Problem Set 3

Linghui Feng

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sta
from mpl_toolkits.mplot3d import Axes3D
```

Ex.1

T = 1:
the optimal amount of cake to eat is:

$$U = \max_{W_{T+1}} \sum_{t=1}^{T} \beta^{t-1} u(W_t - W_{t+1}) = \max_{W_{t+1}} \; u(W_t)$$

$$W_{T+1} = min(W_{T+1}) = 0 \quad \forall W_T$$

Ex.2

T = 2:

$$U = \max_{W_2, W_3} \sum_{t=1}^{T=2} \beta^{t-1} u(W_t - W_{t+1}) = \max_{W_2, W_3} \; u(W_1 - W_2) + \beta \, u(W_2 - W_3)$$

The optimal amount pf cale to leave for the next period W_3 in period 2 is:

$$W_3 = min(W_3) = 0 \quad \forall W_2, W_1$$

To optimize W_2, consider the first order condition of the utility:

$$\frac{\partial U}{\partial W_2} = 0$$

$$-\frac{\partial u(C_1)}{\partial C_1} + \beta \frac{\partial (C_2)}{\partial C_2} = 0$$

$$\Longrightarrow \quad \frac{\partial u(C_1)}{\partial C_1} = \beta \frac{\partial (C_2)}{\partial C_2}$$

where $C_t = W_t - W_{t+1}$.

Ex.3

T = 3:

$$U = \max_{W_2,W_3,W_4} \sum_{t=1}^{T=2} \beta^{t-1} u(W_t - W_{t+1})$$

$$= \max_{W_2,W_3,W_4} u(W_1 - W_2) + \beta\, u(W_2 - W_3) + \beta^2\, u(W_3 - W_4)$$

The optimal amount of cake to leave for W_4 is:

$$W_4 = min(W_4) = 0 \quad \forall W_3, W_2, W_1$$

The optimal amount of cake to leave for W_3 should satisfy:

$$\frac{\partial U}{\partial W_3} = 0$$

$$-\beta\, \frac{\partial u(C_2)}{\partial C_2} + \beta^2\, \frac{\partial (C_3)}{\partial C_3} = 0$$

$$\Longrightarrow \quad \frac{\partial u(C_2)}{\partial C_2} = \beta\, \frac{\partial (C_3)}{\partial C_3}$$

The optimal amount of cake to leave for W_2 should satisfy:

$$\frac{\partial U}{\partial W_2} = 0$$

$$-\frac{\partial u(C_1)}{\partial C_1} + \beta\, \frac{\partial (C_2)}{\partial C_2} = 0$$

$$\Longrightarrow \quad \frac{\partial u(C_1)}{\partial C_1} = \beta\, \frac{\partial (C_2)}{\partial C_2}$$

Let $W_1 = 1$, $\beta = 0.9$, and the period utility function be $\ln C_t$.
Then the utility function becomes:

$$U = \max_{W_2, W_3, W_4} \ln(C_1) + \ln(C_2) + \ln(C_3)$$

where $C_t = W_t - W_{t+1}$
The optimal amount for W_4 is $W_4 = 0$.
The optimal amount of W_3 should satisty:

$$\frac{\partial u(C_2)}{\partial C_2} = \beta \frac{\partial (C_3)}{\partial C_3}$$
$$\implies \frac{1}{C_2} = \beta \frac{1}{C_3}$$
$$\implies \frac{1}{W_2 - W_3} = \beta \frac{1}{W_3 - W_4}$$
$$\implies \frac{1}{W_2 - W_3} = 0.9 \frac{1}{W_3}$$
$$\implies W_3 = \frac{0.9}{1.9} W_2$$

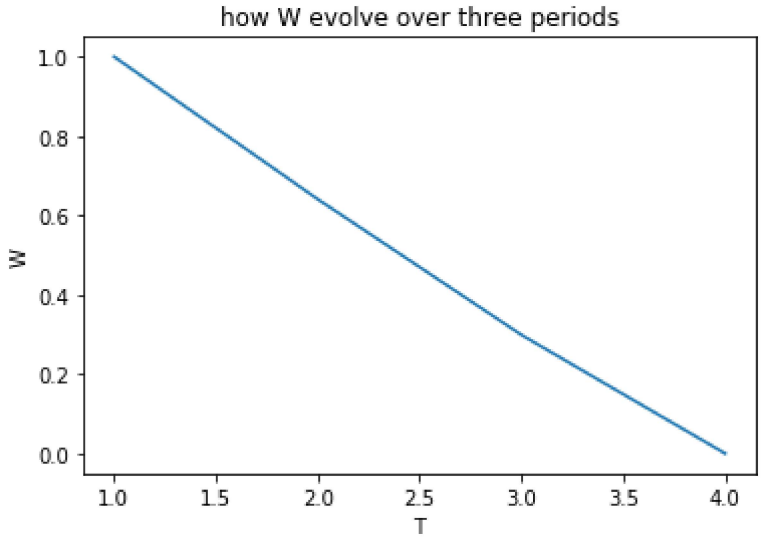
Similarly, the optimal amount of W_2 should satisty:

$$\frac{\partial u(C_1)}{\partial C_1} = \beta \frac{\partial (C_2)}{\partial C_2}$$
$$\implies \frac{1}{C_1} = \beta \frac{1}{C_2}$$
$$\implies \frac{1}{W_1 - W_2} = \beta \frac{1}{W_2 - W_3}$$
$$\implies \frac{1}{1 - W_2} = 0.9 \frac{1}{W_2 - W_3}$$
$$\implies W_2 = \frac{0.9 + W_3}{1.9}$$

Solving for W_2, W_3, we get: $W_2 = 0.630996, W_3 = 0.298893$.
Then $C_1 = W_1 - W_2 = 0.369004, C_2 = W_2 - W_3 = 0.332103, C_3 = W_3 - W_4 = 0.298893$.

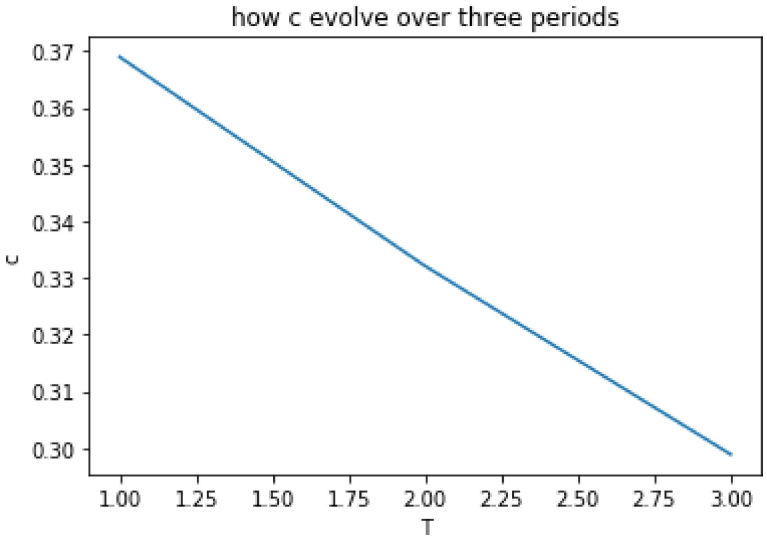
In [3]:

```
W = [1, 0.640996, 0.298893, 0]
c = [0.369004, 0.332103, 0.298893]
T = [1, 2, 3, 4]

plt.plot(T, W)
plt.title("how W evolve over three periods")
plt.xlabel("T")
plt.ylabel("W")
plt.show()
```



In [4]:

```
plt.clf()
plt.plot(T[0:-1], c)
plt.title("how c evolve over three periods")
plt.xlabel("T")
plt.ylabel("c")
plt.show()
```



Ex.4

The optimal choice in period $T-1$ for $W_T = \psi_{T-1}(W_{T-1})$ should satisfy:

$$-u(W_{T-1} - W_T) + \beta\, u(W_T - W_{T+1}) = 0$$
$$\implies \quad -u(W_{T-1} - \psi_{T-1}(W_{T-1})) + \beta\, u(\psi_{T-1}(W_{T-1})) = 0$$

The value function V_{T-1} is:

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

Ex.5

$u(c) = \ln(c)$

If $T < \infty$ represents the last period of an individual's life:

$$\psi_T(W_T) = W_{T+1} = 0$$

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

So $\psi_{T-1}(\bar{W}) \neq \psi_T(\bar{W})$.

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

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

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

Ex.6

$u(c) = \ln(c)$

The finite horizon Bellman equation for the value function at time $T-2$ is:

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

Since $W_T = \psi_{T-1}(W_{T-1})$, we have:

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

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

To get the optimal value of $W_{T-1} = \psi_{T-2}(W_{T-2})$, consider the first order condition of this equation:

$$-\frac{1}{W_{T-2} - W_{T-1}} + \beta \frac{1}{W_{T-1}} + \beta^2 \frac{1}{W_{T-1}} = 0$$

$$\implies \quad W_{T-1} = \frac{\beta + \beta^2}{1 + \beta + \beta^2} W_{T-2}$$

The analytical solution for V_{T-2} is:

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

Ex.7

$u(c) = \ln(c)$

The analytical solutions for $\psi_{T-s}(W_{T-s})$ and $V_{T-s}(W_{T-s})$ are:

$$\psi_{T-s}(W_{T-s}) = \frac{\sum_{i=1}^{s} \beta^i}{\sum_{j=0}^{s} \beta^j} W_{T-s}$$

$$V_{T-s}(W_{T-s}) = \sum_{i=0}^{s} \beta^i \ln(\frac{\beta^i}{\sum_{j=0}^{s} \beta^j} W_{T-s})$$

If $s \to \infty$:

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

$$\lim_{s\to\infty} V_{T-s}(W_{T-s}) = \lim_{s\to\infty} \sum_{i=0}^{s} \beta^i \ln(\frac{\beta^i}{\sum_{j=0}^{s} \beta^j} W_{T-s}) = \frac{1}{1-\beta} \ln((1-\beta) W_{T-s}) + \frac{\beta}{(1-\beta)^2} \ln(\beta)$$

Ex.8

The Bellman equation for a gerneral utility function $u(c)$ when the horizon is infinite:

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

Ex.9

In [5]:

```
W_max = 1
W_min = 0.01
N = 100
W = np.linspace(W_min, W_max, N).reshape(N, 1)  # column vector
W
```

Out[5]:

```
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.  ]])
```

Ex.10

In [6]:

```
# T is the last period, so W_T+1 should be 0.
Wprime = W.reshape(1, N)

beta = 0.9

# u(c) = ln(c)
def u(c):
    c_leq0 = c <= 10e-10
    util = np.zeros_like(c)
    util[~c_leq0] = np.log(c[~c_leq0])
    util[c_leq0] = -10e10
    return util
```

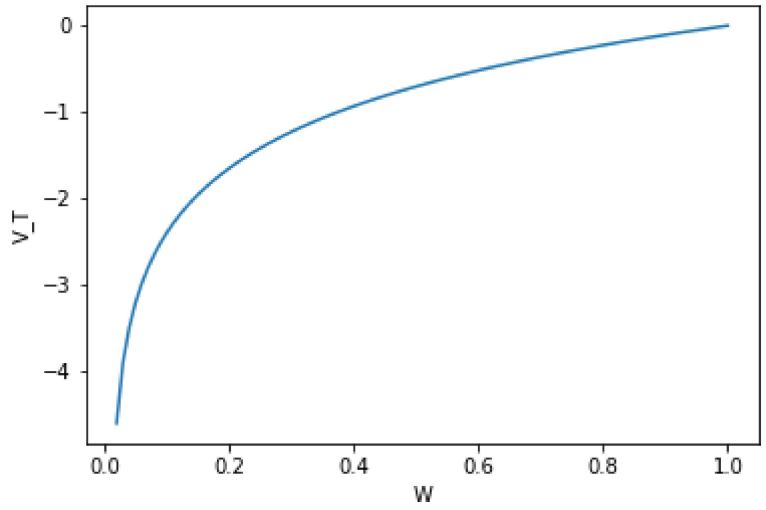
In [7]:

```
W_mat = np.tile(W, (1, N))
Wprime_mat = np.tile(Wprime, (N, 1))

c_mat = W_mat - Wprime_mat
util = u(c_mat)

V_Tp1 = np.zeros(N).reshape(N, 1) ###

V_T = (util + beta * V_Tp1).max(axis=1)

index = np.argmax(util + beta * V_Tp1, axis=1)  ###
Wprime_best = np.array(W)[index].reshape(1, N)

print("W' is: " + str(Wprime_best))
print("V_T is: " + str(V_T))
```

```
W' is: [[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]]
V_T is: [-1.00000000e+11 -4.60517019e+00 -3.91202301e+00 -3.50655790e+00
 -3.21887582e+00 -2.99573227e+00 -2.81341072e+00 -2.65926004e+00
 -2.52572864e+00 -2.40794561e+00 -2.30258509e+00 -2.20727491e+00
 -2.12026354e+00 -2.04022083e+00 -1.96611286e+00 -1.89711998e+00
 -1.83258146e+00 -1.77195684e+00 -1.71479843e+00 -1.66073121e+00
 -1.60943791e+00 -1.56064775e+00 -1.51412773e+00 -1.46967597e+00
 -1.42711636e+00 -1.38629436e+00 -1.34707365e+00 -1.30933332e+00
 -1.27296568e+00 -1.23787436e+00 -1.20397280e+00 -1.17118298e+00
 -1.13943428e+00 -1.10866262e+00 -1.07880966e+00 -1.04982212e+00
 -1.02165125e+00 -9.94252273e-01 -9.67584026e-01 -9.41608540e-01
 -9.16290732e-01 -8.91598119e-01 -8.67500568e-01 -8.43970070e-01
 -8.20980552e-01 -7.98507696e-01 -7.76528789e-01 -7.55022584e-01
 -7.33969175e-01 -7.13349888e-01 -6.93147181e-01 -6.73344553e-01
 -6.53926467e-01 -6.34878272e-01 -6.16186139e-01 -5.97837001e-01
 -5.79818495e-01 -5.62118918e-01 -5.44727175e-01 -5.27632742e-01
 -5.10825624e-01 -4.94296322e-01 -4.78035801e-01 -4.62035460e-01
 -4.46287103e-01 -4.30782916e-01 -4.15515444e-01 -4.00477567e-01
 -3.85662481e-01 -3.71063681e-01 -3.56674944e-01 -3.42490309e-01
 -3.28504067e-01 -3.14710745e-01 -3.01105093e-01 -2.87682072e-01
 -2.74436846e-01 -2.61364764e-01 -2.48461359e-01 -2.35722334e-01
 -2.23143551e-01 -2.10721031e-01 -1.98450939e-01 -1.86329578e-01
 -1.74353387e-01 -1.62518929e-01 -1.50822890e-01 -1.39262067e-01
 -1.27833372e-01 -1.16533816e-01 -1.05360516e-01 -9.43106795e-02
 -8.33816089e-02 -7.25706928e-02 -6.18754037e-02 -5.12932944e-02
 -4.08219945e-02 -3.04592075e-02 -2.02027073e-02 -1.00503359e-02]
```

In [8]:

```
plt.clf()
plt.plot(W[1:], V_T[1:])
plt.xlabel('W')
plt.ylabel('V_T')
plt.show()
```



Ex.11

In [14]:

```
def dist(x1, x2):
    return ((x1 - x2)**2).sum()


delta_T = dist(V_T[1:], V_Tp1[1:])

print("The distance between V_T and V_tm1 is: " + str(delta_T))
```

```
The distance between V_T and V_tm1 is: 17713.684955313078
```

Ex.12

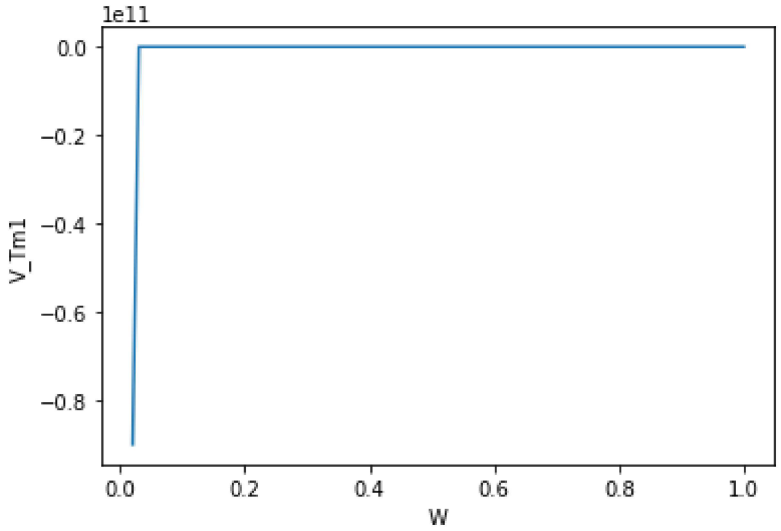
In [15]:

```
# c_mat = W_mat - Wprime_mat
util = u(c_mat)

# get V_T from Exercise 10
V_Tm1 = (util + beta * V_T).max(axis=1)

index = np.argmax(util + beta * V_T, axis=1)
Wprime_best = np.array(W)[index].reshape(1, N)

print("The policy function W_prime is " + str(Wprime_best))
print("The value function V_T-1 is " + str(V_Tm1[1:]))
```

```
The policy function W_prime is [[1.   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]]
The value function V_T-1 is [-9.00000000e+10 -8.74982335e+00 -8.05667617e+
00 -7.43284371e+00
 -7.02737860e+00 -6.66246000e+00 -6.37477793e+00 -6.11586407e+00
 -5.89272052e+00 -5.69189132e+00 -5.50956976e+00 -5.34548036e+00
 -5.19132968e+00 -5.05259407e+00 -4.91906268e+00 -4.79888442e+00
 -4.68110139e+00 -4.57509666e+00 -4.46973614e+00 -4.37442596e+00
 -4.27960150e+00 -4.19259012e+00 -4.10681096e+00 -4.02676825e+00
 -3.94845801e+00 -3.87435004e+00 -3.80231160e+00 -3.73331873e+00
 -3.66662156e+00 -3.60208303e+00 -3.53998945e+00 -3.47936483e+00
 -3.42128016e+00 -3.36412175e+00 -3.30955959e+00 -3.25549236e+00
 -3.20404979e+00 -3.15275650e+00 -3.10396633e+00 -3.05530583e+00
 -3.00878582e+00 -2.96262185e+00 -2.91817009e+00 -2.87425894e+00
 -2.83169933e+00 -2.78983132e+00 -2.74900932e+00 -2.70900273e+00
 -2.66978202e+00 -2.63147837e+00 -2.59373804e+00 -2.55699824e+00
 -2.52063060e+00 -2.48533196e+00 -2.45024064e+00 -2.41627434e+00
 -2.38237279e+00 -2.34958297e+00 -2.31685209e+00 -2.28510339e+00
 -2.25352120e+00 -2.22274954e+00 -2.19223815e+00 -2.16238519e+00
 -2.13287434e+00 -2.10388681e+00 -2.07531298e+00 -2.04714210e+00
 -2.01944761e+00 -1.99204864e+00 -1.96518097e+00 -1.93851272e+00
 -1.91242394e+00 -1.88644845e+00 -1.86109466e+00 -1.83577685e+00
 -1.81108424e+00 -1.78642517e+00 -1.76232761e+00 -1.73832619e+00
 -1.71479569e+00 -1.69141776e+00 -1.66842824e+00 -1.64564221e+00
 -1.62316935e+00 -1.60094600e+00 -1.57896710e+00 -1.55727930e+00
 -1.53577310e+00 -1.51459565e+00 -1.49354224e+00 -1.47285167e+00
 -1.45223238e+00 -1.43200681e+00 -1.41180411e+00 -1.39200148e+00
 -1.37222046e+00 -1.35280238e+00 -1.33344679e+00]
```

In [17]:

```
plt.clf()
plt.plot(W[1:], V_Tm1[1:])
plt.xlabel('W')
plt.ylabel('V_Tm1')
plt.show()
```



In [18]:

```
delta_Tm1 = dist(V_Tm1[1:], V_T[1:])

print("The distance measure for delta_T-1 is " + str(delta_Tm1))
```

```
The distance measure for delta_T-1 is 8.1e+21
```

In [19]:

```
delta_Tm1 - delta_T
```

Out[19]:

```
8.1e+21
```

delta_T-1 is bigger than delta_T from Exercise 11. The reason for the incresasement may because we assume T be the last period and V(W_T+1) = 0.

Ex.13

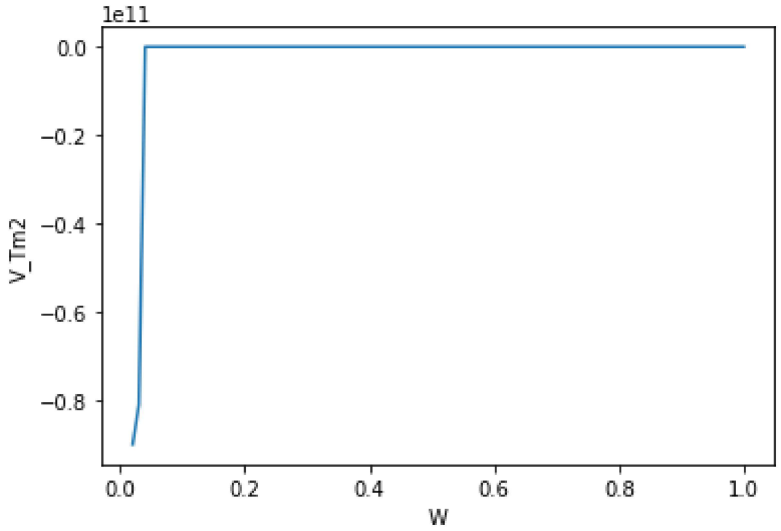
In [20]:

```
util = u(c_mat)

V_Tm2 = (util + beta * V_Tm1).max(axis=1)

index = np.argmax(util + beta * V_Tm1, axis=1)
Wprime_best = np.array(W)[index].reshape(1, N)

print("The policy function W_prime is " + str(Wprime_best))
print("The value function V_T-1 is " + str(V_Tm2[1:]))
```

```
The policy function W_prime is [[1.   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]]
The value function V_T-1 is [-9.00000000e+10 -8.10000000e+10 -1.24800112e+
01 -1.17868640e+01
 -1.11630316e+01 -1.06015823e+01 -1.01961172e+01 -9.83119864e+00
 -9.50277190e+00 -9.21508983e+00 -8.95617596e+00 -8.72315349e+00
 -8.50000993e+00 -8.29918074e+00 -8.11685918e+00 -7.93611290e+00
 -7.77202350e+00 -7.61787282e+00 -7.47019236e+00 -7.33145675e+00
 -7.19792536e+00 -7.07306331e+00 -6.95288505e+00 -6.83510202e+00
 -6.72694159e+00 -6.62093686e+00 -6.51557634e+00 -6.42017208e+00
 -6.32486190e+00 -6.23003744e+00 -6.14302606e+00 -6.05724690e+00
 -5.97190488e+00 -5.89186218e+00 -5.81355194e+00 -5.73635069e+00
 -5.66224272e+00 -5.59020428e+00 -5.51972507e+00 -5.45073219e+00
 -5.38403502e+00 -5.31920043e+00 -5.25466191e+00 -5.19256832e+00
 -5.13194370e+00 -5.07191624e+00 -5.01383157e+00 -4.95667316e+00
 -4.90078893e+00 -4.84622677e+00 -4.79215955e+00 -4.73988335e+00
 -4.68844078e+00 -4.63714748e+00 -4.58804154e+00 -4.53925138e+00
 -4.49059088e+00 -4.44407086e+00 -4.39777255e+00 -4.35160858e+00
 -4.30715682e+00 -4.26324567e+00 -4.21945122e+00 -4.17689161e+00
 -4.13502359e+00 -4.09347602e+00 -4.05265403e+00 -4.01264744e+00
 -3.97312741e+00 -3.93390670e+00 -3.89560304e+00 -3.85786272e+00
 -3.82018150e+00 -3.78344171e+00 -3.74707406e+00 -3.71106814e+00
 -3.67576949e+00 -3.64067817e+00 -3.60620489e+00 -3.57223859e+00
 -3.53833704e+00 -3.50527122e+00 -3.47248140e+00 -3.43975052e+00
 -3.40798174e+00 -3.37623305e+00 -3.34465086e+00 -3.31387920e+00
 -3.28330953e+00 -3.25279814e+00 -3.22294517e+00 -3.19343433e+00
 -3.16397654e+00 -3.13498901e+00 -3.10641518e+00 -3.07799121e+00
 -3.04982033e+00 -3.02212584e+00 -2.99466558e+00]
```

In [21]:

```
plt.clf()
plt.plot(W[1:], V_Tm2[1:])
plt.xlabel('W')
plt.ylabel('V_Tm2')
plt.show()
```



In [22]:

```
delta_Tm2 = dist(V_Tm2, V_Tm1)

print("The distance measure for delta_T-2 is " + str(delta_Tm2))
```

```
The distance measure for delta_T-2 is 6.561e+21
```

delta_T-2 is smaller than delta_T-1, but still much larger than delta_T. The distance starts to decrease.

Ex.14

In [23]:

```
W_max = 1
W_min = 0.01
N = 100
W = np.linspace(W_min, W_max, N).reshape(N, 1)  # column vector
Wprime = W.reshape(1, N)
beta = 0.9

def u(c):
    c_leq0 = c <= 1e-10
    util = np.zeros_like(c)
    util[~c_leq0] = np.log(c[~c_leq0])
    util[c_leq0] = -1e10
    return util
```

In [24]:

```
W_mat = np.tile(W, (1, N))
Wprime_mat = np.tile(Wprime, (N, 1))

c_mat = W_mat - Wprime_mat
util = u(c_mat)
```

In [25]:

```
distance = 100 # random initial value of distance
s = 0
V_prime = u(Wprime).reshape((N,))

while distance >= 10e-9:
    s += 1
    V_Tms = (util + beta * V_prime).max(axis=1)
    index = np.argmax(util + beta * V_Tms, axis=1)
    Wprime = np.array(W)[index]
    distance = dist(V_Tms[1:], V_prime[1:])
    print("s = " + str(s) + ", distance = " + str(distance))
    V_prime = V_Tms

print("The convergence takes " + str(s) + " iterations.")
print("V_Tms = V_init? " + str(np.array_equal(V_prime, V_Tms)))
```

```
s = 1, distance = 518.0557837063243
s = 2, distance = 8.099999992539624e+19
s = 3, distance = 6.560999993957095e+19
s = 4, distance = 5.314409995105247e+19
s = 5, distance = 4.304672096035251e+19
s = 6, distance = 3.486784397788553e+19
s = 7, distance = 2.824295362208728e+19
s = 8, distance = 2.2876792433890697e+19
s = 9, distance = 1.853020187145146e+19
s = 10, distance = 1.5009463515875686e+19
s = 11, distance = 1.2157665447859306e+19
s = 12, distance = 9.847709012766036e+18
s = 13, distance = 7.976644300340488e+18
s = 14, distance = 6.461081883275798e+18
s = 15, distance = 5.233476325453397e+18
s = 16, distance = 4.2391158236172524e+18
s = 17, distance = 3.433683817129975e+18
s = 18, distance = 2.78128389187528e+18
s = 19, distance = 2.252839952418977e+18
s = 20, distance = 1.8248003614593718e+18
s = 21, distance = 1.478088292782091e+18
s = 22, distance = 1.197251517153494e+18
s = 23, distance = 9.697737288943301e+17
s = 24, distance = 7.855167204044073e+17
s = 25, distance = 6.362685435275699e+17
s = 26, distance = 5.153775202573316 5e+17
s = 27, distance = 4.1745579140843846e+17
s = 28, distance = 3.381391910408352e+17
s = 29, distance = 2.7389274474307645e+17
s = 30, distance = 2.2185312324189197e+17
s = 31, distance = 1.797010298259325e+17
s = 32, distance = 1.455578341590053e+17
s = 33, distance = 1.1790184566879434e+17
s = 34, distance = 9.550049499172341e+16
s = 35, distance = 7.735540094329598e+16
s = 36, distance = 6.2657874764069736e+16
s = 37, distance = 5.0752878558896504e+16
s = 38, distance = 4.110983163270617e+16
s = 39, distance = 3.3298963622492012e+16
s = 40, distance = 2.697216053421853e+16
s = 41, distance = 2.1847450032717016e+16
s = 42, distance = 1.7696434526500784e+16
s = 43, distance = 1.433411196646564e+16
s = 44, distance = 1.1610630692837172e+16
s = 45, distance = 9404610861198114.0
s = 46, distance = 7617734797570475.0
s = 47, distance = 6170365186032085.0
s = 48, distance = 4997995800685991.0
s = 49, distance = 4048376598555654.0
s = 50, distance = 3279185044830079.5
s = 51, distance = 2656139886312364.5
s = 52, distance = 2151473307913016.2
s = 53, distance = 1742693379409543.5
s = 54, distance = 1411581637321730.2
s = 55, distance = 1143381126230602.2
s = 56, distance = 926138712246788.8
s = 57, distance = 750172356919899.9
s = 58, distance = 607639609105119.4
s = 59, distance = 492188083375147.7
s = 60, distance = 398672347533870.44
s = 61, distance = 322924601502435.75
```

```
s = 62, distance = 261568927216973.78
s = 63, distance = 211870831045749.53
s = 64, distance = 171615373147057.78
s = 65, distance = 139008452249117.6
s = 66, distance = 112596846321785.9
s = 67, distance = 91203445520647.31
s = 68, distance = 73874790871725.05
s = 69, distance = 59838580606097.93
s = 70, distance = 48469250290939.984
s = 71, distance = 39260092735662.055
s = 72, distance = 31800675115886.914
s = 73, distance = 25758546843869.04
s = 74, distance = 20864422943534.555
s = 75, distance = 16900182584263.62
s = 76, distance = 13689147893254.154
s = 77, distance = 11088209793536.479
s = 78, distance = 8981449932765.148
s = 79, distance = 7274974445540.364
s = 80, distance = 5892729300888.277
s = 81, distance = 4773110733720.075
s = 82, distance = 3866219694313.8184
s = 83, distance = 3131637952394.7417
s = 84, distance = 2536626741440.27
s = 85, distance = 2054667660567.1375
s = 86, distance = 1664280805059.887
s = 87, distance = 1348067452098.9915
s = 88, distance = 1091934636200.6525
s = 89, distance = 884467055322.9829
s = 90, distance = 716418314812.0355
s = 91, distance = 580298834998.158
s = 92, distance = 470042056348.8742
s = 93, distance = 380734065642.9417
s = 94, distance = 308394593171.0837
s = 95, distance = 249799620468.86304
s = 96, distance = 202337692579.99512
s = 97, distance = 163893530989.95004
s = 98, distance = 132753760101.937
s = 99, distance = 107530545682.55177
s = 100, distance = 87099742002.73306
s = 101, distance = 0.955175757759564
s = 102, distance = 57146140728.19228
s = 103, distance = 46288373989.58955
s = 104, distance = 37493582931.7522
s = 105, distance = 30369802174.78575
s = 106, distance = 24599539761.72604
s = 107, distance = 19925627207.18655
s = 108, distance = 16139758037.651493
s = 109, distance = 13073204010.410473
s = 110, distance = 10589295248.373602
s = 111, distance = 8577329151.235612
s = 112, distance = 6947636612.4690485
s = 113, distance = 5627585656.09993
s = 114, distance = 4558344381.428066
s = 115, distance = 3692258948.921963
s = 116, distance = 2990729748.647652
s = 117, distance = 2422491096.3975573
s = 118, distance = 1962217788.0799098
s = 119, distance = 1589396408.3466275
s = 120, distance = 1287411090.7624793
s = 121, distance = 1042802983.5083693
s = 122, distance = 844670416.6403934
```

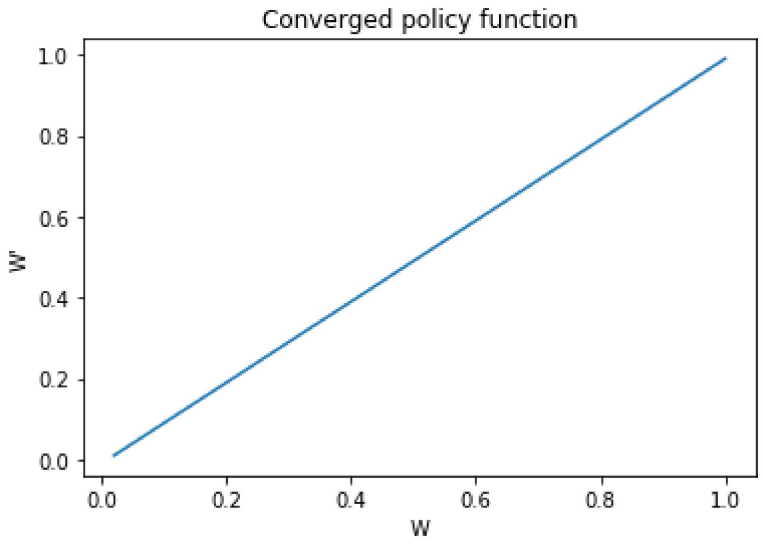
```
s = 123, distance = 684183037.4787184
s = 124, distance = 554188260.3572006
s = 125, distance = 448892490.8918582
s = 126, distance = 363602917.621496
s = 127, distance = 294518363.2721843
s = 128, distance = 238559874.24862796
s = 129, distance = 193233498.14205158
s = 130, distance = 156519133.49580738
s = 131, distance = 126780498.13066447
s = 132, distance = 102692203.48583819
s = 133, distance = 83180684.82407254
s = 134, distance = 67376354.70681381
s = 135, distance = 54574847.3117266
s = 136, distance = 44205626.32249854
s = 137, distance = 35806557.32125949
s = 138, distance = 29003311.430220183
s = 139, distance = 23492682.258536134
s = 140, distance = 19029072.62938827
s = 141, distance = 15413548.8298513
s = 142, distance = 12484974.552158492
s = 143, distance = 10112829.387210468
s = 144, distance = 8191391.803631952
s = 145, distance = 6635027.360980261
s = 146, distance = 5374372.162352557
s = 147, distance = 4353241.451542879
s = 148, distance = 3526125.575732944
s = 149, distance = 2856161.716358795
s = 150, distance = 2313490.990237024
s = 151, distance = 1873927.7020940299
s = 152, distance = 1517881.438712687
s = 153, distance = 1229483.9653688425
s = 154, distance = 995882.0119547107
s = 155, distance = 806664.4296793005
s = 156, distance = 653398.1880414379
s = 157, distance = 529252.5323130225
s = 158, distance = 428694.55117598735
s = 159, distance = 347242.58645254985
s = 160, distance = 281266.495025775
s = 161, distance = 227825.8609712335
s = 162, distance = 184538.9473873393
s = 163, distance = 149476.547384609
s = 164, distance = 121076.00338049627
s = 165, distance = 98071.5627380853
s = 166, distance = 79437.96581795409
s = 167, distance = 64344.75231235383
s = 168, distance = 52119.249372921535
s = 169, distance = 42216.59199183684
s = 170, distance = 34195.43951321562
s = 171, distance = 27698.306005673643
s = 172, distance = 22435.627864763042
s = 173, distance = 18172.858570357617
s = 174, distance = 14720.01544205748
s = 175, distance = 11923.21250812759
s = 176, distance = 9657.802131555873
s = 177, distance = 7822.819726617934
s = 178, distance = 6336.483978597583
s = 179, distance = 5132.552022677383
s = 180, distance = 4157.367138356661
s = 181, distance = 3367.467382063491
s = 182, distance = 2727.6485794811365
s = 183, distance = 2209.3953493819017
```

```
s = 184, distance = 1789.610232993419
s = 185, distance = 1449.584288724703
s = 186, distance = 1174.1632738574035
s = 187, distance = 951.0722518273398
s = 188, distance = 770.3685239827472
s = 189, distance = 623.9985044225217
s = 190, distance = 505.4387885864289
s = 191, distance = 409.4054187550196
s = 192, distance = 331.61838919278193
s = 193, distance = 268.61089524538505
s = 194, distance = 217.57482514801438
s = 195, distance = 176.2356083696033
s = 196, distance = 142.75084277987912
s = 197, distance = 115.62818265068273
s = 198, distance = 93.65882794721058
s = 199, distance = 75.86365063743605
s = 200, distance = 61.44955701563342
s = 201, distance = 7.235294831797398e-10
The convergence takes 201 iterations.
V_Tms = V_init? True
```

Ex.15

In [26]:

```
plt.clf()
plt.plot(W[1:], Wprime[1:])
plt.title("Converged policy function")
plt.xlabel("W")
plt.ylabel("W'")
plt.show()
```



Ex.16

In [27]:

```
sigma = 0.5
mu = 4 * sigma

epsilon_max = mu + 3 * sigma
epsilon_min = mu - 3 * sigma

M = 7
epsilon = np.linspace(epsilon_min, epsilon_max, M)
epsilon
```

Out[27]:

```
array([0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5])
```

In [28]:

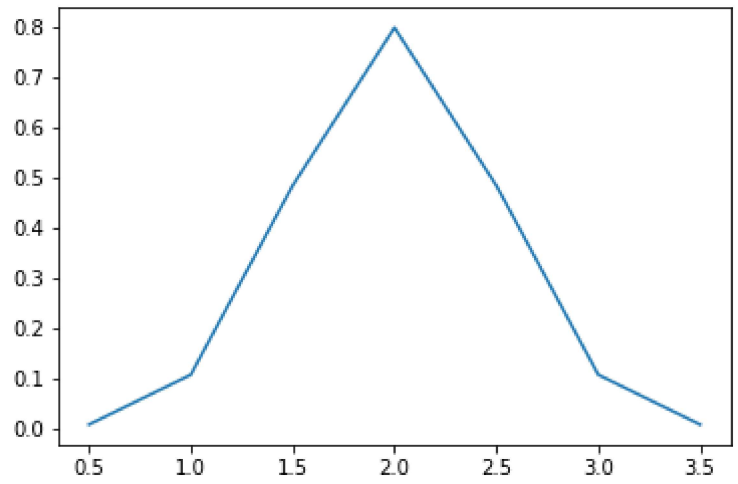
```
pdf = sta.norm.pdf(epsilon, mu, sigma)
pdf
```

Out[28]:

```
array([0.0088637 , 0.10798193, 0.48394145, 0.79788456, 0.48394145,
       0.10798193, 0.0088637 ])
```

In [29]:

```
plt.clf()
plt.plot(epsilon, pdf)
plt.show()
```



Ex.17

In [56]:

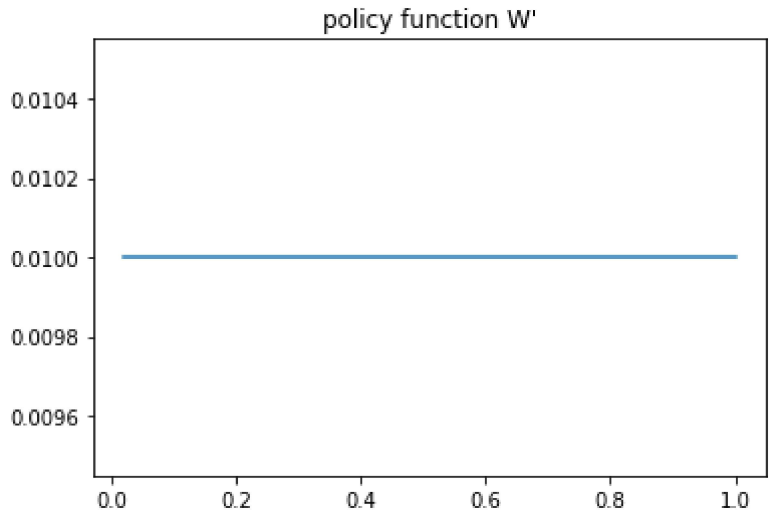
```
# from Exericise 9
W_max = 1
W_min = 0.01
N = 100
W = np.linspace(W_min, W_max, N).reshape((N, 1))
V_Tp1 = np.zeros((N, M))
beta = 0.9
Wprime = W.reshape((1, N))
W_mat = np.tile(W, (1, N))
Wprime_mat = np.tile(Wprime, (N, 1))

c_mat = W_mat - Wprime_mat
util = u(c_mat)
u_cube = np.array([util * e for e in epsilon])
```

In [59]:

```
EV = V_Tp1 @ pdf.reshape((M, 1))
EV_mat = np.tile(EV.reshape((1, N)), (N, 1))
EV_mat[c_leq0] = -10e10
EV_cube = np.array([EV_mat for m in range(M)])

V = u_cube + beta * EV_cube
V_T = np.zeros((N, M))
W_new = np.zeros((N, M))
for i in range(N):
    V_tmp = V[:, i, :]
    V_T[i] = V_tmp.max(axis=1)
    index = np.argmax(V_tmp, axis=1)
    W_new[i] = W[index].reshape(M)
```

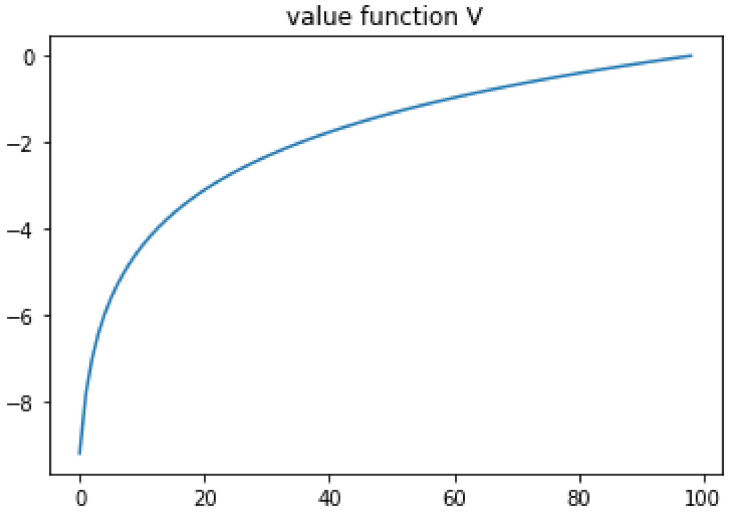
In [60]:

```
plt.clf()
plt.plot(W[1:], np.average(W_new[1:], axis=1))
plt.title("policy function W'")
plt.show()
```



In [61]:

```
plt.clf()
plt.plot(np.average(V_T[1:], axis=1))
plt.title("value function V")
plt.show()
```



Ex.18

In [62]:

```
dist_T = ((V_T[1:] - V_Tp1[1:])**2).sum()
dist_T
```

Out[62]:

```
6262.4138730904815
```

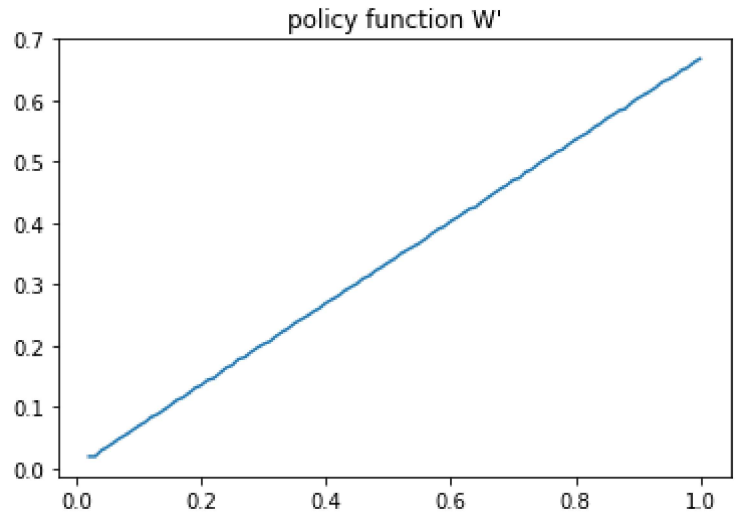
Ex.19

In [63]:

```
EV = V_T @ pdf.reshape((M, 1))
EV_mat = np.tile(EV.reshape((1, N)), (N, 1))
EV_mat[c_leq0] = -10e10
EV_cube = np.array([EV_mat for m in range(M)])

V = u_cube + beta * EV_cube
V_Tm1 = np.zeros((N, M))
W_new = np.zeros((N, M))
for i in range(N):
    V_tmp = V[:, i, :]
    V_Tm1[i] = V_tmp.max(axis=1)
    index = np.argmax(V_tmp, axis=1)
    W_new[i, :] = W[index].reshape(M)
```

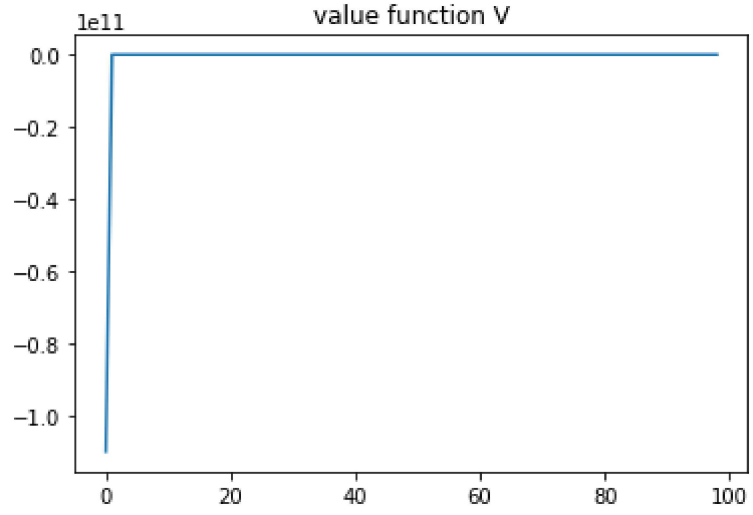
In [64]:

```
plt.clf()
plt.plot(W[1:], np.average(W_new[1:], axis=1))
plt.title("policy function W'")
plt.show()
```



In [65]:

```
plt.clf()
plt.plot(np.average(V_Tm1[1:], axis=1))
plt.title("value function V")
plt.show()
```



In [66]:

```
dist_Tm1 = ((V_Tm1[1:] - V_T[1:])**2).sum()
dist_Tm1
```

Out[66]:

```
8.539999998517135e+22
```

delta_T is 6262.4138730904815. The distance measure for delta_T-1 is larger than delta_T.

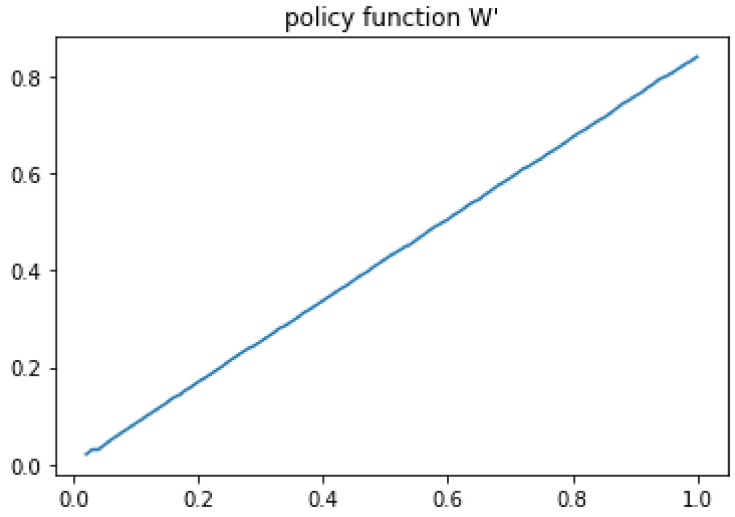
Ex.20

In [67]:

```
EV = V_Tm1 @ pdf.reshape((M, 1))
EV_mat = np.tile(EV.reshape((1, N)), (N, 1))
EV_mat[c_leq0] = -10e10
EV_cube = np.array([EV_mat for m in range(M)])

V = u_cube + beta * EV_cube
V_Tm2 = np.zeros((N, M))
W_new = np.zeros((N, M))
for i in range(N):
    V_tmp = V[:, i, :]
    V_Tm2[i] = V_tmp.max(axis=1)
    index = np.argmax(V_tmp, axis=1)
    W_new[i, :] = W[index].reshape(M)
```

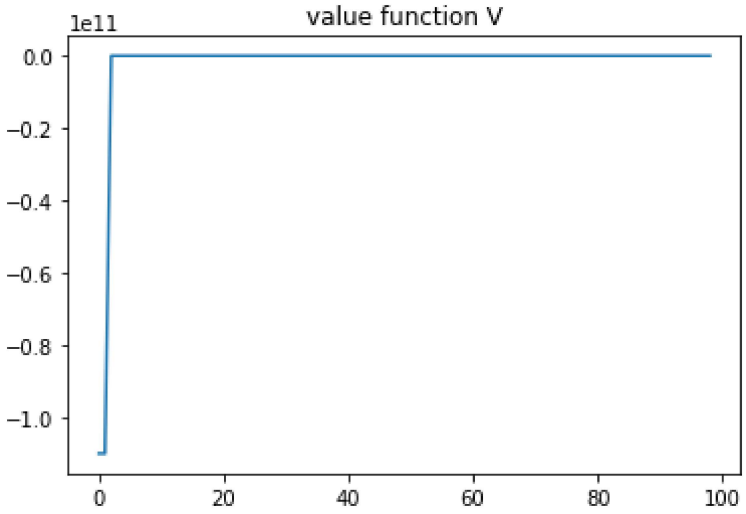
In [68]:

```
plt.clf()
plt.plot(W[1:], np.average(W_new[1:], axis=1))
plt.title("policy function W'")
plt.show()
```



In [69]:

```
plt.clf()
plt.plot(np.average(V_Tm2[1:], axis=1))
plt.title("value function V")
plt.show()
```



In [70]:

```
dist_Tm2 = ((V_Tm2[1:] - V_Tm1[1:])**2).sum()
dist_Tm2
```

Out[70]:

```
8.53999999596472e+22
```

delta_T is 6262.4138730904815, and delta_T-1 is 8.539999998517135e+22. delta_T-2 is larger than delta_T, but smaller than delta_T-1.

Ex.21

In [71]:

```
distance = 100 # random initial value of distance
s = 0
V_prime = np.zeros((N, M))

while distance >= 10e-9:
    s += 1
    EV = V_prime @ pdf.reshape((M, 1))
    EV_mat = np.tile(EV.reshape((1, N)), (N, 1))
    EV_mat[c_leq0] = -10e10
    EV_cube = np.array([EV_mat for m in range(M)])
    V = u_cube + beta * EV_cube
    V_Tms = np.zeros((N, M))
    W_new = np.zeros((N, M))
    for i in range(N):
        V_tmp = V[:, i, :]
        V_Tms[i] = V_tmp.max(axis=1)
        index = np.argmax(V_tmp, axis=1)
        W_new[i, :] = W[index].reshape(M)
    distance = ((V_Tms[1:] - V_prime[1:])**2).sum()
    print("s = " + str(s) + ", distance = " + str(distance))
    V_prime = V_Tms

print("The convergence takes " + str(s) + " iterations.")
print("V_Tms = V_init? " + str(np.array_equal(V_prime, V_Tms)))
```

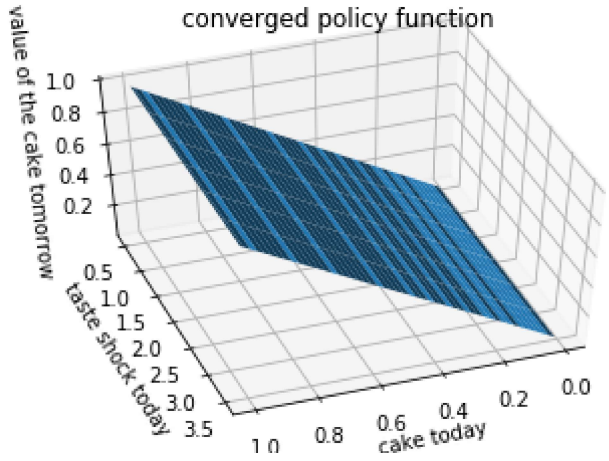
```
s = 1, distance = 6262.4138730904815
s = 2, distance = 8.539999998517135e+22
s = 3, distance = 8.53999999596472e+22
s = 4, distance = 8.539999991371615e+22
s = 5, distance = 8.539999983106266e+22
s = 6, distance = 8.539999968232661e+22
s = 7, distance = 8.539999941467422e+22
s = 8, distance = 8.539999893303033e+22
s = 9, distance = 8.539999806630596e+22
s = 10, distance = 8.539999650662464e+22
s = 11, distance = 8.539999369995857e+22
s = 12, distance = 8.539998864932911e+22
s = 13, distance = 8.539997956066318e+22
s = 14, distance = 8.539996320551384e+22
s = 15, distance = 8.539993377428237e+22
s = 16, distance = 8.539988081262861e+22
s = 17, distance = 8.53997855081789e+22
s = 18, distance = 8.539961400893885e+22
s = 19, distance = 8.539930540139983e+22
s = 20, distance = 8.539875008248734e+22
s = 21, distance = 8.539775085732864e+22
s = 22, distance = 8.539595299127749e+22
s = 23, distance = 8.53927185215248e+22
s = 24, distance = 8.53869006714661e+22
s = 25, distance = 8.537643981361449e+22
s = 26, distance = 8.535764254044734e+22
s = 27, distance = 8.532390411355947e+22
s = 28, distance = 8.526347313773291e+22
s = 29, distance = 8.515563391205091e+22
s = 30, distance = 8.49644957870112e+22
s = 31, distance = 8.462993846443604e+22
s = 32, distance = 8.405814355111598e+22
s = 33, distance = 8.312651374434622e+22
s = 34, distance = 8.176315209924524e+22
s = 35, distance = 8.031702427188001e+22
s = 36, distance = 8.095443563748337e+22
s = 37, distance = 9.252013221451266e+22
s = 38, distance = 1.468328437062851e+23
s = 39, distance = 3.5117252322364475e+23
s = 40, distance = 6.606987705868875e+23
s = 41, distance = 1.4639562968581868e+23
s = 42, distance = 0.0
The convergence takes 42 iterations.
V_Tms = V_init? True
```

Ex.22

In [81]:

```
plt.clf()
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y = np.meshgrid(W, epsilon)
ax.plot_surface(X.T, Y.T, W_new)
ax.set_xlabel("cake today")
ax.set_ylabel("taste shock today")
ax.set_zlabel("value of the cake tomorrow")
ax.set_title("converged policy function")
ax.view_init(elev=50, azim=70)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]: