PS3_psy

January 30, 2019

1 PS3

1.1 Siyuan Peng

1.1.1 5.1

The condition that characterizes the optimal amount of cake to eat in period 1 is:

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

1.1.2 5.2

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 leave for the next period W_2 in period 1 is:

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

1.1.3 5.3

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

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

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

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

The condition that characterizes the optimal amount of cake to leave for the next period W_4 in period 3 is:

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

From the third condition, we could clearly see that $W_4 = 0$. Then, we shall calculate the derivatives of the first condition respect to W_2 and W_3, which are:

$$-u'(W_1 - W_2) + \beta u'(W_2 - W_3) = 0$$

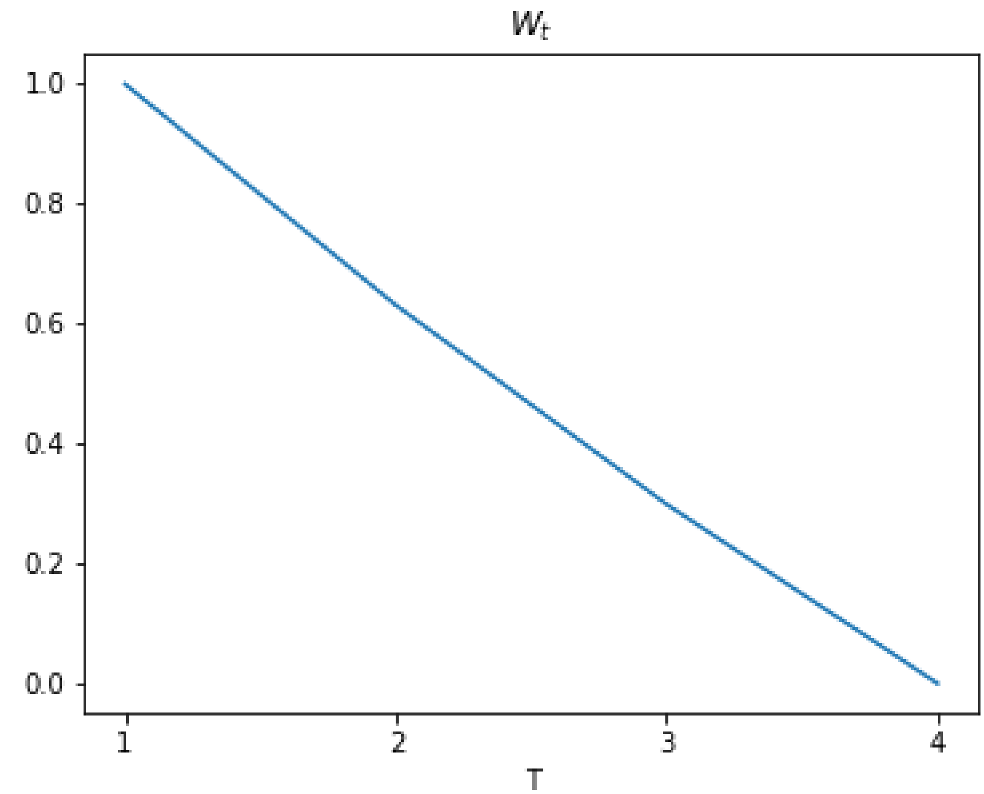
$$-\beta u'(W_2 - W_3) + \beta^2 u'(W_3 - W_4) = 0$$

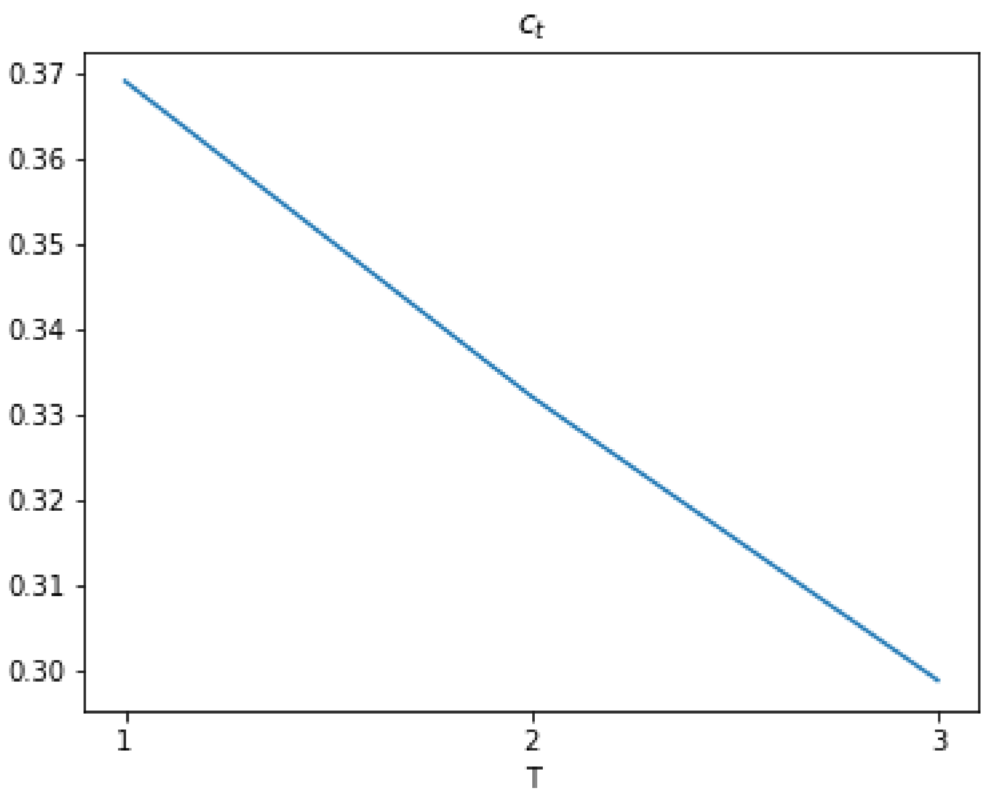
Considering that we already know that $u(x) = ln(x)$, $\beta = 0.9$, $W_1 = 1$ and $W_4 = 0$, we can easily solve them and get $W_2 = 0.631$, $W_3 = 0.299$. Therefore, $c_1 = W_1 - W_2 = 0.369$, $c_2 = W_2 - W_3 = 0.332$, $c_3 = W_3 - W_4 = 0.299$. The evolve of $\{c_t\}_{t=1}^3$ and $\{W_t\}_{t=1}^4$ will be shown as follows:

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: W = [1, 1-1/(1+0.9+0.81), 1-1.9/(1+0.9+0.81), 0]
        c = [1/(1+0.9+0.81),0.9/(1+0.9+0.81),0.81/(1+0.9+0.81)]
        T = [1,2,3,4]
        fig, ax=plt.subplots(2,1,figsize=(6,10))
        ax[0].plot(T, W)
        ax[0].set_title("$W_t$")
        ax[0].set_xlabel("T")
        ax[0].set_xticks([1,2,3,4])
        ax[1].plot(T[0:3], c)
        ax[1].set_title("$c_t$")
        ax[1].set_xlabel("T")
        ax[1].set_xticks([1,2,3])
```

```
Out[2]: [<matplotlib.axis.XTick at 0x1c6991a4198>,
         <matplotlib.axis.XTick at 0x1c6991a0a90>,
         <matplotlib.axis.XTick at 0x1c6991a07b8>]
```


W_t
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
T


c_t
0.37
0.36
0.35
0.34
0.33
0.32
0.31
0.30
1
2
3
T

1.1.4 5.4

The condition that characterizes the optimal choice (the policy function) in period T-1 is:

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

The value function could be shown as:

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

1.1.5 5.5

$V_{T-1}(\bar{W})$ could be represented as:

$$V_{T-1}(\bar{W}) = u(\bar{W} - \psi_{T-1}(\bar{W})) + \beta u(\psi_{T-1}(\bar{W}))$$

The derivative is:

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

Considering that $u(x) = ln(x)$, we could solve this and get:

$$\psi_{T-1}(\bar{W}) = \frac{\beta}{1+\beta}\bar{W}$$

Thus:

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

As for $V_T(\bar{W})$, considering that it only has 1 period, it could be represented as:

$$V_T(\bar{W}) = ln(\bar{W})$$

$$\psi_T(\bar{W}) = 0$$

It is quite clear to see that

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

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

1.1.6 5.6

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

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

$$W_{T-1} = \psi_{T-2}(W_{T-2})$$

The condition that characterizes the optimal choice in period T-2 is:

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

The analytical solution for $\psi_{T-2}(W_{T-2})$ and $V_{T-2}(W_{T-2})$ is:

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

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

1.1.7 5.7

According to the result of 5.5 and 5.6, it is quite clear to get the analytical solution for $\psi_{T-s}(W_{T-s})$ and $V_{T-s}(W_{T-s})$ by induction:

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

$$V_{T-s}(W_{T-s}) = \sum_{i=0}^{s} \beta^i ln\left(\frac{\beta^i W_{T-s}}{\sum\limits_{i=0}^{s} \beta^i}\right)$$

When s is approching infinite:

$$\psi(W_{T-s}) = \beta W_{T-s}$$

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

1.1.8 5.8

When the horizon is infinite, the Bellman equation could be represented as:

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

1.1.9 5.9

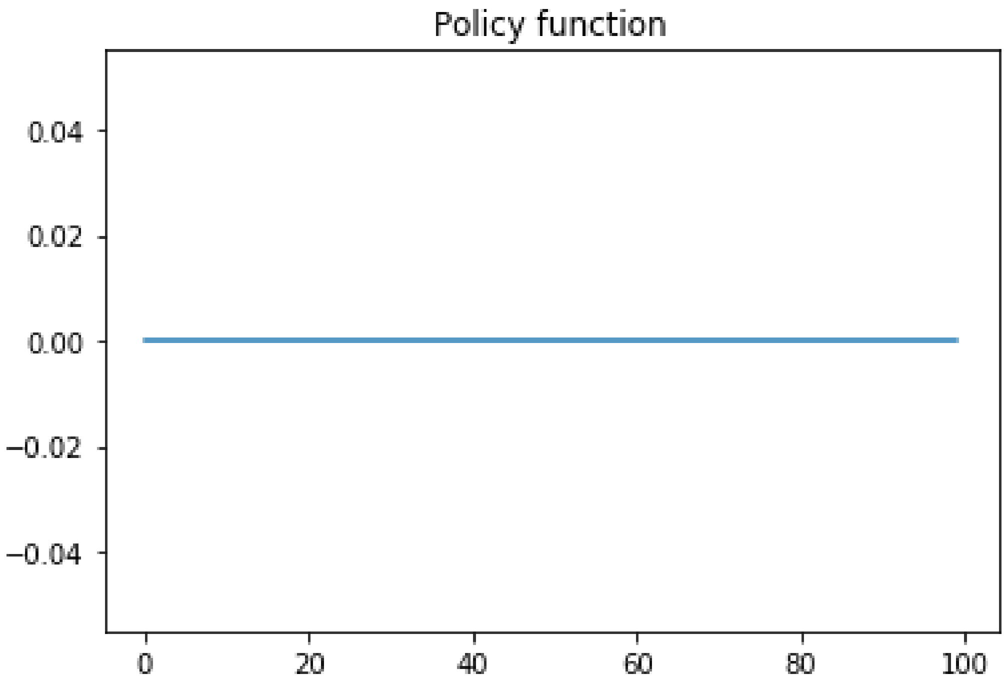
```
In [3]: import numpy as np
        import scipy.optimize as opt
```

```
In [4]: # From the question, we know that:
        W = np.linspace(0.01, 1, 100)
```

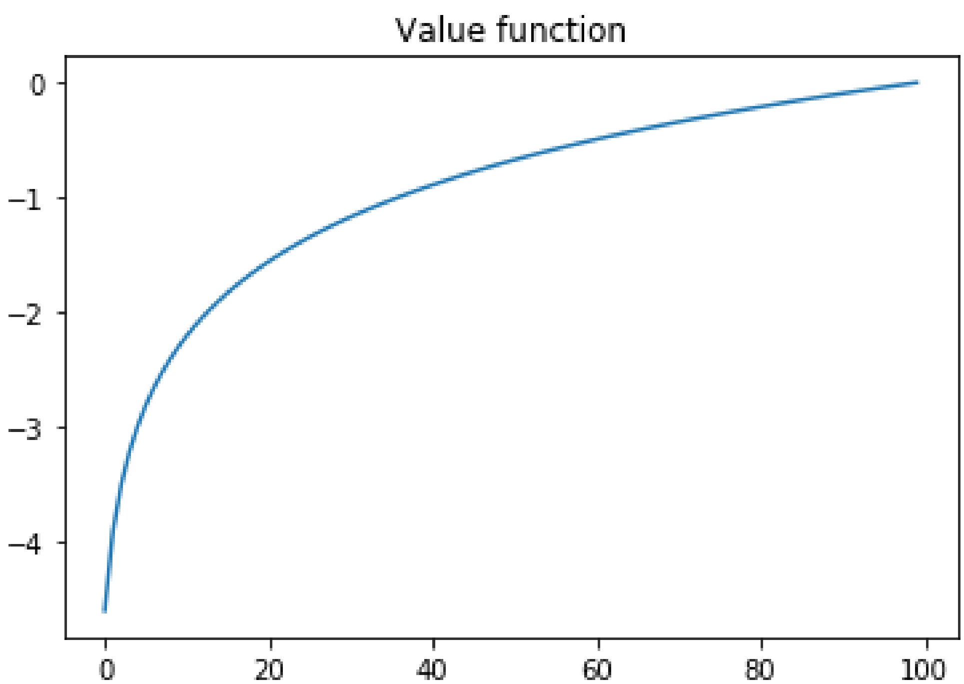
1.1.10 5.10

```
In [5]: c_mat = W.reshape(-1,1)-W
        c_mat[c_mat<=0] = 1e-10
        u_mat = np.log(c_mat)
        N = 100
        beta = 0.9
```

```
In [6]: psi_T = np.zeros(100)
        Value_T = np.zeros(100)
        for i in range(100):
            w = W[i]
            value_func = lambda x: -np.log(w-x)
            psi_T[i] = max(float(opt.fmin(value_func, 0, disp = 0)),0)
            Value_T[i] = np.log(w-psi_T[i])
        plt.plot(psi_T)
        plt.title("Policy function")
        plt.show()
```




```
In [7]: plt.plot(Value_T)
        plt.title("Value function")
        plt.show()
```



Considering that it is the last period, I will definitely spend all of my rest cake, thus, the policy function must be 0 for sure. As for the value function, since I come to this last period with different size of cake, the value of it should be different.

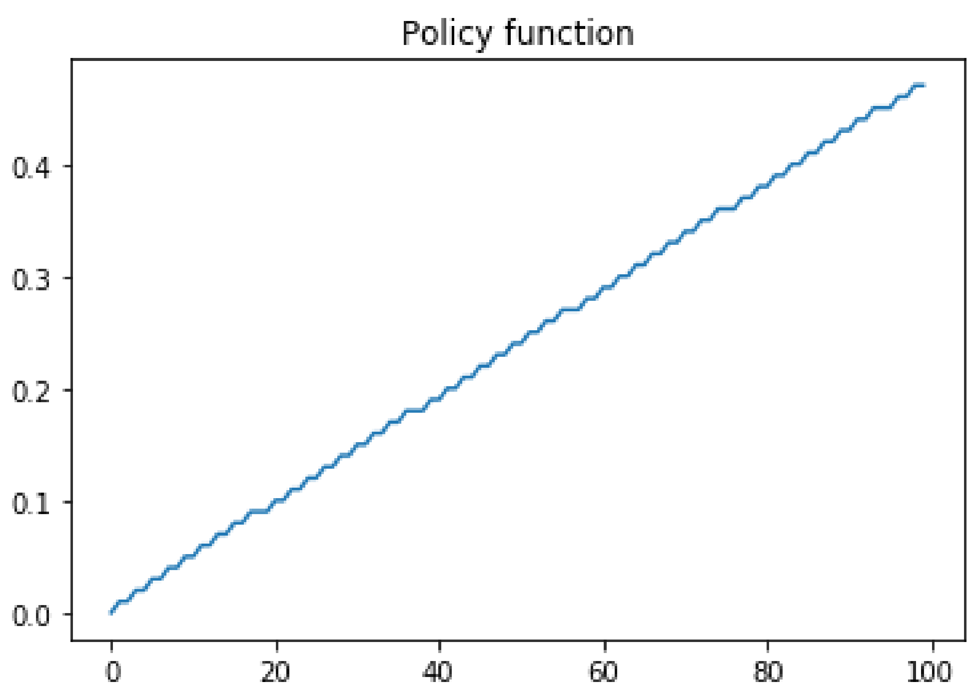
1.1.11 5.11

In [8]:
```
Value_T_plus_1 = np.zeros(100)
delta_T = np.sum((Value_T - Value_T_plus_1) ** 2)
print("The value of distance metric is", delta_T)
```

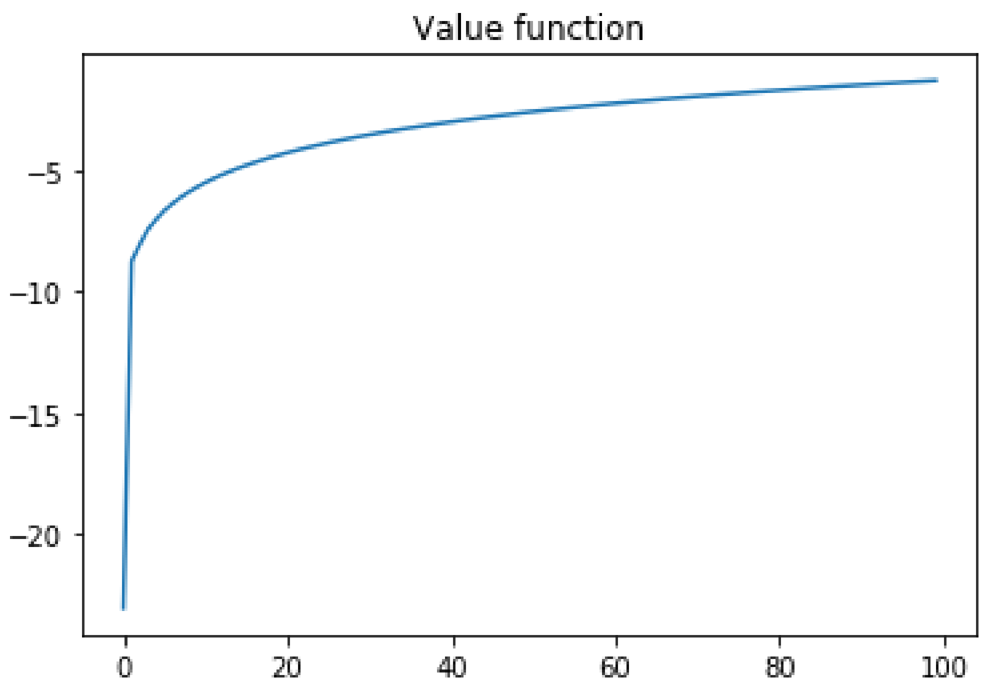
```
The value of distance metric is 178.92611065972804
```

1.1.12 5.12

In [9]:
```
Value_T_minus_1 = np.zeros(N)
psi_T_minus_1 = np.zeros(N)
VT1_matrix = np.tile(Value_T.reshape(1,N),(N,1))
VT1_matrix[c_mat<0] = -1e+10
for i in range(N):
    Value_T_minus_1[i] = -1e+10
    for j in range(N):
        if u_mat[i,j] + beta * VT1_matrix[i,j] > Value_T_minus_1[i]:
            psi_T_minus_1[i] = W[j]
            Value_T_minus_1[i] = u_mat[i,j] + beta * VT1_matrix[i,j]
    for i in range(N):
        if psi_T_minus_1[i] >= W[i]:
            psi_T_minus_1[i] = W[i] - 0.01
delta_T_minus_1 = np.sum((Value_T - Value_T_minus_1) ** 2)
plt.plot(psi_T_minus_1)
plt.title("Policy function")
plt.show()
```



In [10]:
```
plt.plot(Value_T_minus_1)
plt.title("Value function")
plt.show()
```



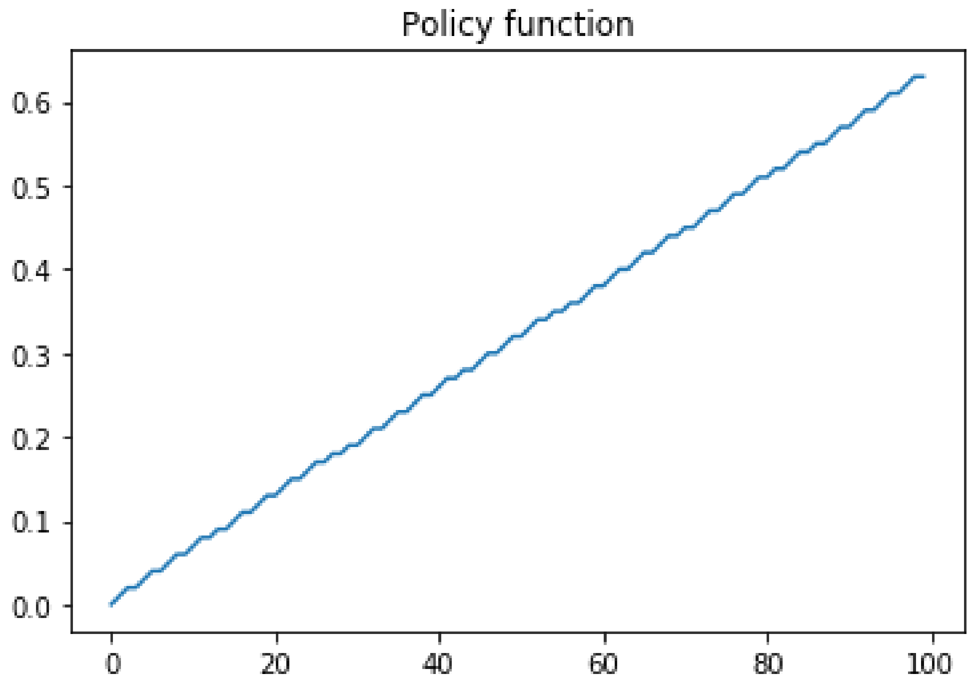
In [11]:
```
print("The distance metric is", delta_T_minus_1)
```

```
The distance metric is 857.3772627769418
```

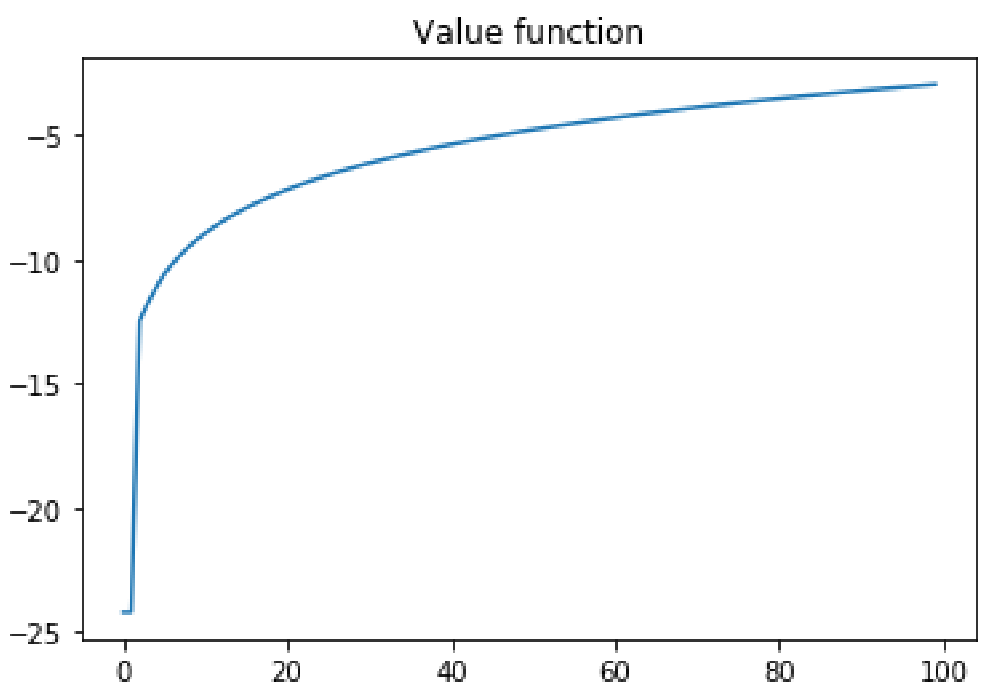
From the above results, it is quite clear that the distance metric is bigger than former one.

1.1.13 5.13

In [12]:
```
Value_T_minus_2=np.zeros(N)
psi_T_minus_2 = np.zeros(N)
VT2_matrix = np.tile(Value_T_minus_1.reshape(1,N),(N,1))
VT2_matrix[c_mat<0] = -1e+10
for i in range(N):
    Value_T_minus_2[i] = -1e+10
    for j in range(N):
        if u_mat[i,j] + beta * VT2_matrix[i,j] > Value_T_minus_2[i]:
            psi_T_minus_2[i] = W[j]
            Value_T_minus_2[i] = u_mat[i,j] + beta * VT2_matrix[i,j]
    for i in range(N):
        if psi_T_minus_2[i] >= W[i]:
            psi_T_minus_2[i] = W[i]-0.01
delta_T_minus_2 = np.sum((Value_T_minus_1-Value_T_minus_2) ** 2)
plt.plot(psi_T_minus_2)
plt.title("Policy function")
plt.show()
```




```
In [13]: plt.plot(Value_T_minus_2)
         plt.title("Value function")
         plt.show()
```



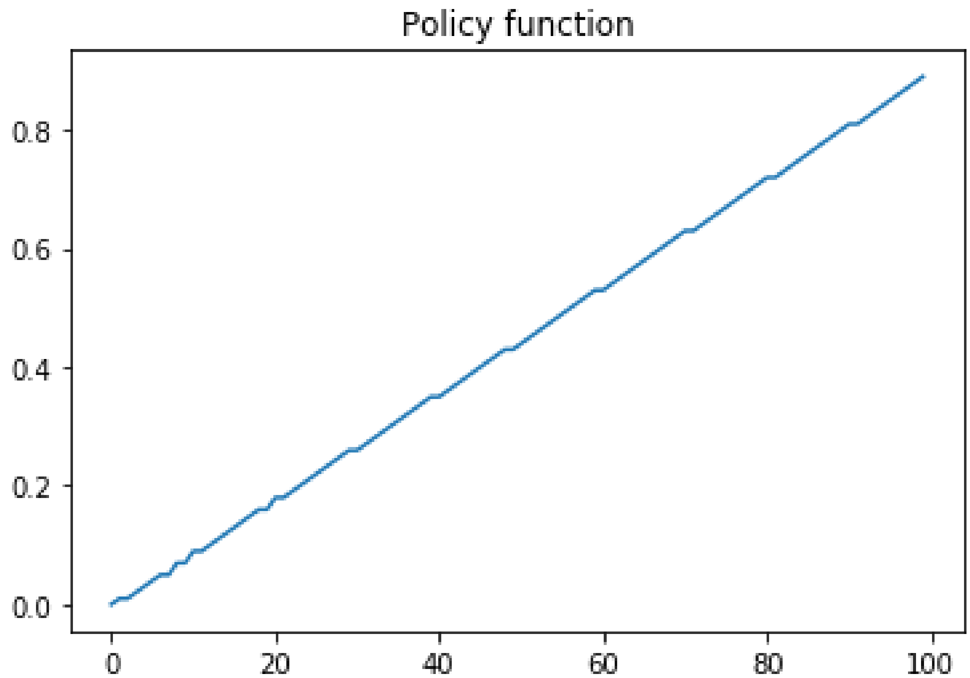
In [14]:
```python
print("The distance metric is", delta_T_minus_2)
```

```
The distance metric is 839.2935802021861
```

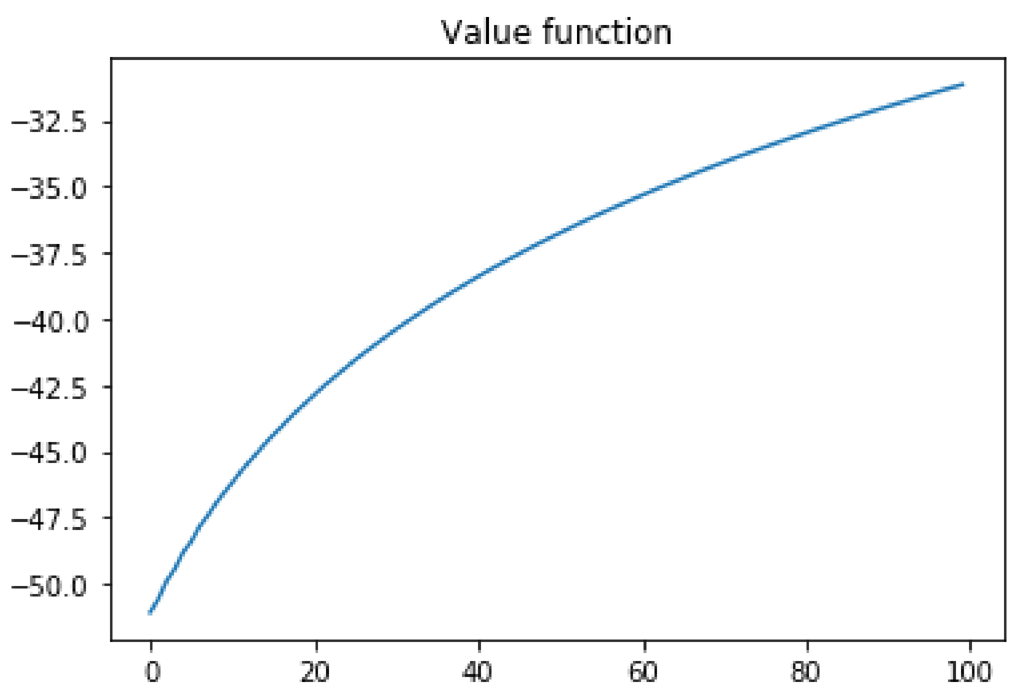
From the above result, we could see that the distance metric decreases from T-1 to T-2. Thus, it first increases from T to T-1 and then decrease from T-1 to T-2.

1.1.14 5.14

In [15]:
```python
def optimize(init = W, u = lambda x: np.log(x), beta = 0.9, error = 1e-9, max_loop =
1000):
    W = init
    V = np.log(W)
    N = 100
    c_mat = W.reshape(-1,1)-W
    c_mat[c_mat<=0] = 1e-10
    u_mat = u(c_mat)
    Error = 1
    count = 0
    while Error>error and count <= max_loop:
        count += 1
        new_W = np.array(W)
        new_V = np.array(V)
        V_matrix = np.tile(V.reshape(1,N),(N,1))
        V_matrix[c_mat<=0] = -1e+10
        for i in range(N):
            new_V[i] = -1e+10
            for j in range(N):
                if u_mat[i,j] + beta * V_matrix[i,j] > new_V[i]:
                    new_W[i] = W[j]
                    new_V[i] = u_mat[i,j] + beta * V_matrix[i,j]
            for i in range(N):
                if new_W[i] > W[i]:
                    new_W[i] = W[i]-0.01
        Error = ((V-new_V) ** 2).sum()
        V = new_V
    return new_V,new_W
V, new_W = optimize()
plt.plot(new_W)
plt.title("Policy function")
plt.show()
```

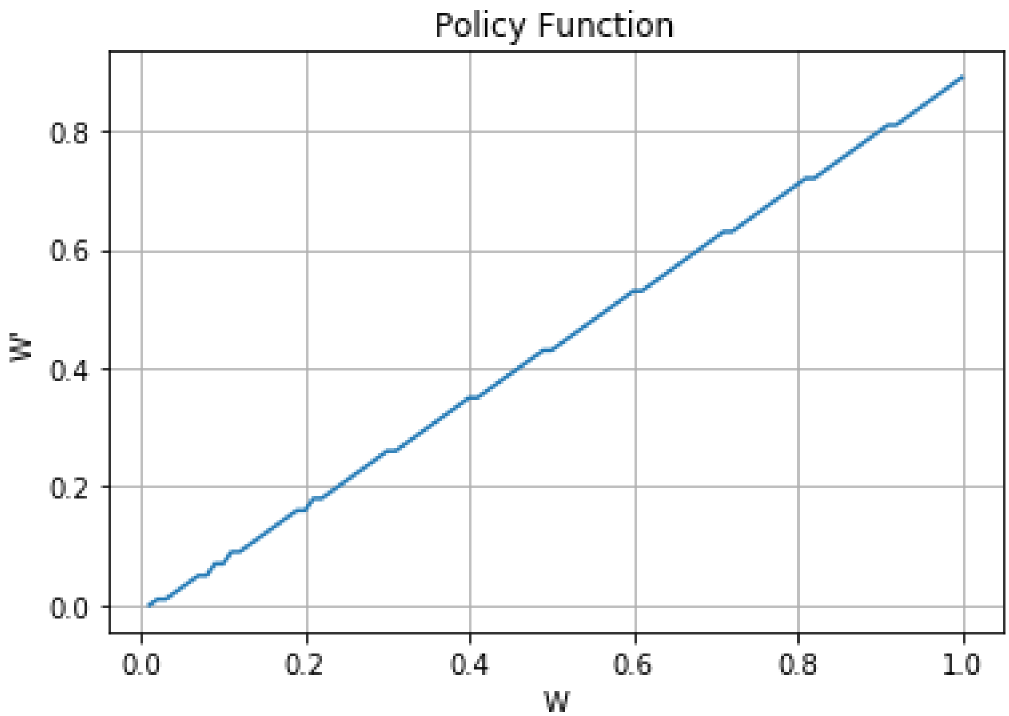



```
In [16]: plt.plot(V)
         plt.title("Value function")
         plt.show()
```



1.1.15 5.15

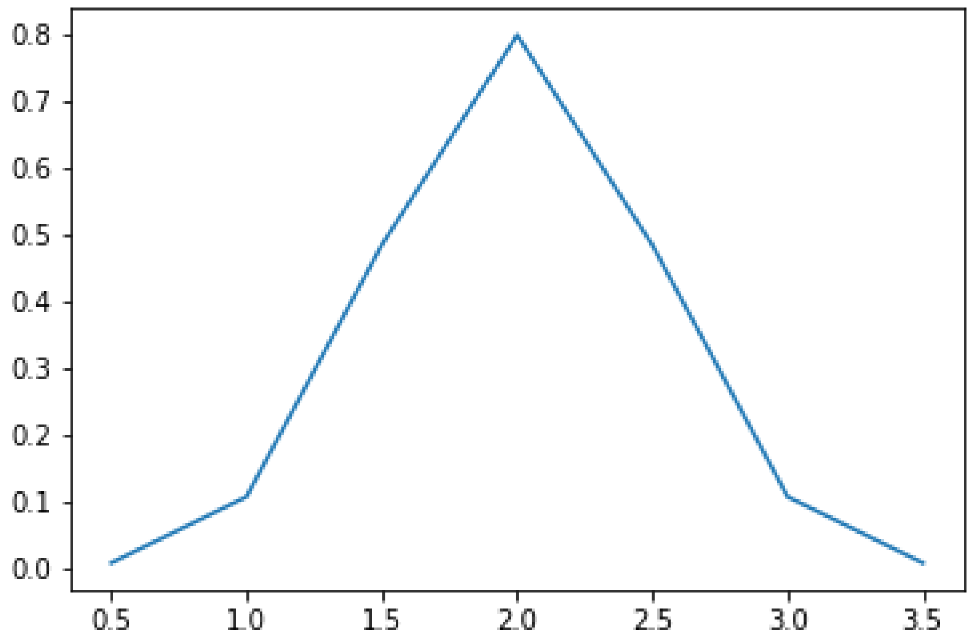
In [17]:
```
fig,ax = plt.subplots()
ax.plot(W,new_W)
ax.set_xlabel("W")
ax.set_ylabel("W'")
ax.set_title("Policy Function")
ax.grid()
```



1.1.16 5.16

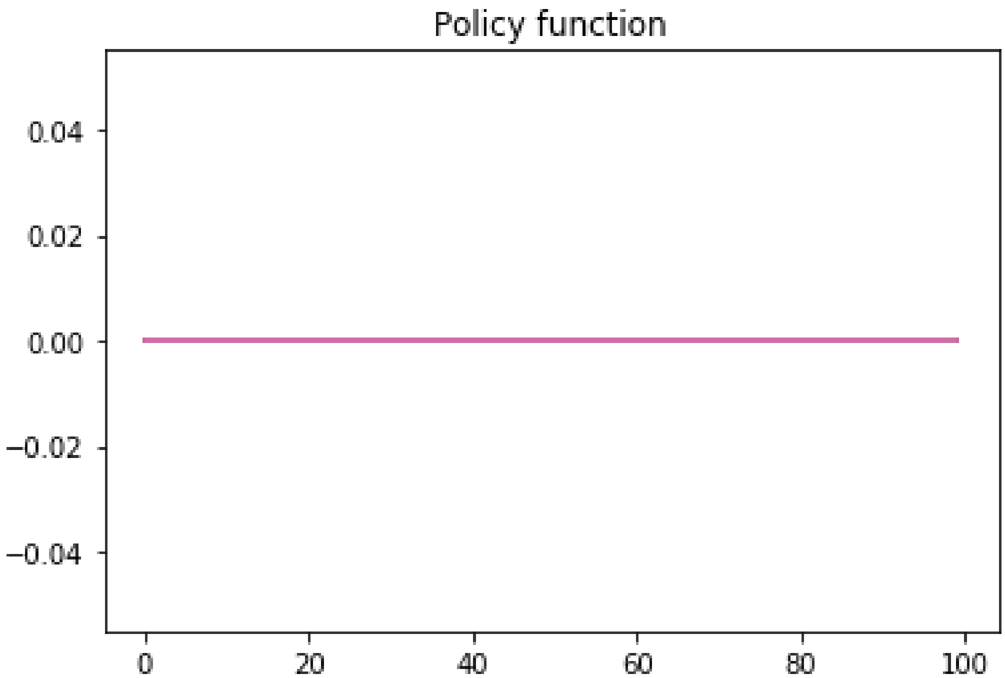
In [18]:
```
from scipy.stats import norm
```

In [19]:
```
sigma = 0.5
mu = 4*sigma
M=7
epsilon = np.linspace(mu-3*sigma, mu+3*sigma, M)
PDF = lambda x: norm(loc = mu, scale = sigma).pdf(x)
pdf = PDF(epsilon)
plt.plot(epsilon,pdf)
plt.show()
```

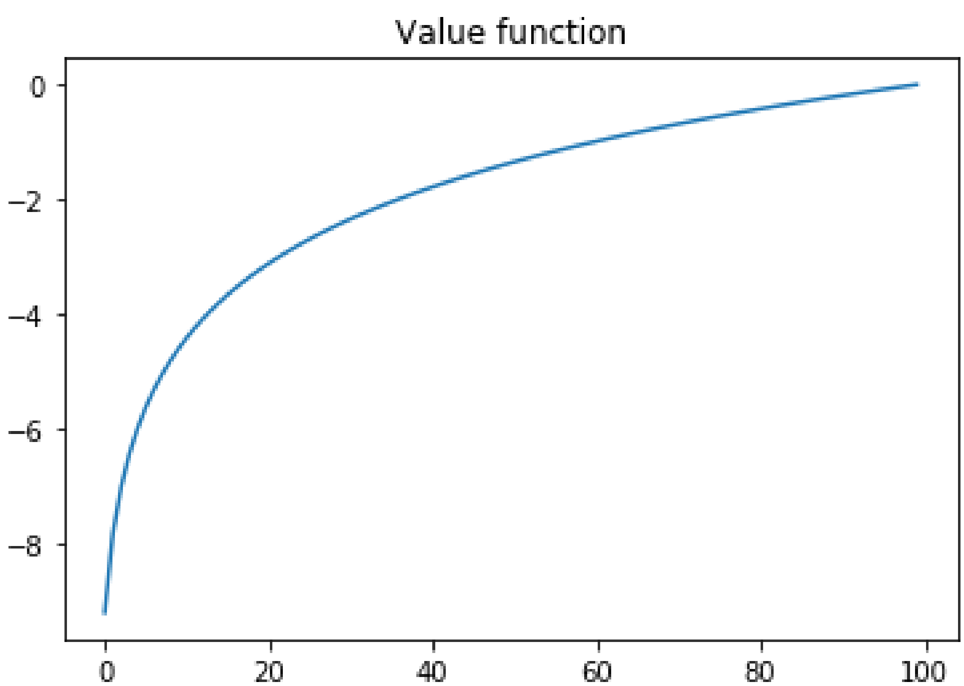


1.1.17 5.17

In [20]:
```
new_psi_T = np.zeros((100,7))
new_Value_T = np.zeros((100,7))
for j in range(7):
    e = epsilon[j]
    for i in range(100):
        w = W[i]
        new_value_func = lambda x: -e*np.log(w-x)
        new_psi_T[i,j] = max(float(opt.fmin(new_value_func, 0, disp = 0)),0)
        new_Value_T[i,j] = e * np.log(w-psi_T[i])
plt.plot(new_psi_T)
plt.title("Policy function")
plt.show()
```



In [21]:
```
plt.plot(np.average(new_Value_T, axis=1))
plt.title("Value function")
plt.show()
```



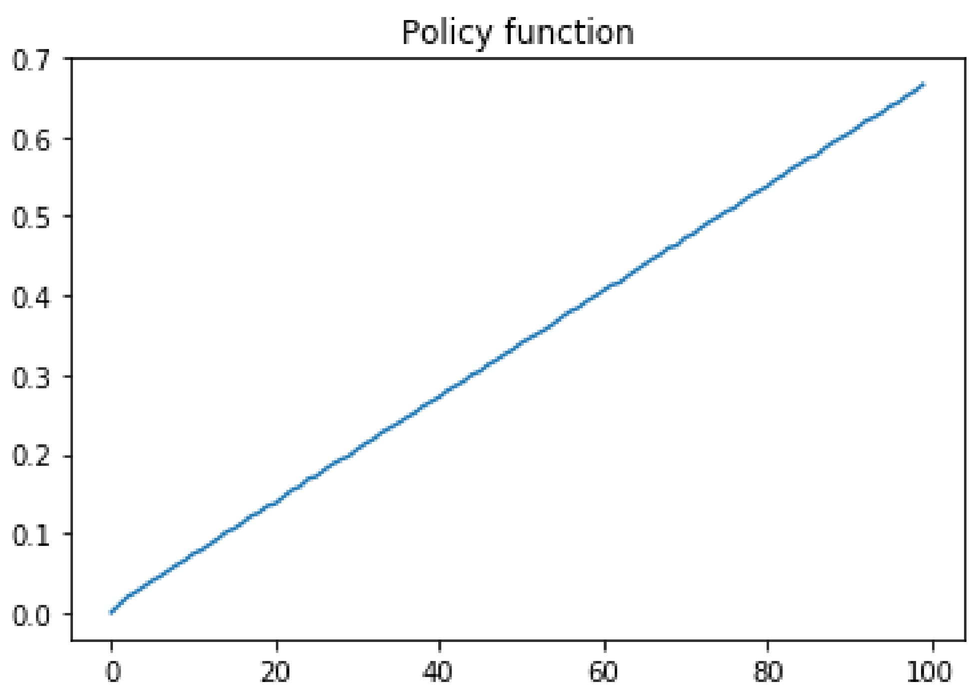
1.1.18 5.18

```
In [22]: new_Value_T_plus_1 = np.zeros((100,7))
         new_delta_T = np.sum((new_Value_T - new_Value_T_plus_1) ** 2)
         print("The distance metric is", new_delta_T)
```

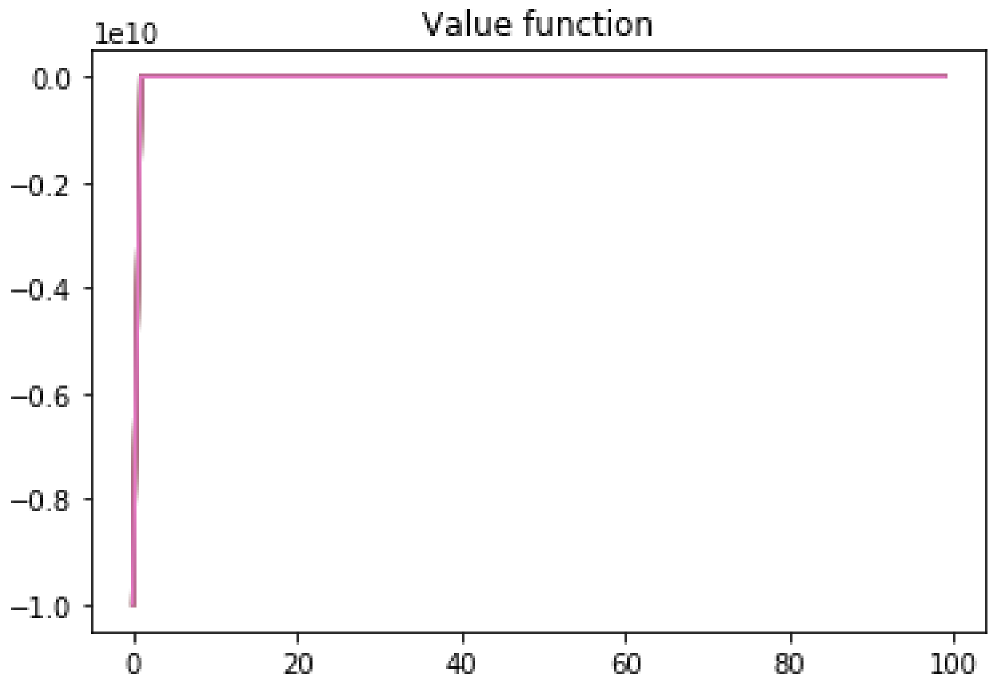
```
The distance metric is 6262.4138730904815
```

1.1.19 5.19

```
In [23]: new_Value_T_minus_1 = np.zeros((100,7))
         new_psi_T_minus_1 = np.zeros((100,7))
         for j in range(7):
             e = epsilon[j]
             for i in range(N):
                 w = W[i]
                 new_Value_T_minus_1[i,j] = -1e+10
                 for k in range(i):
                     v_value = e * u_mat[i,k] + beta * sum(pdf[s] * new_Value_T[k,s] for s in
         range(7))
                     if v_value > new_Value_T_minus_1[i,j]:
                         new_psi_T_minus_1[i,j] = W[k]
                         new_Value_T_minus_1[i,j] = v_value
         new_delta_T_minus_1 = np.sum((new_Value_T - new_Value_T_minus_1) ** 2)
         plt.plot(np.average(new_psi_T_minus_1, axis=1))
         plt.title("Policy function")
         plt.show()
```




```
In [24]: plt.plot(new_Value_T_minus_1)
         plt.title("Value function")
         plt.show()
```



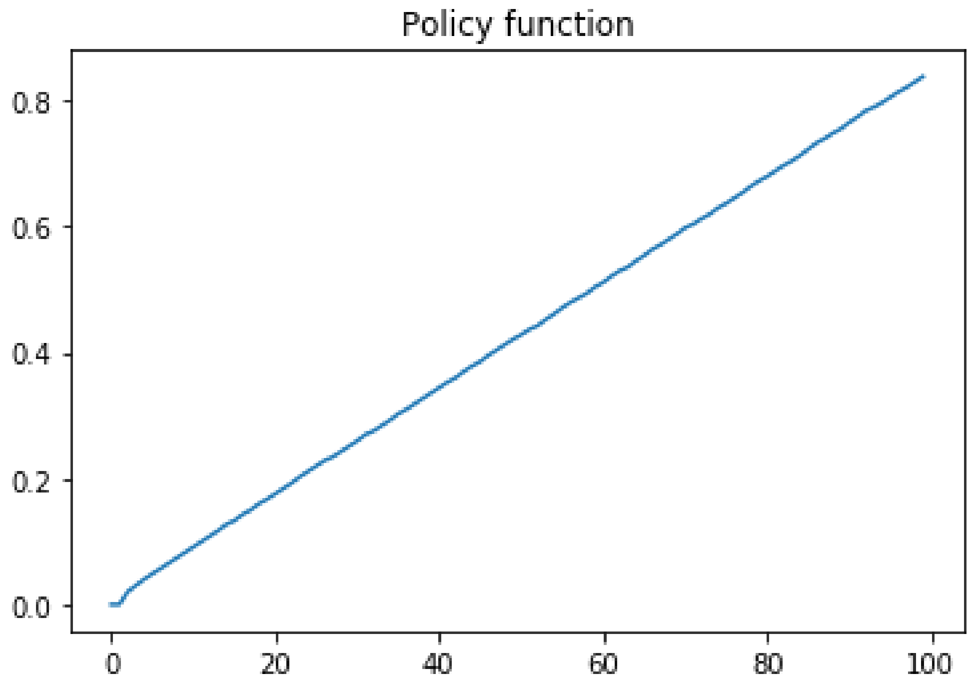

```
In [25]: print("The distance metric is", new_delta_T_minus_1)
```

```
The distance metric is 6.999999987105525e+20
```

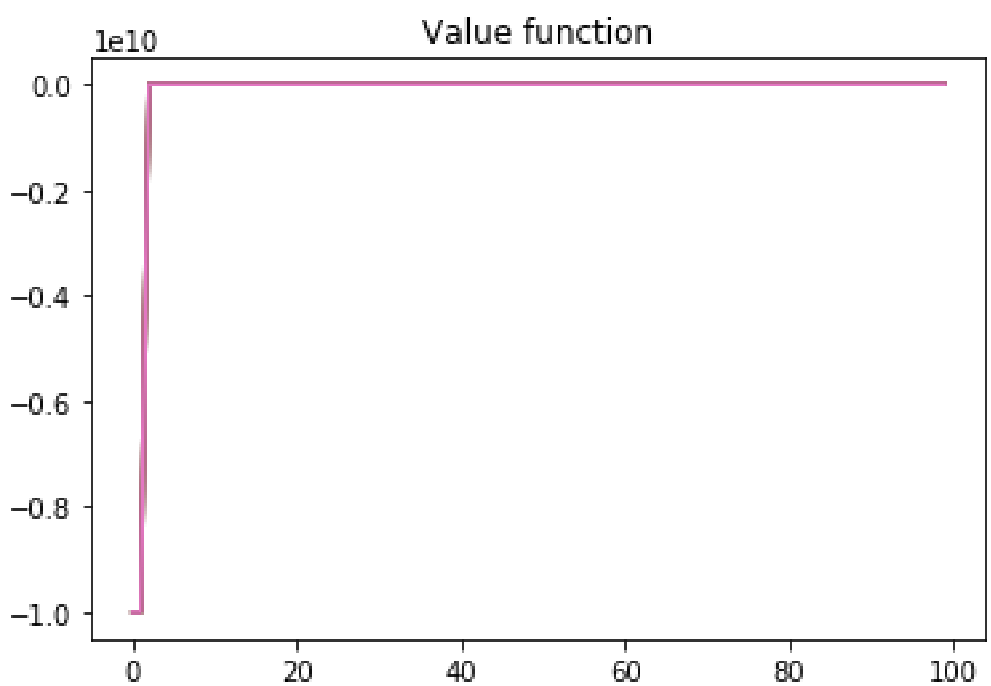
We could clearly see that the distance metric increases a lot.

1.1.20 5.20

```
In [26]: new_Value_T_minus_2 = np.zeros((100,7))
         new_psi_T_minus_2 = np.zeros((100,7))
         for j in range(7):
             e = epsilon[j]
             for i in range(100):
                 w = W[i]
                 new_Value_T_minus_2[i,j] = -1e+10
                 for k in range(i):
                     v_value = e * u_mat[i,k] + beta * sum(pdf[s] * new_Value_T_minus_1[k,s] for
         s in range(7))
                     if v_value > new_Value_T_minus_2[i,j]:
                         new_psi_T_minus_2[i,j] = W[k]
                         new_Value_T_minus_2[i,j] = v_value
         new_delta_T_minus_2 = np.sum((new_Value_T_minus_1-new_Value_T_minus_2) ** 2)
         plt.plot(np.average(new_psi_T_minus_2, axis=1))
         plt.title("Policy function")
         plt.show()
```




```
In [27]: plt.plot(new_Value_T_minus_2)
         plt.title("Value function")
         plt.show()
```



In [28]:
```
print("The distance metric is", new_delta_T_minus_2)
```

```
The distance metric is 6.999999963901749e+20
```

From the above result, we could see that we reach the same conclusion. The distance metric decreases from T-1 to T-2. Thus, it first increases from T to T-1 and then decrease from T-1 to T-2.

1.1.21 5.21

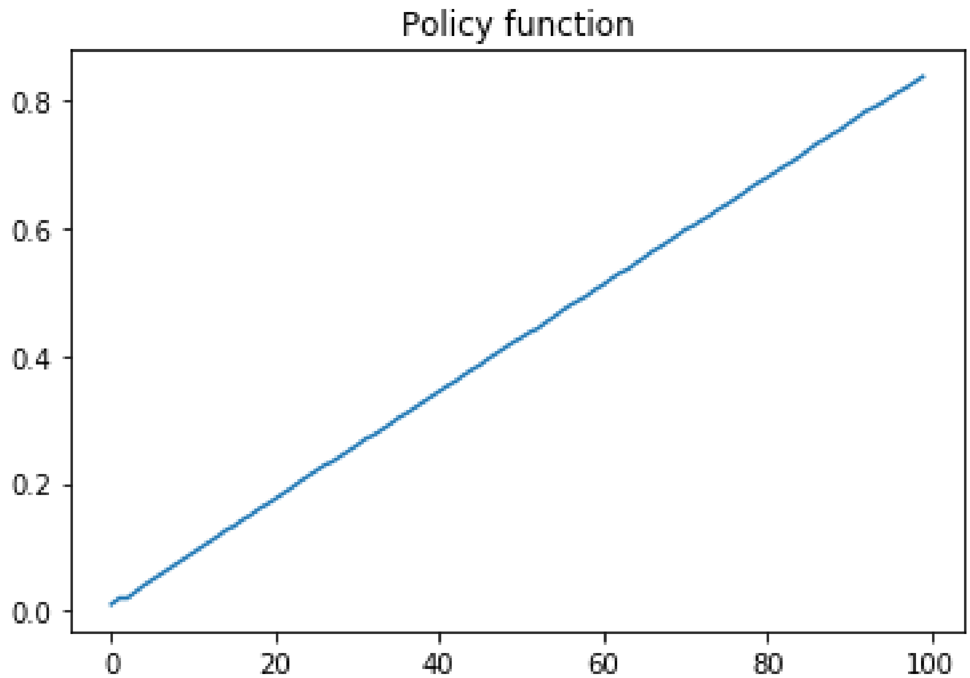
In [29]:
```
import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

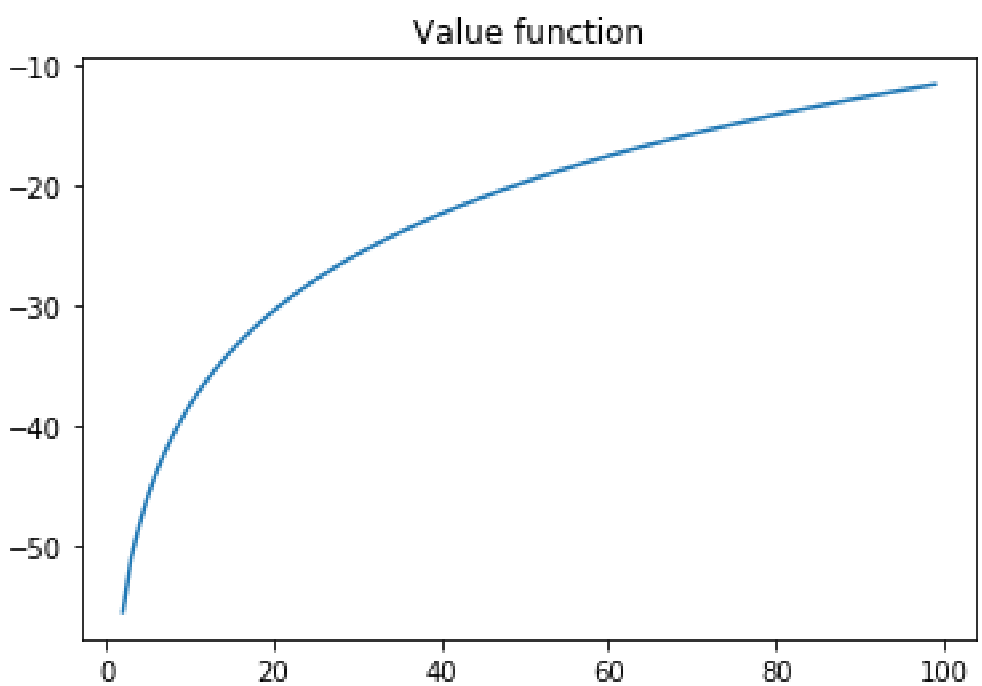
In [30]:
```
def new_optimize(init=W, E = epsilon, P = pdf, u = lambda x: np.log(x), beta = 0.9,
error = 1e-9, max_loop = 1000):
    W = init
    pdf = P
    epsilon = E
    V = np.zeros((W.size, pdf.size))
    for i in range(pdf.size):
        V[:,i] = epsilon[i] * np.log(W)
    c_mat = W.reshape(-1,1)-W
    c_mat[c_mat<=0] = 1e-10
    u_mat = u(c_mat)
    Error = 1
    count = 0
    while Error>error and count <= max_loop:
        count += 1
        new_W = np.tile(W.reshape(-1,1),(1,7))
        new_V = np.array(V)
        for j in range(7):
            e = epsilon[j]
            for i in range(100):
                w = W[i]
                new_V[i,j] = -np.inf
                for k in range(i):
                    v_value = e * u_mat[i,k] + beta*sum(pdf[s] * V[k,s] for s in
range(7))
                    if v_value > new_V[i,j]:
                        new_W[i,j] = W[k]
                        new_V[i,j] = v_value
        Error = ((V-new_V) ** 2).sum()
        V = new_V
    return new_V,new_W, count

V, new_W, count = new_optimize()
```

In [31]:
```
plt.plot(np.average(new_W, axis=1))
plt.title("Policy function")
plt.show()
```



In [32]:
```
plt.plot(np.average(V, axis=1))
plt.title("Value function")
plt.show()
```



```
In [33]: print('It takes',count,'iterations')
```

```
It takes 34 iterations
```

1.1.22 5.22

```
In [34]: from mpl_toolkits.mplot3d import Axes3D
```

```
In [35]: %matplotlib notebook
         X, Y = np.meshgrid(W, epsilon)
         new_fig = plt.figure(figsize=(7, 5))
         new_ax = new_fig.add_subplot(111, projection='3d')
         new_ax.plot_surface(X.T, Y.T, new_W)
         new_ax.set_xlabel("Cake today")
         new_ax.set_ylabel("Taste shock today")
         new_ax.set_title("Converged Policy Function")
         new_ax.view_init(elev=60,azim=60)
         plt.show()
```