Dynamic Programming Lab

July 7, 2019

0.0.1 Dynamic Programming

Author: Alina Ozhegova Date: 07/06/0219

0.0.2 Asset Market Equilibrium

- 2 periods
- 4 states of the world: $s = 1, 2, 3, 4$
- Endowment of the first agent: $e^1 = (1, 1, 2, 1, 2)$
- Endowment of the first agent: $e^2 = (1, 3, 1, 3, 1)$;
- Asset 1 payoff: $(1, 1, 1, 1)$
- Asset 2 payoff: $(1, 1, 1.5.1.5)$ \

The utility function of agents looks as follows:

$$\max_{\theta_1,\theta_2} U(c) = v(c_0) + \frac{1}{4}\sum_{i=1}^{4} v(c_i)$$

where $v(c) = \dfrac{c^{1-\gamma}}{1-\gamma}$

Every agent chooses the amount of assets to hold that maximize his utility:

$$\max_{\theta_1,\theta_2} U(c^z) = v(c_0) + \frac{1}{4}\sum_{i=1}^{4} v(c_i)$$

where $v(c) = \dfrac{c^{1-\gamma}}{1-\gamma}$

For all states in s, agents maximize their utility and choose over the amount of assets to hold: (θ_1^h, θ_2^h)

$$\max_{\theta_1^h,\theta_2^h} U(c^h) = v(c_0^h) + E[v(c_s^h)]$$

s.t.

$$c_0^h = e_0^h - q_1 * \theta_1^h - q_2 * \theta_2^h$$
$$c_s^h = e_s^h + A_s^1 * \theta_1^h + A_s^2 * \theta_2^h$$

The first order conditions:

$$-q_1 v'(c_0^1) + E[v'(c_s^1)A_s^1)] = 0$$
$$-q_1 v'(c_0^2) + E[v'(c_s^2)A_s^1)] = 0$$
$$-q_2 v'(c_0^1) + E[v'(c_s^1)A_s^2)] = 0$$
$$-q_2 v'(c_0^2) + E[v'(c_s^2)A_s^2)] = 0$$

Market clearing conditions:

$$\theta_1^1 + \theta_1^2 = 0$$
$$\theta_1^2 + \theta_2^2 = 0$$

```
In [9]: from scipy.optimize import fsolve
        import numpy as np

In [10]: gamma = 2

In [11]: def Eq(x):
             res = np.zeros(6)
             theta11 = x[0]
             theta12 = x[1]
             theta21 = x[2]
             theta22 = x[3]
             q1 = x[4]
             q2 = x[5]
             c01 = 1 - q1 * theta11 - q2 * theta21
             c02 = 1 - q1 * theta12 - q2 * theta22
             c11 = 1 + theta11 + theta21
             c12 = 3 + theta12 + theta22
             c21 = 2 + theta11 + theta21
             c22 = 1 + theta12 + theta22
             c31 = 1 + theta11 + 1.5 * theta21
             c32 = 3 + theta12 + 1.5 * theta22
             c41 = 2 + theta11 + 1.5 * theta21
             c42 = 1 + theta12 + 1.5 * theta22
             res[0] = - q1 * c01**(-gamma) + 0.25 * (c11**(-gamma) + c21**(-gamma) + c31**(-gamm
             res[1] = - q1 * c02**(-gamma) + 0.25 * (c12**(-gamma) + c22**(-gamma) + c32**(-gamm
             res[2] = - q2 * c01**(-gamma) + 0.25 * (c11**(-gamma) + c21**(-gamma) + 1.5 * c31**
             res[3] = - q2 * c02**(-gamma) + 0.25 * (c12**(-gamma) + c22**(-gamma) + 1.5 * c32**
             res[4] = theta11 + theta12
             res[5] = theta21 + theta22
             return res

In [19]: gamma = 2
         fsolve(Eq, [0.1, -0.1, 0.1, -0.1, 1, 1])

Out[19]: array([  1.95406655e-02,  -1.95406655e-02,   1.18686543e-11,
                 -1.18686543e-11,   5.89777656e-01,   7.37222070e-01])

In [20]: gamma = 4
         fsolve(Eq, [0.1, -0.1, 0.1, -0.1, 1, 1])

Out[20]: array([  4.02982364e-03,  -4.02982364e-03,   1.00341657e-12,
                 -1.00341657e-12,   5.18661303e-01,   6.48326628e-01])

In [21]: gamma = 8
         fsolve(Eq, [0.1, -0.1, 0.1, -0.1, 1, 1])
```

```
Out[21]: array([  1.56094152e-04,  -1.56094152e-04,  -3.63402314e-12,
                  3.63402314e-12,   5.01014401e-01,   6.26268001e-01])
```

```
In [24]: gamma = 166
         fsolve(Eq, [0.1, -0.1, 0.1, -0.1, 1, 1])
         #It seems that increasing gamma the second assets gets less desired
```

```
/anaconda3/lib/python3.6/site-packages/scipy/optimize/minpack.py:161: RuntimeWarning: The iterat
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

```
Out[24]: array([  2.77686917e-02,  -1.11099450e-02,  -8.37865820e-04,
                  2.20259144e-02,   2.00466424e+21,  -3.49196270e+21])
```

0.0.3 Ramsey I

```
In [48]: from matplotlib import pyplot as plt
```

For N = 50

```
In [49]: beta = .9
         N = 50
         k_low = .1
         k_high = 10
         k_grid = np.linspace(k_low, k_high, N).reshape(1, N)
         V_init = np.zeros((2, N))
```

```
In [50]: u = lambda c: np.log(c)
```

```
In [52]: def ActionValue(k_i, V_prev):
             V_prev = np.zeros((2, N))
             k = k_grid[0, k_i]
             action_value = np.zeros_like(V_prev) #(2,N)
             c = np.zeros_like(V_prev)
             c[0,:] = 0.9 * k**0.3 + 0.3 * k - k_grid
             c[1,:] = 1.1 * k**0.3 + 0.9 * k - k_grid
             action_value[c <= 0] = - 999999 # negative consumption
             action_value[c > 0] = u(c[c > 0])
             EV = V_prev.mean(axis=0).reshape(1, N)
             action_value = action_value + beta * EV #(2,N)

             return action_value
```

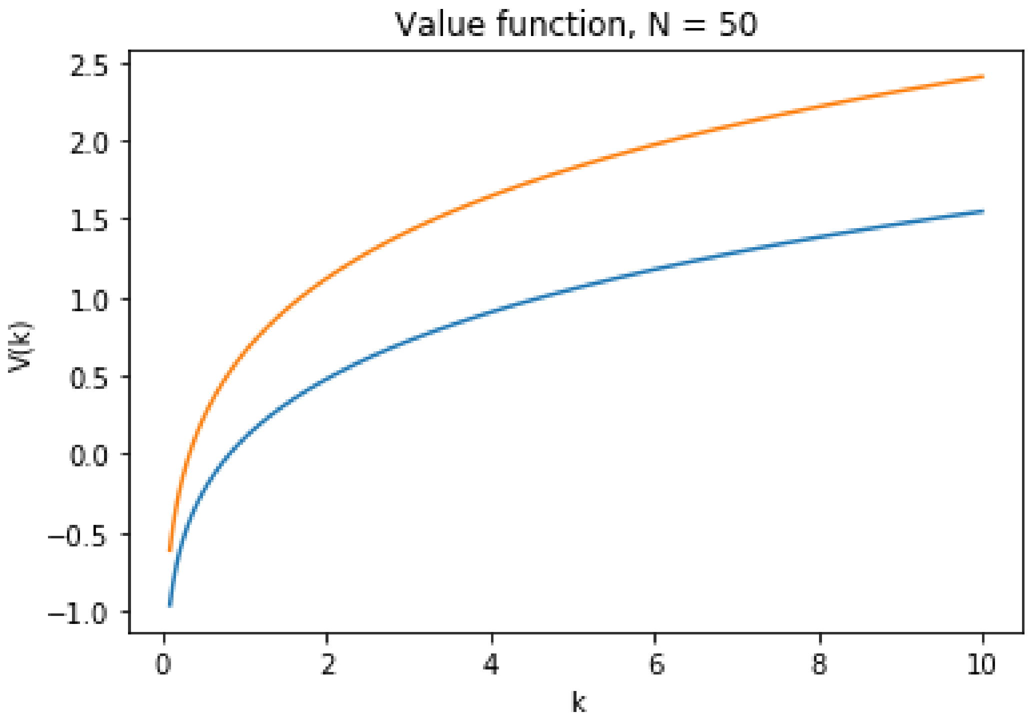
```
In [53]: def ValueFunctionUpdate(i, V_prev):
             V_upd = ActionValue(i, V_prev).max(axis=1)
             return V_upd
```

In [54]:
```
def VFUpdateIteration(V_prev):
    V_upd = np.zeros_like(V_prev)
    for ii in range(V_upd.shape[1]):
        V_upd[:,ii] = ValueFunctionUpdate(ii, V_prev)
    return V_upd
```

In [59]:
```
# For N = 50
MaxIters = 10000
DifList = []
t = 1e-10
PlotInterval = 50
V = V_init.copy()

for i in range(MaxIters):
    V_upd = VFUpdateIteration(V)
    diff = np.max(np.abs(V_upd - V))
    DifList.append(diff)
    V = V_upd.copy()
    print('Iteration: {}'.format(i + 1))
    if diff < t:
        print('Converged after iteration {}'.format(i + 1))
        plt.figure()
        plt.plot(k_grid[0,:], V[0,:], label = 's=1')
        plt.plot(k_grid[0,:], V[1,:], label = 's=2')
        plt.xlabel('k')
        plt.ylabel('V(k)')
        plt.title('Value function, N = 50')
        plt.show();
        break
```

```
Iteration: 1
Iteration: 2
Converged after iteration 2
```



For N = 500

```
In [56]: N = 500
         k_low = .1
         k_high = 10
         k_grid = np.linspace(k_low, k_high, N).reshape(1, N)
         V_init = np.zeros((2, N))
```

```
In [60]: for i in range(MaxIters):
             V_upd = VFUpdateIteration(V)
             diff = np.max(np.abs(V_upd - V))
             DifList.append(diff)
             V = V_upd.copy()
             print('Iteration: {}'.format(i + 1))
             if diff < t:
                 print('Converged after iteration {}'.format(i + 1))
                 plt.figure()
                 plt.plot(k_grid[0,:], V[0,:], label = 's=1')
                 plt.plot(k_grid[0,:], V[1,:], label = 's=2')
                 plt.xlabel('k')
                 plt.ylabel('V(k)')
                 plt.title('Value function, N = 500')
                 plt.show();
                 break
```

```
Iteration: 1
Converged after iteration 1
```