
    2.3687    2.3755    2.3776    2.3756    2.3701                  2.3776
```

Third Iteration:

[4] In Matlab, look at the help for the `plot` command.

```
W =                                                        V1 =
   3.3006    3.3047    3.3039    3.2993    3.2907             3.3047
   3.3374    3.3427    3.3432    3.3399    3.3327             3.3432
   3.3701    3.3764    3.3780    3.3759    3.3699             3.3780
   3.3995    3.4067    3.4092    3.4081    3.4032             3.4092
   3.4262    3.4342    3.4376    3.4373    3.4334             3.4376
```

Fourth Iteration:

```
W =                                                        V1 =
   4.3169    4.3215    4.3215    4.3169    4.3082             4.3215
   4.3537    4.3595    4.3608    4.3575    4.3503             4.3608
   4.3864    4.3932    4.3956    4.3935    4.3875             4.3956
   4.4159    4.4235    4.4268    4.4257    4.4208             4.4268
   4.4426    4.4510    4.4551    4.4549    4.4509             4.4551
```

Fifth Iteration:

```
W =                                                        V1 =
   5.2931    5.2984    5.2984    5.2937    5.2851             5.2984
   5.3299    5.3364    5.3376    5.3344    5.3272             5.3376
   5.3626    5.3701    5.3724    5.3703    5.3644             5.3724
   5.3920    5.4004    5.4037    5.4026    5.3977             5.4037
   5.4187    5.4279    5.4320    5.4318    5.4278             5.4320
```

Approximate Policy Function:

```
P =   0     1     0     0     0
      0     0     1     0     0
      0     0     1     0     0
      0     0     1     0     0
      0     0     1     0     0


k = 0.8000
    0.9000
    1.0000
    1.1000
    1.2000
```

```
P*k = 0.9000
      1.0000
      1.0000
      1.0000
      1.0000
```