PS3 Solution

January 29, 2019

0.1 PS3

0.1.1 Li Liu

```
In [394]: import matplotlib.pyplot as plt
          import numpy as np
          import scipy.optimize as opt
          from scipy.stats import norm
          import sympy as sp
          from mpl_toolkits.mplot3d import Axes3D
```

5.1. T=1 The optimal amount is to eat the whole cake if the individual only lives for one period. Equivalently,

$$\max_{W_2 \in [0, W_1]} u(W_1 - W_2)$$

and the optimal W_2 would be zero.

5.2. T=2 The condition that characterizes the optimal amount of cake to leave for the next period W_3 in period 2 is:

$$\max_{W_3 \in [0, W_2]} u(W_2 - W_3)$$

The condition that characterizes the optimal amount of cake to leave for the next period W_2 in period 1 is:

$$\max_{W_2 \in [0, W_1]} [u(W_1 - W_2) + \max_{W_3 \in [0, W_2]} \beta u(W_2 - W_3)]$$

5.3. T=3 The conditions that characterizes the optimal amount of cake to leave for the next period in period W_2:

$$\max_{W_2 \in [0, W_1]} \{u(W_1 - W_2) + \max_{W_3 \in [0, W_2]} \beta[u(W_2 - W_3) + \max_{W_4 \in [0, W_3]} \beta u(W_3 - W_4)]\}$$

in period W_3:

$$\max_{W_3 \in [0, W_2]} \beta[u(W_2 - W_3) + \max_{W_4 \in [0, W_3]} \beta u(W_3 - W_4)]$$

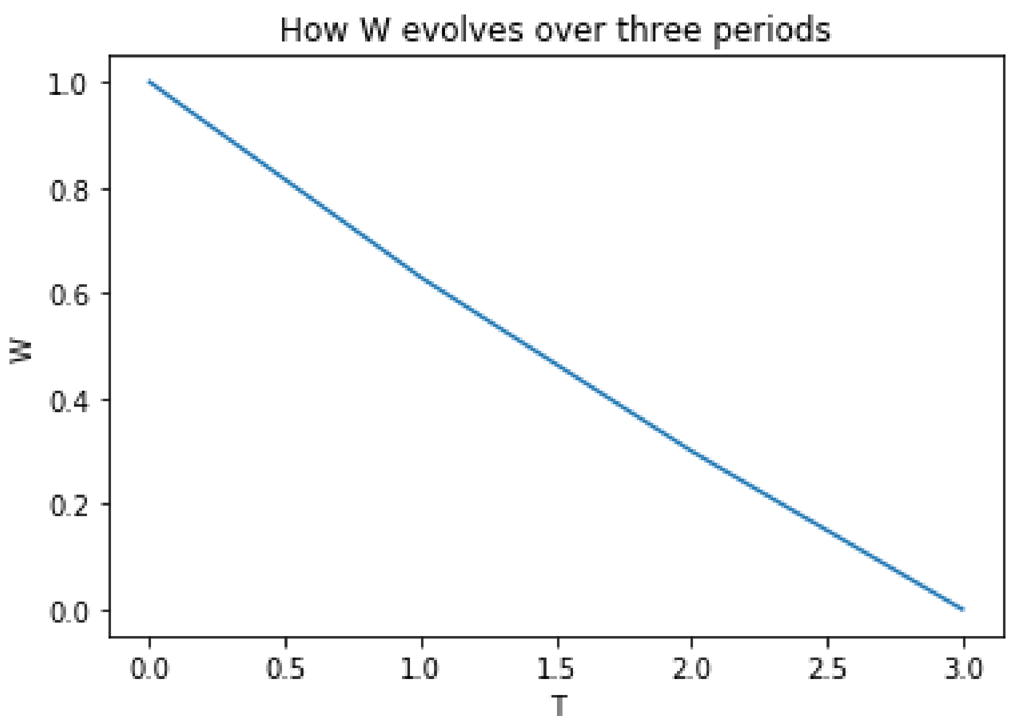
in period W_4:

$$\max_{W_4 \in [0, W_3]} u(W_3 - W_4)$$

Then we differentiate the condition in period W_2 with respect to W_2 and W_3 and set them to 0. With the initial conditions $u(x) = ln(x)$,$W_1 = 1$, $W_4 = 0$, β=0.9, we get the solutions: W_2=0.63, W_3=0.30.

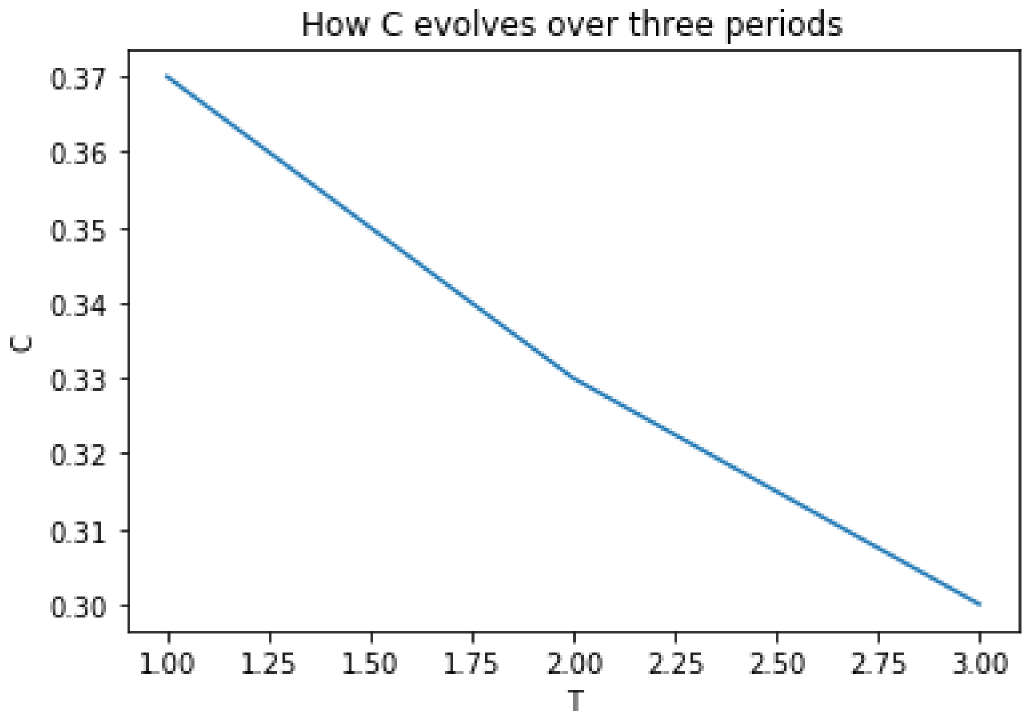
In [338]:
```
W1,W2,W3,W4=1,0.63,0.30,0
c1=W1-W2
c2=W2-W3
c3=W3-W4
W=[W1,W2,W3,W4]
c=[c1,c2,c3]
T=[0,1,2,3]
```

In [339]:
```
plt.plot(T,W)
plt.title("How W evolves over three periods")
plt.xlabel("T")
plt.ylabel("W")
plt.show()
```



In [340]:
```
plt.plot(T[1:],c)
plt.title("How C evolves over three periods")
plt.xlabel("T")
```

```
plt.ylabel("C")
plt.show()
```



5.4 The condition for optimal choice is derived by first-order condition of Eqn 5.7:

$$-u'(W_{T-1} - \psi_{T-1}(W_{T-1})) + \beta u'(\psi_{T-1}(W_{T-1})) = 0$$

Value function V_{T-1} in terms of $\psi_{T-1}(W_{T-1})$:

$$V_{T-1}(W_{T-1}) = \max_{\psi_{T-1}(W_{T-1})} u(W_{T-1} - \psi_{T-1}(W_{T-1})) + \beta V_T(\psi_{T-1}(W_{T-1}))$$

5.5

$$\psi_T(\bar{W}) = W_{T+1} = 0$$
$$\psi_{T-1}(\bar{W}) = W_T \neq 0$$

So

$$\psi_T \neq \psi_{T-1}$$

$$V_T(\bar{W}) = ln(\bar{W})$$
$$V_{T-1}(\bar{W}) = ln(\frac{\bar{W}}{1+\beta}) + \beta ln(\frac{\beta\bar{W}}{1+\beta})$$

So

$$V_T(\bar{W}) \neq V_{T-1}(\bar{W})$$

5.6 Bellman equation at time T-2:

$$V_{T-2}(W_{T-2}) = \max_{W_{T-1}} ln(W_{T-2} - W_{T-1}) + \beta ln(\frac{W_{T-1}}{1+\beta}) + \beta^2 ln(\frac{\beta W_{T-1}}{1+\beta})$$

The solution for how much cake to save in period T-2 is derived by FOC of the above equation:

$$-\frac{1}{(W_{T-2} - \psi_{T-2}(W_{T-2}))} + \beta(1+\beta)\frac{1}{\psi_{T-2}(W_{T-2})} = 0$$

The analytical solution for V_{T-2}:

$$V_{T-2}(W_{T-2}) = ln(\frac{W_{T-2}}{1+\beta+\beta^2}) + \beta ln(\frac{\beta W_{T-2}}{1+\beta+\beta^2}) + \beta^2 ln(\frac{\beta^2 W_{T-2}}{1+\beta+\beta^2})$$

5.7 The analytical solution for $\psi_{T-s}(W_{T-s})$ is:

$$\psi_{T-s}(W_{T-s}) = \frac{\sum_{i=1}^{s} \beta^i}{1 + \sum_{i=1}^{s} \beta^i} W_{T-s}$$

The analytical solution for $V_{T-s}(W_{T-s})$ is:

$$V_{T-s}(W_{T-s}) = \left[\sum_{i=0}^{s} \beta^i ln \left(\frac{\beta^i W_{T-s}}{1 + \sum_{i=1}^{s} \beta^i} \right) \right]$$

As the horizon becomes infinite:

$$\lim_{s\to\infty} \psi_{T-s}(W_{T-s}) = \beta(W_{T-s}) = \psi(W_{T-s})$$

,

$$\lim_{s\to\infty} V_{T-s}(W_{T-s}) = \left(\frac{1}{1-\beta}\right) ln((1-\beta)W_{T-s}) + \frac{\beta}{(1-\beta)^2} ln(\beta) = V(W_{T-s})$$

5.8 Bellman equation in infinite horizon:

$$V(W) = \max_{w\in[0,W]} u(W-w) + \beta V(w)$$

5.9

In [341]:
```
Wmin,Wmax=0.01,1
N=100
W=np.linspace(Wmin,Wmax,N)
W
```

Out[341]:
```
array([0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 , 0.11,
       0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 , 0.21, 0.22,
       0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 , 0.31, 0.32, 0.33,
       0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 , 0.41, 0.42, 0.43, 0.44,
```

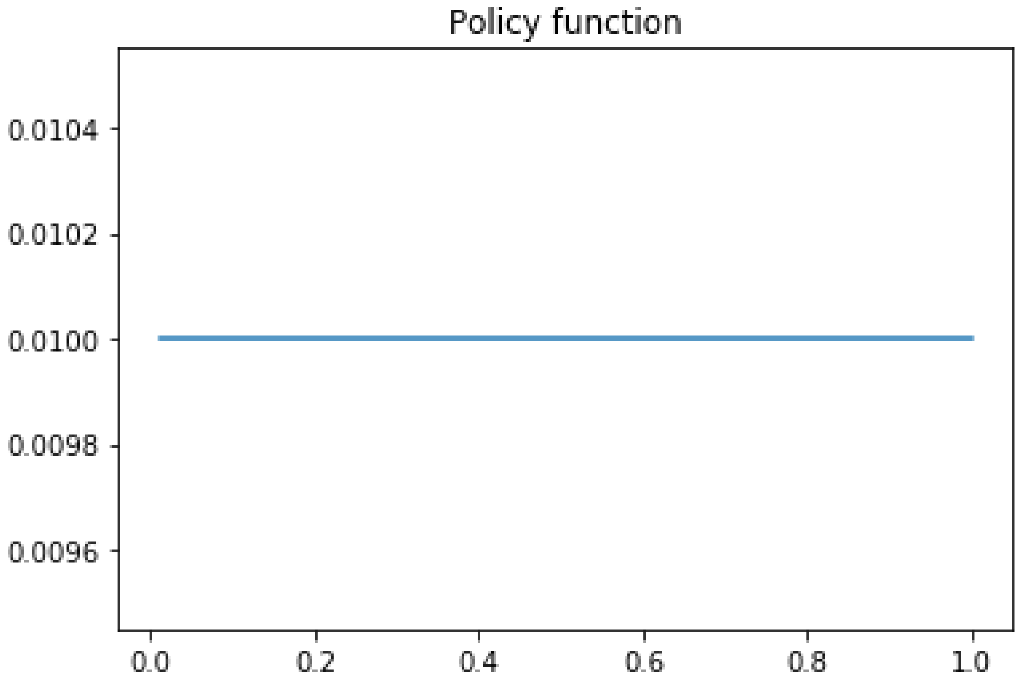
```
           0.45, 0.46, 0.47, 0.48, 0.49, 0.5 , 0.51, 0.52, 0.53, 0.54, 0.55,
           0.56, 0.57, 0.58, 0.59, 0.6 , 0.61, 0.62, 0.63, 0.64, 0.65, 0.66,
           0.67, 0.68, 0.69, 0.7 , 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77,
           0.78, 0.79, 0.8 , 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88,
           0.89, 0.9 , 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99,
           1.  ])
```

5.10

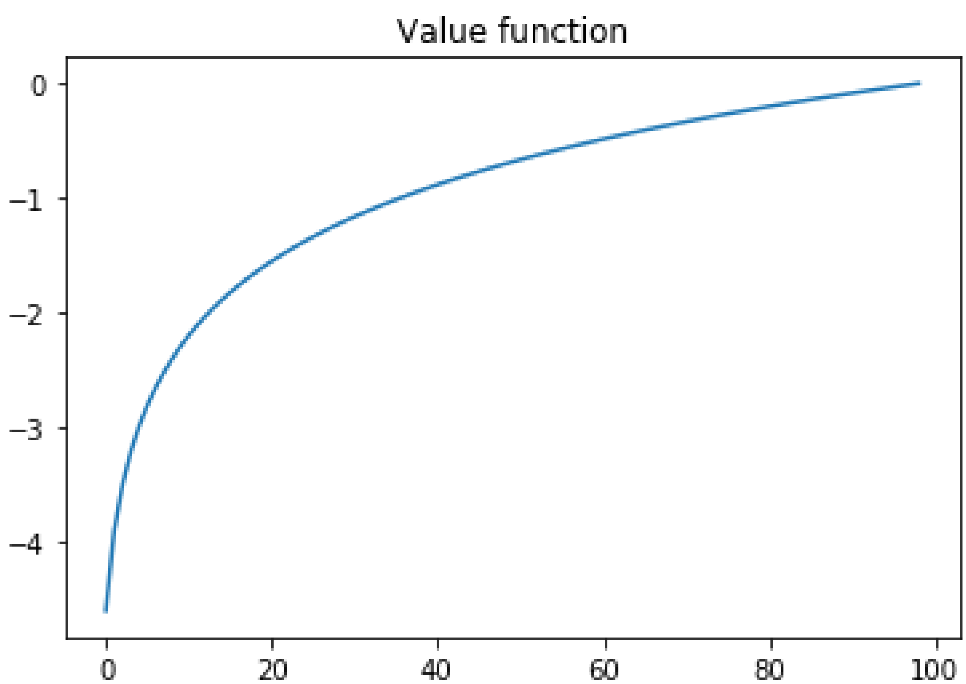
```
In [342]: #Create utility matrix
          def utility(c):
              util=np.zeros_like(c)
              util=np.log(c)
              return util

          #Discount factor
          beta=0.9

          #V_T+1(W) is a column vector of zeros
          V_prime=np.zeros(N).reshape(N,1)

          V_init=utility(W)
          c_mat=(np.tile(W.reshape((N,1)),(1,N)))-(np.tile(W.reshape((1,N)),(N,1)))
          c_pos=c_mat>0
          c_mat[~c_pos]=1e-10
          u_mat=utility(c_mat)
          #Construct the period T value funtion for any W and W'
          VTW=u_mat+beta*V_prime
          #Max over the W' dim (axis=1)
          V_new=VTW.max(axis=1)
          ind=np.argmax(u_mat+beta*V_prime,axis=1)
          W[ind]
```

```
Out[342]: array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                 0.01])
```

```
In [343]: plt.plot(W,W[ind])
          plt.title("Policy function")
          plt.show()
```



In [344]:
```
plt.plot(V_new[1:])
plt.title("Value function")
plt.show()
```



5.11

In [345]:
```
dist=np.sum((V_new-V_init)**2)
dist
```

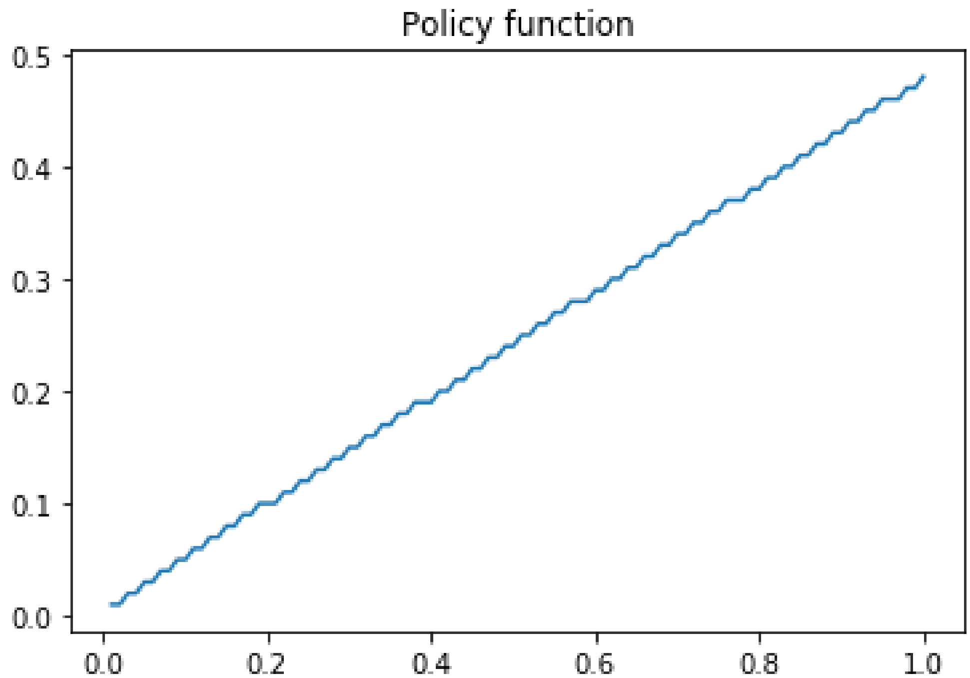
Out[345]: 340.2886682816635

5.12

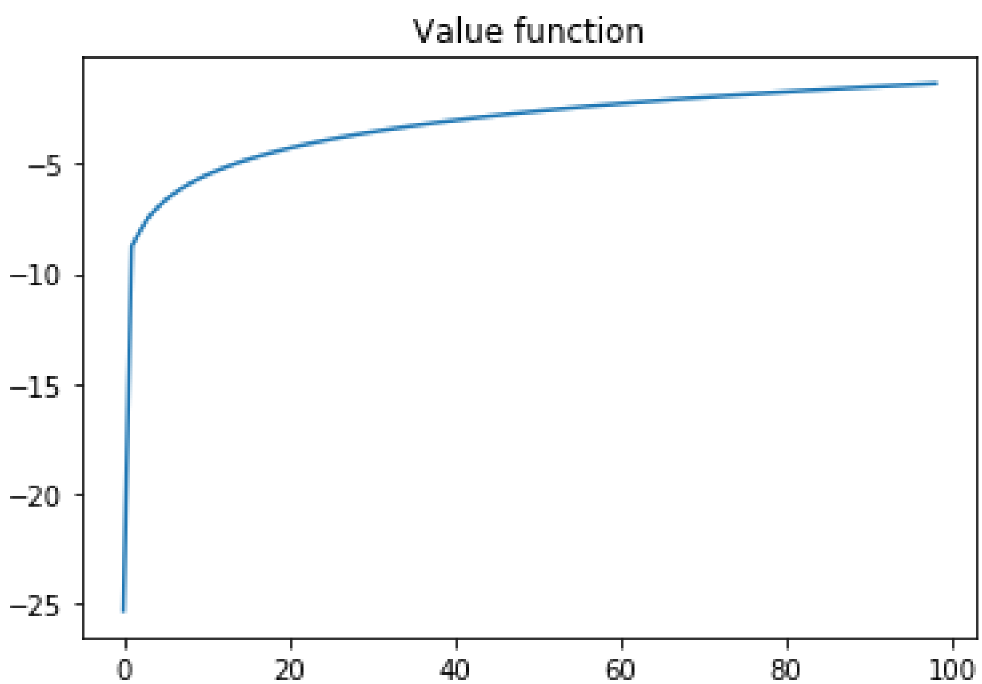
In [346]:
```
#Take V_T from 5.10
V_prime=V_new
V_init=utility(W)
c_mat=(np.tile(W.reshape((N,1)),(1,N)))-(np.tile(W.reshape((1,N)),(N,1)))
c_pos=c_mat>0
c_mat[~c_pos]=1e-10
u_mat=utility(c_mat)
V_prime=np.tile(V_new.reshape((1,N)),(N,1))
V_prime[~c_pos]=-9e+4
#Construct the period T value funtion for any W and W'
VTW=u_mat+beta*V_prime
#Max over the W' dim (axis=1)
V_new_1=VTW.max(axis=1)
ind=np.argmax(u_mat+beta*V_prime,axis=1)
W[ind]
```

Out[346]:
```
array([0.01, 0.01, 0.02, 0.02, 0.03, 0.03, 0.04, 0.04, 0.05, 0.05, 0.06,
       0.06, 0.07, 0.07, 0.08, 0.08, 0.09, 0.09, 0.1 , 0.1 , 0.1 , 0.11,
       0.11, 0.12, 0.12, 0.13, 0.13, 0.14, 0.14, 0.15, 0.15, 0.16, 0.16,
       0.17, 0.17, 0.18, 0.18, 0.19, 0.19, 0.19, 0.2 , 0.2 , 0.21, 0.21,
       0.22, 0.22, 0.23, 0.23, 0.24, 0.24, 0.25, 0.25, 0.26, 0.26, 0.27,
       0.27, 0.28, 0.28, 0.28, 0.29, 0.29, 0.3 , 0.3 , 0.31, 0.31, 0.32,
       0.32, 0.33, 0.33, 0.34, 0.34, 0.35, 0.35, 0.36, 0.36, 0.37, 0.37,
       0.37, 0.38, 0.38, 0.39, 0.39, 0.4 , 0.4 , 0.41, 0.41, 0.42, 0.42,
       0.43, 0.43, 0.44, 0.44, 0.45, 0.45, 0.46, 0.46, 0.46, 0.47, 0.47,
       0.48])
```

In [347]:
```
plt.plot(W,W[ind])
plt.title("Policy function")
plt.show()
```




```
In [348]: plt.plot(V_new_1[1:])
          plt.title("Value function")
          plt.show()
```



```
In [349]: #Distance between the two value functions
          dist1=np.sum((V_new_1-V_new)**2)
          print("Distance at T:",dist)
          print("Distance at T-1:",dist1)
          print("Distance of two value functions goes up.")
```

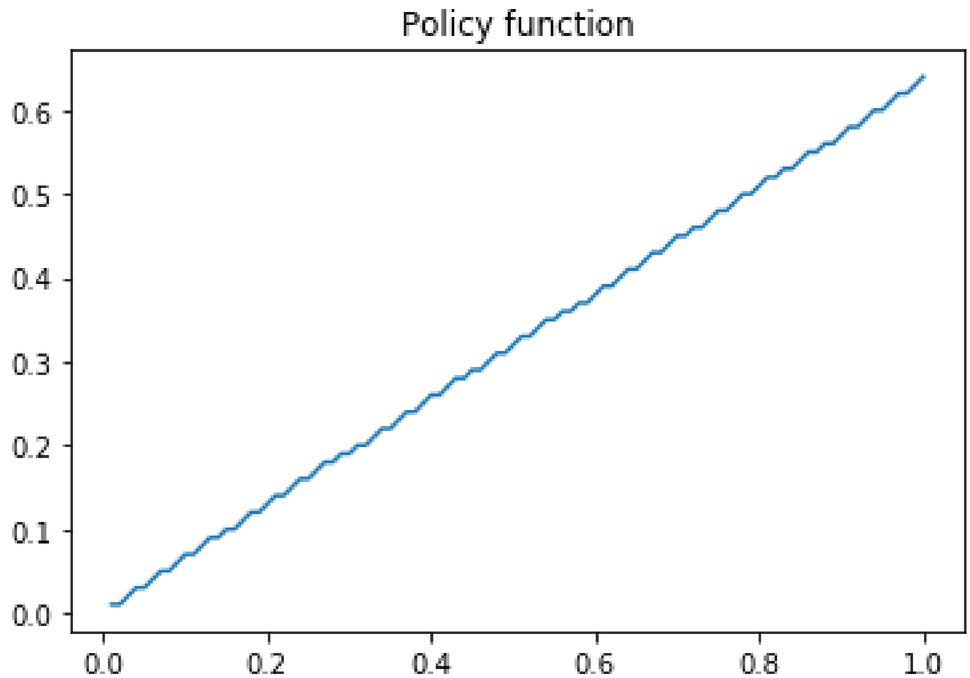
```
Distance at T: 340.2886682816635
Distance at T-1: 6561000945.781889
Distance of two value functions goes up.
```

5.13

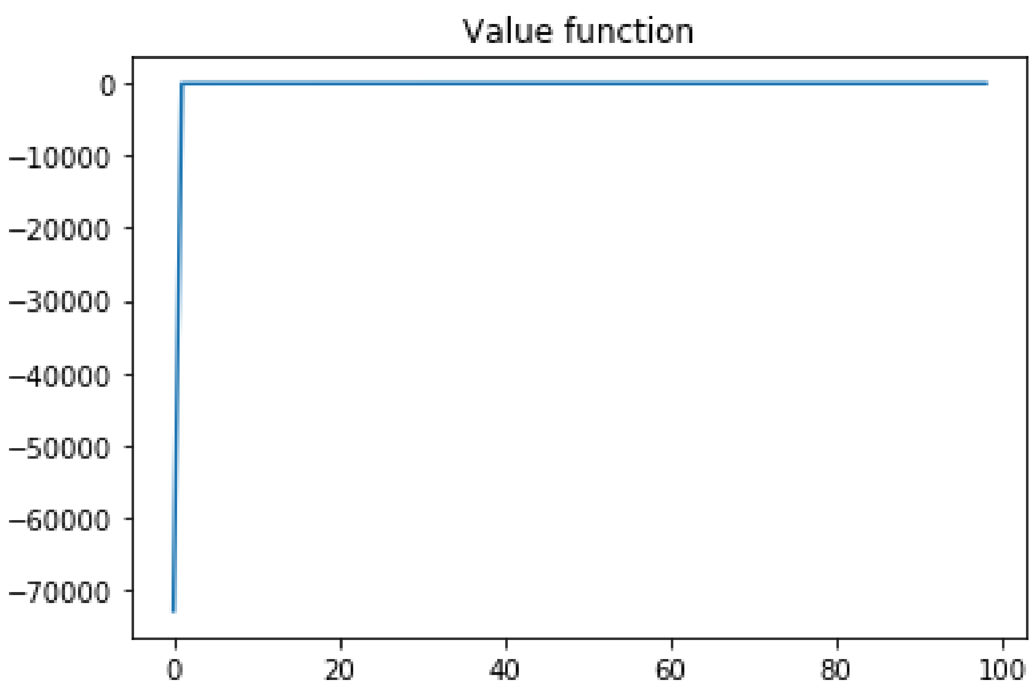
```
In [350]: #Take V_T from 5.10
          V_prime=V_new_1
          V_init=utility(W)
          c_mat=(np.tile(W.reshape((N,1)),(1,N)))-(np.tile(W.reshape((1,N)),(N,1)))
          c_pos=c_mat>0
          c_mat[~c_pos]=1e-10
          u_mat=utility(c_mat)
          V_prime=np.tile(V_new_1.reshape((1,N)),(N,1))
          V_prime[~c_pos]=-9e+4
          #Construct the period T value funtion for any W and W'
          VTW=u_mat+beta*V_prime
          #Max over the W' dim (axis=1)
          V_new_2=VTW.max(axis=1)
          ind=np.argmax(u_mat+beta*V_prime,axis=1)
          W[ind]
```

```
Out[350]: array([0.01, 0.01, 0.02, 0.03, 0.03, 0.04, 0.05, 0.05, 0.06, 0.07, 0.07,
                 0.08, 0.09, 0.09, 0.1 , 0.1 , 0.11, 0.12, 0.12, 0.13, 0.14, 0.14,
                 0.15, 0.16, 0.16, 0.17, 0.18, 0.18, 0.19, 0.19, 0.2 , 0.2 , 0.21,
                 0.22, 0.22, 0.23, 0.24, 0.24, 0.25, 0.26, 0.26, 0.27, 0.28, 0.28,
                 0.29, 0.29, 0.3 , 0.31, 0.31, 0.32, 0.33, 0.33, 0.34, 0.35, 0.35,
                 0.36, 0.36, 0.37, 0.37, 0.38, 0.39, 0.39, 0.4 , 0.41, 0.41, 0.42,
                 0.43, 0.43, 0.44, 0.45, 0.45, 0.46, 0.46, 0.47, 0.48, 0.48, 0.49,
                 0.5 , 0.5 , 0.51, 0.52, 0.52, 0.53, 0.53, 0.54, 0.55, 0.55, 0.56,
                 0.56, 0.57, 0.58, 0.58, 0.59, 0.6 , 0.6 , 0.61, 0.62, 0.62, 0.63,
                 0.64])
```

```
In [351]: plt.plot(W,W[ind])
          plt.title("Policy function")
          plt.show()
```



In [352]:
```
plt.plot(V_new_2[1:])
plt.title("Value function")
plt.show()
```



In [353]:
```
dist2=np.sum((V_new_2-V_new_1)**2)
print("Distance at T:",dist)
print("Distance at T-1:",dist1)
print("Distance at T-2:",dist2)
print("Distance goes down from T-1 to T-2.")
```

```
Distance at T: 340.2886682816635
Distance at T-1: 6561000945.781889
Distance at T-2: 5314410944.0394945
Distance goes down from T-1 to T-2.
```

5.14

In [354]:
```
N=100
c_mat=(np.tile(W.reshape((N,1)),(1,N)))-(np.tile(W.reshape((1,N)),(N,1)))
c_pos=c_mat>0
c_mat[~c_pos]=1e-10
u_mat=utility(c_mat)
dist=10.0
VF_iter=0
maxiters=500
tooler=1e-10
VF_iter=0
V_init=utility(W)

while dist>tooler and VF_iter<maxiters:
    VF_iter+=1
    #One contraction mapping
    V_prime=np.tile(V_init.reshape((1,N)),(N,1))
    V_prime[~c_pos]=-9e+4
    VWT=u_mat+beta*V_prime
    V_new=VWT.max(axis=1)
    ind=np.argmax(VWT,axis=1)
    dist=((V_new-V_init)**2).sum()
    print("Iter=",VF_iter,",distance=",dist)
    V_init=V_new
print("The convergence takes",VF_iter,"iterations")
print("V_new is equal to V_init (Converge to the fixed point)?",np.array_equal(V_ini
```

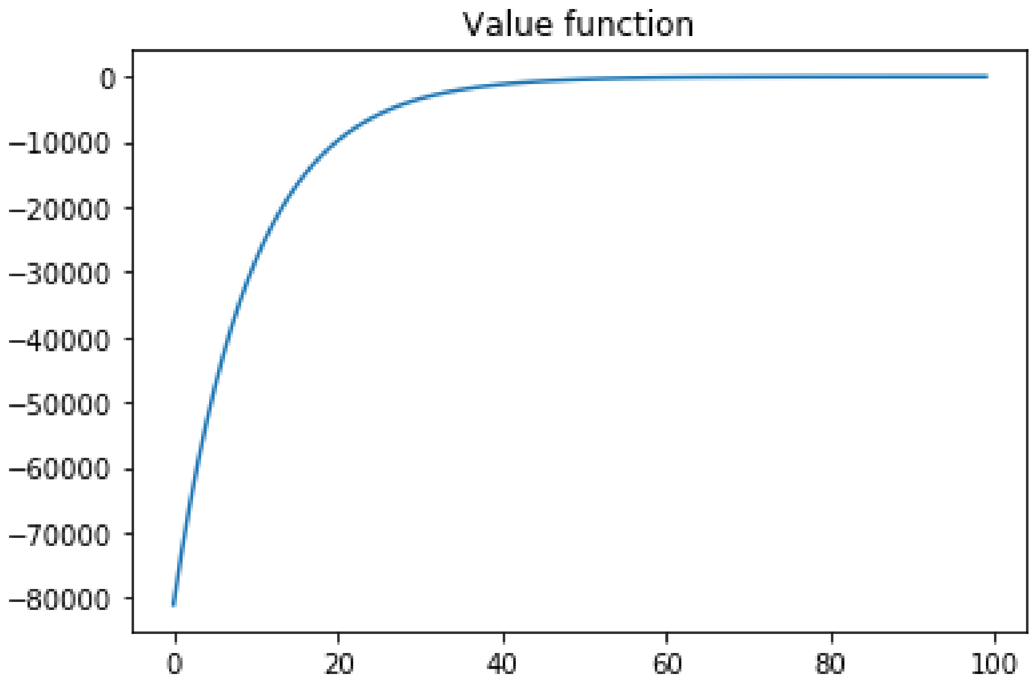
```
Iter= 1 ,distance= 6563985007.657785
Iter= 2 ,distance= 5316828035.491551
Iter= 3 ,distance= 4306630819.958046
Iter= 4 ,distance= 3488371038.8842497
Iter= 5 ,distance= 2825580594.096034
```

```
Iter= 6 ,distance= 2288720319.2797203
Iter= 7 ,distance= 1853863486.7757986
Iter= 8 ,distance= 1501629445.548632
Iter= 9 ,distance= 1216319867.1479275
Iter= 10 ,distance= 985219105.0198653
Iter= 11 ,distance= 798027484.9834925
Iter= 12 ,distance= 646402270.7859683
Iter= 13 ,distance= 523585845.731953
Iter= 14 ,distance= 424104540.31342936
Iter= 15 ,distance= 343524682.05194384
Iter= 16 ,distance= 278254996.15010464
Iter= 17 ,distance= 225386550.10481408
Iter= 18 ,distance= 182563108.40885067
Iter= 19 ,distance= 147876120.28517076
Iter= 20 ,distance= 119779659.61285393
Iter= 21 ,distance= 97021526.32175387
Iter= 22 ,distance= 78587438.24377468
Iter= 23 ,distance= 63655826.797743194
Iter= 24 ,distance= 51561221.43000043
Iter= 25 ,distance= 41764590.99362178
Iter= 26 ,distance= 33829320.258351654
Iter= 27 ,distance= 27401750.88813295
Iter= 28 ,distance= 22195419.630133
Iter= 29 ,distance= 17978291.248549566
Iter= 30 ,distance= 14562417.202229027
Iter= 31 ,distance= 11795559.172756335
Iter= 32 ,distance= 9554404.12031933
Iter= 33 ,distance= 7739068.484603304
Iter= 34 ,distance= 6268646.579869382
Iter= 35 ,distance= 5077604.801231563
Iter= 36 ,distance= 4112860.92786322
Iter= 37 ,distance= 3331418.3610032974
Iter= 38 ,distance= 2698449.8550905357
Iter= 39 ,distance= 2185745.340916173
Iter= 40 ,distance= 1770454.662405837
Iter= 41 ,distance= 1434069.1918269114
Iter= 42 ,distance= 1161596.942174743
Iter= 43 ,distance= 940894.4026855434
Iter= 44 ,distance= 762125.3301672665
Iter= 45 ,distance= 617322.3668386245
Iter= 46 ,distance= 500031.9534910452
Iter= 47 ,distance= 405026.70660177537
Iter= 48 ,distance= 328072.44560896023
Iter= 49 ,distance= 265739.4831122093
Iter= 50 ,distance= 215249.773787078
Iter= 51 ,distance= 174353.09978101993
Iter= 52 ,distance= 141226.7852404473
Iter= 53 ,distance= 114394.46174083267
```

```
Iter= 54 ,distance= 92660.27195996972
Iter= 55 ,distance= 75055.57103803426
Iter= 56 ,distance= 60795.755437120926
Iter= 57 ,distance= 49245.297958901705
Iter= 58 ,distance= 39889.420365982616
Iter= 59 ,distance= 32311.15301448741
Iter= 60 ,distance= 26172.749416523955
Iter= 61 ,distance= 21200.63621759014
Iter= 62 ,distance= 17173.217133022226
Iter= 63 ,distance= 13911.001395746944
Iter= 64 ,distance= 11268.599915507328
Iter= 65 ,distance= 9128.248415880727
Iter= 66 ,distance= 7394.556568738393
Iter= 67 ,distance= 5990.259783037198
Iter= 68 ,distance= 4852.771512109664
Iter= 69 ,distance= 3931.398197437206
Iter= 70 ,distance= 3185.07882150994
Iter= 71 ,distance= 2580.5520350857705
Iter= 72 ,distance= 2090.8780717923887
Iter= 73 ,distance= 1694.2330104831076
Iter= 74 ,distance= 1372.9413650084316
Iter= 75 ,distance= 1112.6869372506203
Iter= 76 ,distance= 901.8712561442742
Iter= 77 ,distance= 731.0989623881655
Iter= 78 ,distance= 592.7623610583149
Iter= 79 ,distance= 480.69981033250417
Iter= 80 ,distance= 389.9174929052662
Iter= 81 ,distance= 316.3696813569722
Iter= 82 ,distance= 256.7824724428508
Iter= 83 ,distance= 208.50473663213486
Iter= 84 ,distance= 169.37992660717697
Iter= 85 ,distance= 137.6722666225492
Iter= 86 ,distance= 111.97419578181234
Iter= 87 ,distance= 91.13432753236344
Iter= 88 ,distance= 74.23363961424533
Iter= 89 ,distance= 60.52577469850647
Iter= 90 ,distance= 49.386593018756344
Iter= 91 ,distance= 40.245726192931954
Iter= 92 ,distance= 32.12873413001096
Iter= 93 ,distance= 25.543941374558337
Iter= 94 ,distance= 19.768795412191732
Iter= 95 ,distance= 15.156684329052599
Iter= 96 ,distance= 11.175532170156538
Iter= 97 ,distance= 8.021189330265935
Iter= 98 ,distance= 5.341950547088503
Iter= 99 ,distance= 3.2354766128598538
Iter= 100 ,distance= 1.4639595217218375
Iter= 101 ,distance= 0.0
```

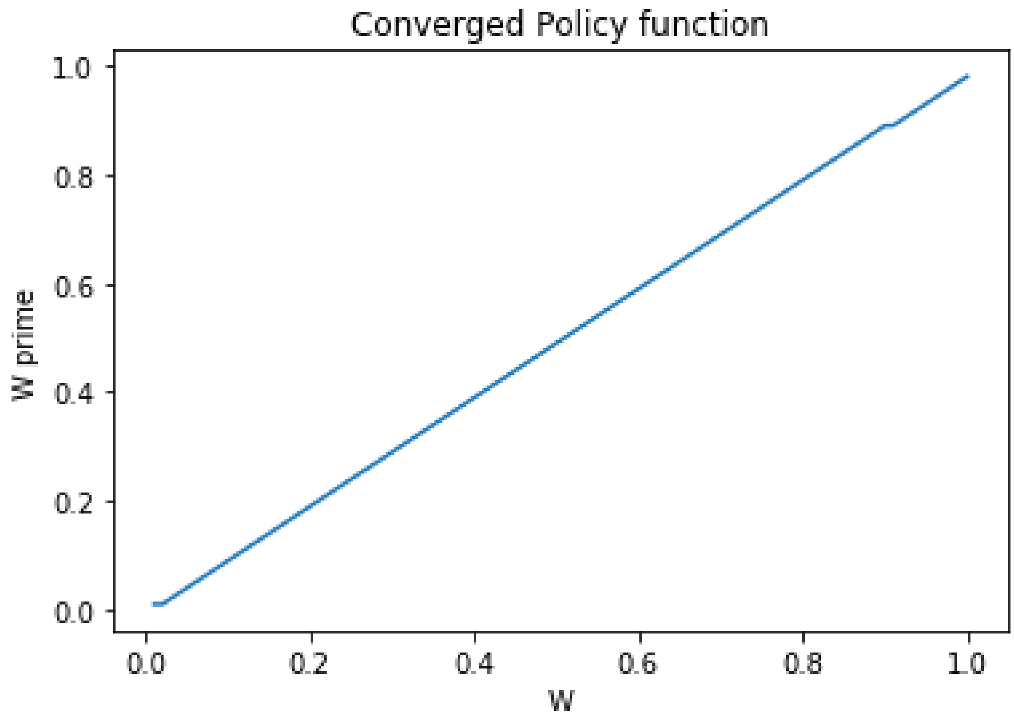
```
The convergence takes 101 iterations
V_new is equal to V_init (Converge to the fixed point)? True
```

```
In [355]: plt.plot(V_new)
          plt.title("Value function")
          plt.show()
```



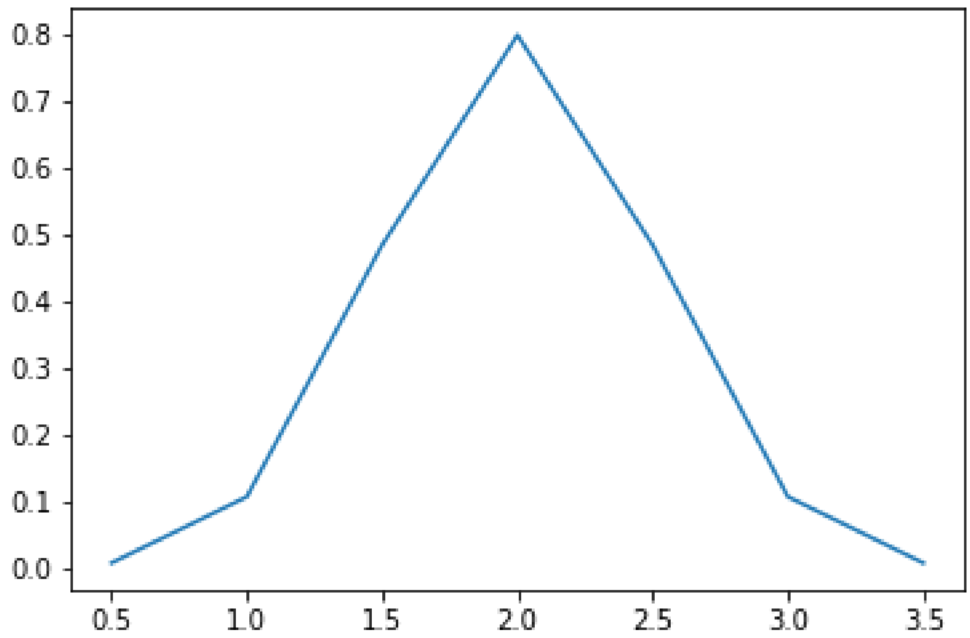
5.15

```
In [356]: plt.plot(W,W[ind])
          plt.title("Converged Policy function")
          plt.xlabel("W")
          plt.ylabel("W prime")
          plt.show()
```



5.16

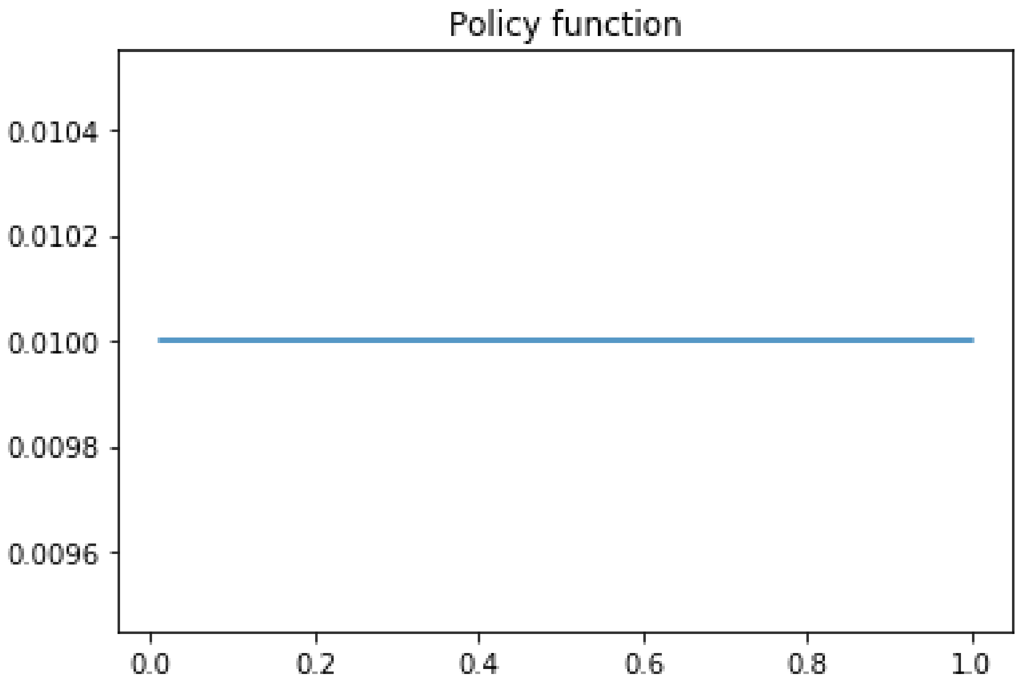
```
In [357]: sigma=0.5
          mu=4*sigma
          M=7
          eps=np.linspace(mu-3*sigma,mu+3*sigma,M)
          eps

Out[357]: array([0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5])

In [358]: PDF = lambda x: norm(loc = mu, scale = sigma).pdf(x)
          pdf = PDF(eps)
          pdf

Out[358]: array([0.0088637 , 0.10798193, 0.48394145, 0.79788456, 0.48394145,
                 0.10798193, 0.0088637 ])

In [359]: #Probability Distribution over eps
          plt.plot(eps,pdf)
          plt.show()
```



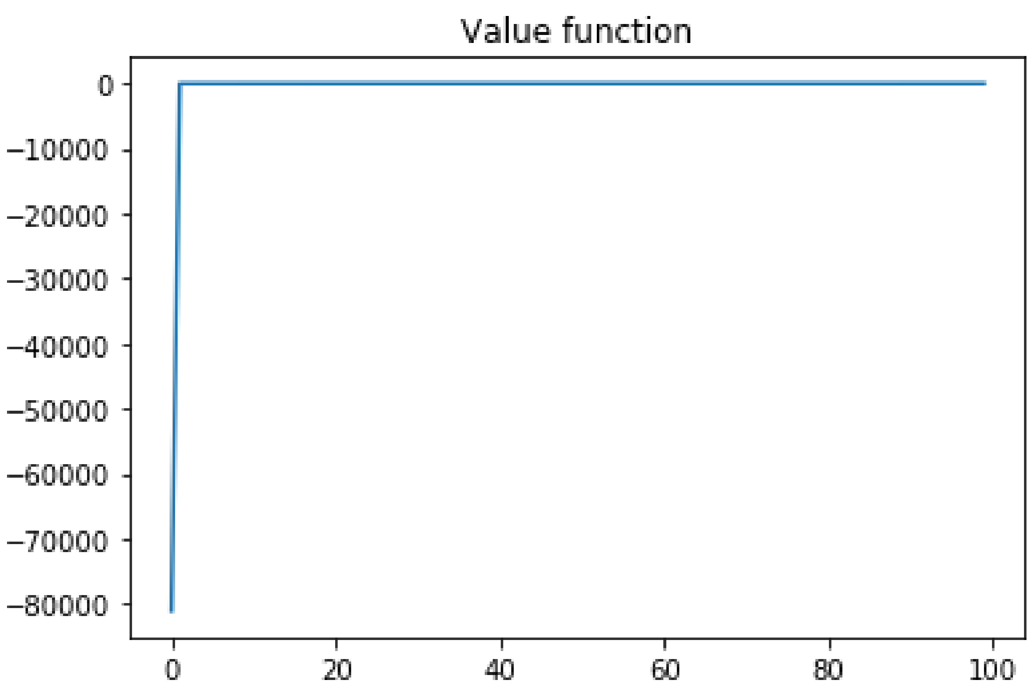
5.17

In [360]:
```
c_mat=np.tile(W.reshape((N,1)), (1,N)) - np.tile(W.reshape((1,N)), (N,1))
c_pos=c_mat>0
c_mat[~c_pos]=1e-10
u_mat=utility(c_mat)
cube = np.array([u_mat*e for e in eps])

V_init=np.zeros((N,M))
EV = V_init @ pdf.reshape((M,1))
EV_mat = np.tile(EV.reshape((1,N)), (N,1))
EV_mat[~c_pos] = -9e+4
EV_cube = np.array([EV_mat for e in range(M)])

VT = cube + beta*EV_cube
V_new = np.zeros((N,M))
W_new = np.zeros((N,M))
for i in range(N):
    VTW = VT[:, i, :]
    V_new[i] = VTW.max(axis=1)
    ind = np.argmax(VTW, axis=1)
    W_new[i] = W[ind]
```

In [361]:
```
plt.plot(W,np.average(W_new,axis=1))
plt.title("Policy function")
plt.show()
```



In [362]:
```
plt.plot(np.average(V_new,axis=1))
plt.title("Value function")
plt.show()
```



5.18

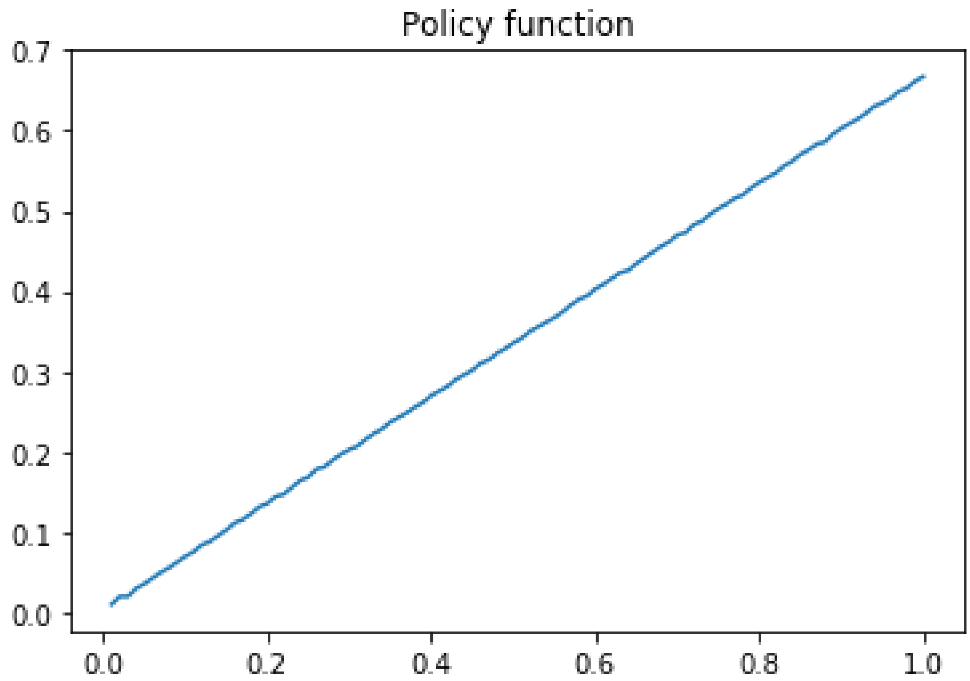
```
In [363]: dist=np.sum((V_init-V_new)**2)
          dist
```

```
Out[363]: 45979247448.96637
```

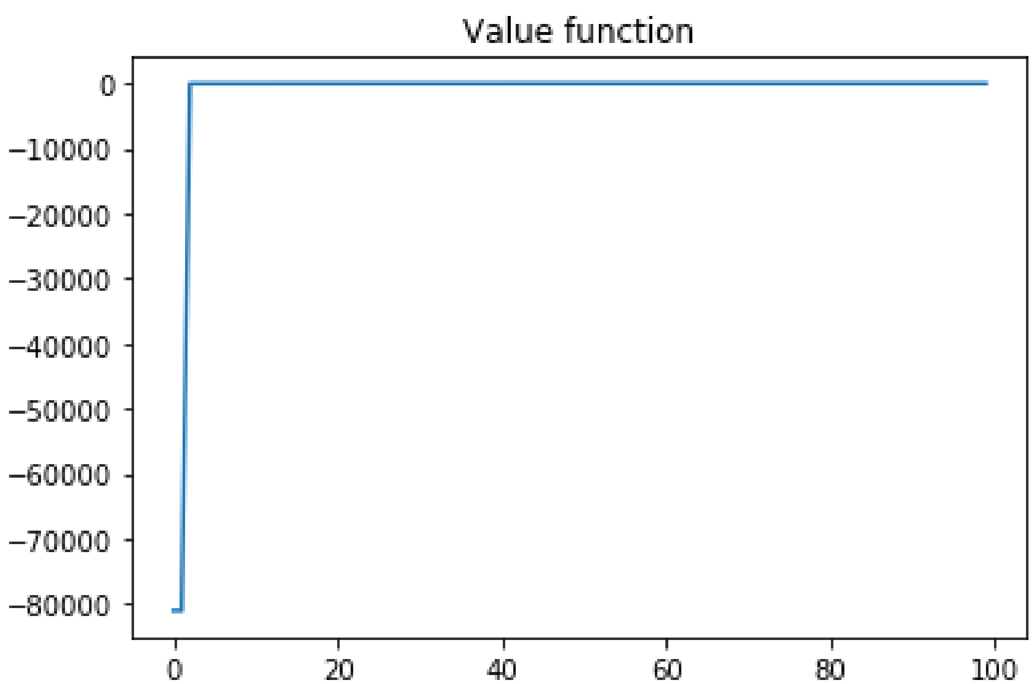
5.19

```
In [364]: V_init=V_new
          EV = V_init @ pdf.reshape((M,1))
          EV_mat = np.tile(EV.reshape((1,N)), (N,1))
          EV_mat[~c_pos] = -9e+4
          EV_cube = np.array([EV_mat for e in range(M)])

          VT = cube + beta*EV_cube
          V_new_1 = np.zeros((N,M))
          W_new_1 = np.zeros((N,M))
          for i in range(N):
              VTW = VT[:, i, :]
              V_new_1[i] = VTW.max(axis=1)
              ind = np.argmax(VTW, axis=1)
              ind_list.append(ind)
              W_new_1[i] = W[ind]
```

```
In [365]: plt.plot(W,np.average(W_new_1,axis=1))
          plt.title("Policy function")
          plt.show()
```



In [366]:
```
plt.plot(np.average(V_new_1,axis=1))
plt.title("Value function")
plt.show()
```



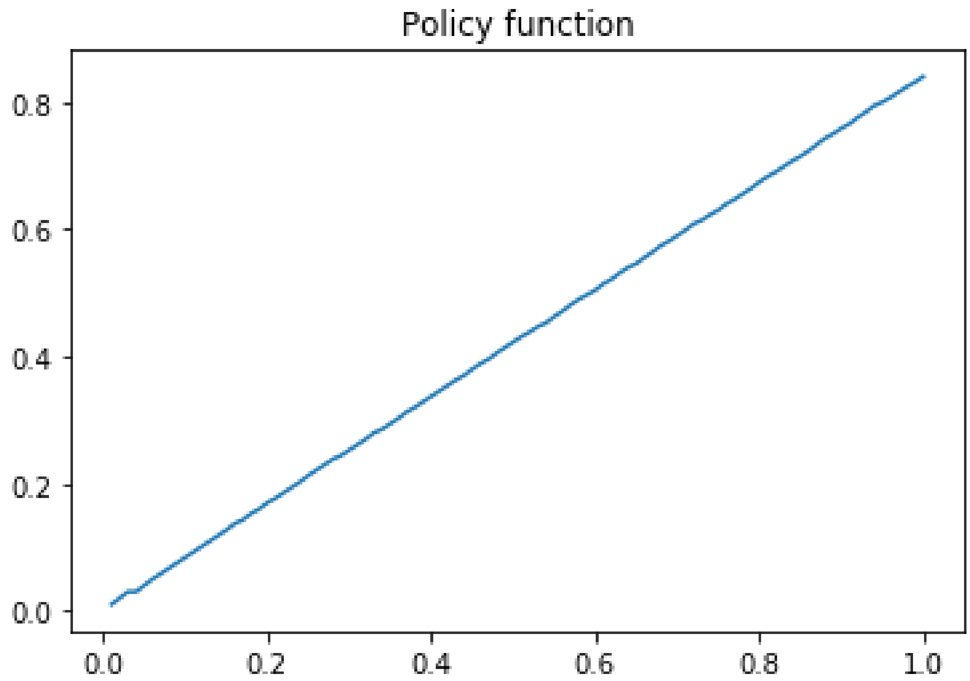
```
In [367]: dist1=np.sum((V_new_1-V_new)**2)
          print("Distance at T:",dist)
          print("Distance at T-1:",dist1)
          print("Distance of two value functions goes down.")
```

```
Distance at T: 45979247448.96637
Distance at T-1: 45968829677.923256
Distance of two value functions goes down.
```

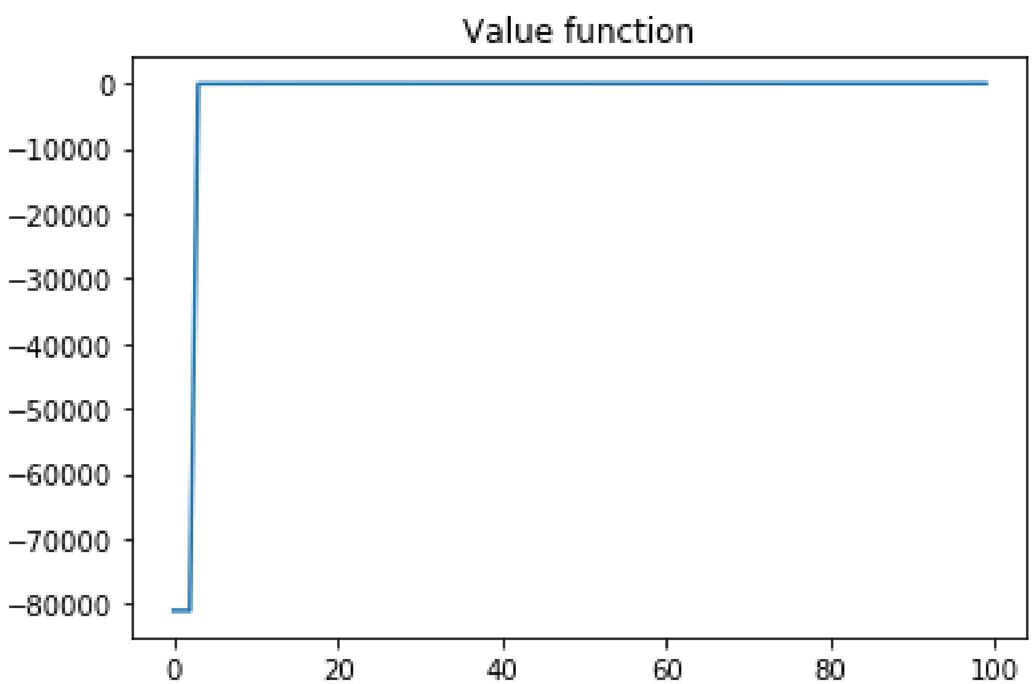
5.20

```
In [368]: V_init=V_new_1
          EV = V_init @ pdf.reshape((M,1))
          EV_mat = np.tile(EV.reshape((1,N)), (N,1))
          EV_mat[~c_pos] = -9e+4
          EV_cube = np.array([EV_mat for e in range(M)])

          VT = cube + beta*EV_cube
          V_new_2 = np.zeros((N,M))
          W_new_2 = np.zeros((N,M))
          for i in range(N):
              VTW = VT[:, i, :]
              V_new_2[i] = VTW.max(axis=1)
              ind = np.argmax(VTW, axis=1)
              ind_list.append(ind)
              W_new_2[i] = W[ind]
```

```
In [369]: plt.plot(W,np.average(W_new_2,axis=1))
          plt.title("Policy function")
          plt.show()
```



In [370]:
```
plt.plot(np.average(V_new_2,axis=1))
plt.title("Value function")
plt.show()
```



```
In [371]: dist2=np.sum((V_new_2-V_new_1)**2)
          print("Distance at T:",dist)
          print("Distance at T-1:",dist1)
          print("Distance at T-2:",dist2)

          print("Distance of two value functions goes down.")
```

```
Distance at T: 45979247448.96637
Distance at T-1: 45968829677.923256
Distance at T-2: 45950143416.50976
Distance of two value functions goes down.
```

5.21

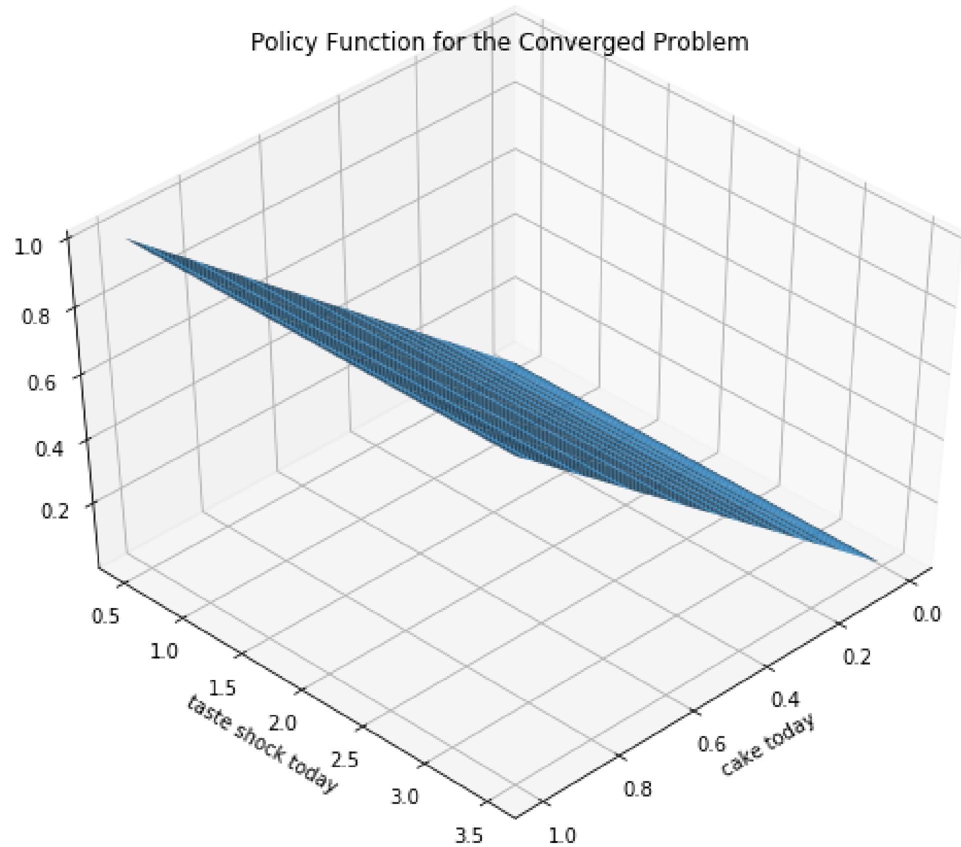
```
In [381]: VF_iter = 0
          dist=10
          V_init = np.zeros((N,M))
          #V_new = np.zeros((N,M))

          while dist>toler and VF_iter<maxiters:
              VF_iter += 1
              EV = V_init @ pdf.reshape((M,1)) #(W', 1)
              EV_mat = np.tile(EV, (1,N))
              EV_mat[~c_pos] = -9e+4
              EV_cube = np.array([EV_mat for e in range(M)])
              V = eu_cube + beta*EV_cube
              V_new = np.zeros((N,M))
              W_new = np.zeros((N,M))
              for i in range(N):
                  VTW = V[:, i, :]
                  V_new[i] = VTW.max(axis=1)
                  ind = np.argmax(VTW, axis=1)
                  W_new[i] = W[ind]
              dist = np.sum((V_init-V_new)**2)
              print('Iter=', VF_iter, ', distance= ', dist)
              V_init = V_new
          print("The convergence takes",VF_iter,"iterations")
          print("V_new is equal to V_init (Converge to the fixed point)?",np.array_equal(V_ini
```

```
Iter= 1 , distance=  45979247448.96637
Iter= 2 , distance=  16223.39177231219
Iter= 3 , distance=  52535.341389860754
Iter= 4 , distance=  170122.3846211698
Iter= 5 , distance=  550898.2141073309
```

```
Iter= 6 , distance=  1783944.2057108395
Iter= 7 , distance=  5776851.054502134
Iter= 8 , distance=  18706867.624598633
Iter= 9 , distance=  60577448.340394475
Iter= 10 , distance=  196164709.1898902
Iter= 11 , distance=  635229680.1166917
Iter= 12 , distance=  2057030279.134176
Iter= 13 , distance=  6661171072.008981
Iter= 14 , distance=  21570513813.362812
Iter= 15 , distance=  69850640546.92517
Iter= 16 , distance=  154859621684.6649
Iter= 17 , distance=  216072512184.37057
Iter= 18 , distance=  211148526297.82758
Iter= 19 , distance=  163195170454.01764
Iter= 20 , distance=  109344841433.4629
Iter= 21 , distance=  67584715045.2302
Iter= 22 , distance=  39650112914.37802
Iter= 23 , distance=  22718912538.545013
Iter= 24 , distance=  12289589466.281145
Iter= 25 , distance=  6342943585.090034
Iter= 26 , distance=  3055582773.550291
Iter= 27 , distance=  0.0
The convergence takes 27 iterations
V_new is equal to V_init (Converge to the fixed point)? True
```

5.22

```
In [393]: X, Y = np.meshgrid(W, eps)
          new_fig = plt.figure(figsize=(10,8))
          new_ax = new_fig.add_subplot(111, projection='3d')
          new_ax.plot_surface(X.T, Y.T, W_new)
          new_ax.set_xlabel('cake today')
          new_ax.set_ylabel('taste shock today')
          new_ax.set_title("Policy Function for the Converged Problem")
          new_ax.view_init(elev=45,azim=45)
          plt.show()
```



Policy Function for the Converged Problem
1.0
0.8
0.6
0.4
0.2
0.5
1.0
1.5
2.0
2.5
3.0
3.5
taste shock today
0.0
0.2
0.4
0.6
0.8
1.0
cake today