Exercise 1:

For 1 period:

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

Exercise 2:

For 2 periods:

$$max_{W_2 \in [0,W_1]} \{\beta^0 u(W_1 - W_2) + max_{W_3 \in [0,W_2]} \beta^1 u(W_2 - W_3)\}$$

Exercise 3:

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

(equation 1)

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

(equation 2)

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

(equation 3)

From equation 1, we know

$$W_4 = 0$$

From equation 2, using FOC, we can calculate

$$W_3 = 0.4737 W_2$$

From equation 3, using FOC, we can calculate

$$W_2 = 0.6310 W_1$$

Since

$$W_1 = 1$$

, we can find out the results are:

$$W_1 = 1, W_2 = 0.631, W_3 = 0.299, W_4 = 0$$

So,

$$C_1 = 0.369, C_2 = 0.332, C_3 = 0.299$$

Exercise 4:

It can be deduced from (7) that,

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

And notice (6),

$$V_T(W_T) = u(W_T)$$

So we can get,

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

Exercise 5:

Since u(c) = In(c), we know:

$$V_{T-1}(\bar{W}) = In(\bar{W} - W_T) + \beta In(W_T)$$

Using F.O.C, we calculate:

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

So we can get,

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

We know,

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

Therefore,

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

Exercise 6:

Because

$$V_{T-2}(W_{T-2}) = In(W_{T-2} - W_{T-1}) + \beta V_{T-1}(W_{T-1})$$

(equation 4)

From Exercise 5, we know:

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

Applying F.O.C to equation 4, we get:

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

Therefore,

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

Exercise 7:

Notice that exercise 5 and 6's answers can be written as:

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

So we can deduce that:

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

$$V_{T-S}(W_{T-S}) = \sum_{i=0}^{s} \beta^i In(\frac{\beta^i W_{T-S}}{1 + \sum_{i=1}^{s} \beta^i})$$

Thus,

$$\lim_{s \to +\infty} \psi_{T-S}(W_{T-S}) = \beta W_{T-S} = \psi(W_{T-S})$$
$$\lim_{s \to +\infty} V_{T-S}(W_{T-S}) = \frac{In((1-\beta)W_{T-S})}{1-\beta} + \frac{\beta}{(1-\beta)^2} In(\beta) = V(W_{T-S})$$

Exercise 8:

Let w be the cake left for tomorrow.

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

Exercise 9:

In [1]:
```
import numpy as np
min = 0.01
max = 1
W_vec = np.linspace(min, max , 100)
W_vec
```

Out[1]:
```
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.  ])
```

Exercise 10:

In [74]:
```
import numpy as np
beta = 0.9
N = 100

def utility(c):
    u = np.log(c)
    return u

v_t_add_1 = np.zeros((N, N))# V(W') = [0,0...0]
```

In [75]:
```
c_matrix = (np.tile(W_vec.reshape((N, 1)), (1, N)) -
          np.tile(W_vec.reshape((1, N)), (N, 1)))
c_matrix
```

Out[75]:
```
array([[ 0.  , -0.01, -0.02, ..., -0.97, -0.98, -0.99],
       [ 0.01,  0.  , -0.01, ..., -0.96, -0.97, -0.98],
       [ 0.02,  0.01,  0.  , ..., -0.95, -0.96, -0.97],
       ...,
       [ 0.97,  0.96,  0.95, ...,  0.  , -0.01, -0.02],
       [ 0.98,  0.97,  0.96, ...,  0.01,  0.  , -0.01],
       [ 0.99,  0.98,  0.97, ...,  0.02,  0.01,  0.  ]])
```

In [76]:
```
c_t = c_matrix > 0
c_matrix[~c_t] = 1e-10
c_matrix # W-W'
```

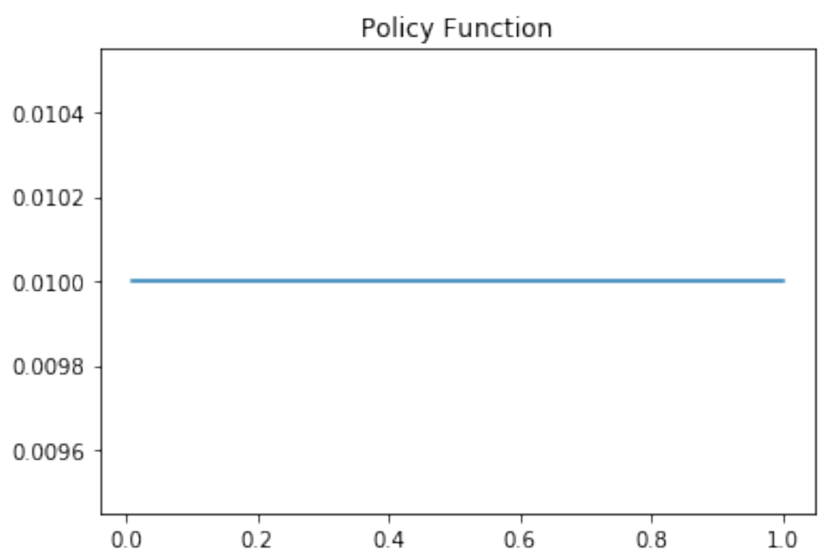
Out[76]:
```
array([[1.0e-10, 1.0e-10, 1.0e-10, ..., 1.0e-10, 1.0e-10, 1.0e-10],
       [1.0e-02, 1.0e-10, 1.0e-10, ..., 1.0e-10, 1.0e-10, 1.0e-10],
       [2.0e-02, 1.0e-02, 1.0e-10, ..., 1.0e-10, 1.0e-10, 1.0e-10],
       ...,
       [9.7e-01, 9.6e-01, 9.5e-01, ..., 1.0e-10, 1.0e-10, 1.0e-10],
       [9.8e-01, 9.7e-01, 9.6e-01, ..., 1.0e-02, 1.0e-10, 1.0e-10],
       [9.9e-01, 9.8e-01, 9.7e-01, ..., 2.0e-02, 1.0e-02, 1.0e-10]])
```

In [77]:
```
u = utility(c_matrix) # u(W - W')
```

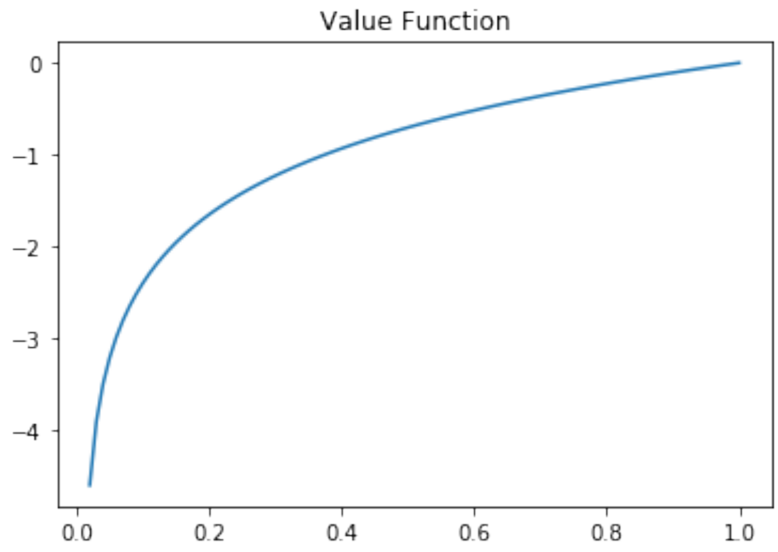
In [78]:
```
v_t_add_1[~c_t] = -1e10
v = u + beta * v_t_add_1 # V(W) = max u(W - W') + beta * V(W')
v_t = v.max(axis = 1)
index = np.argmax(v, axis = 1)
W_vec[index]
```

Out[78]:
```
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 [79]:
```
import matplotlib.pyplot as plt
plt.plot(W_vec,W_vec[index])
plt.title("Policy Function")
plt.show()
```



In [80]:
```
plt.plot(W_vec[1:], v_t[1:]) #To make the picture more readable
plt.title("Value Function")
plt.show()
```



Exercise 11:

In [83]:
```
sigma_t = ((v_t) ** 2).sum() # compare VT(W) with VT+1(W')
sigma_t
```

Out[83]: 8.100000041446531e+19

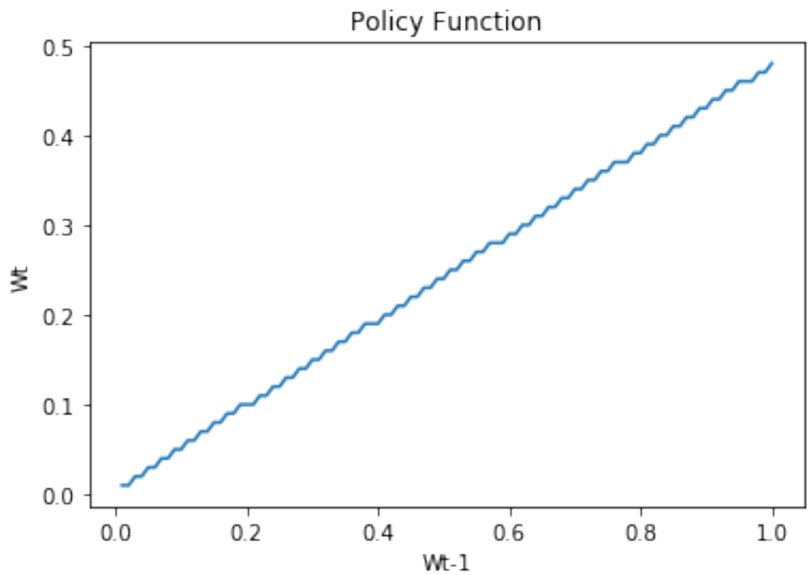
Exercise 12:

In [84]:
```
c_matrix = (np.tile(W_vec.reshape((N, 1)), (1, N)) -
          np.tile(W_vec.reshape((1, N)), (N, 1)))
```

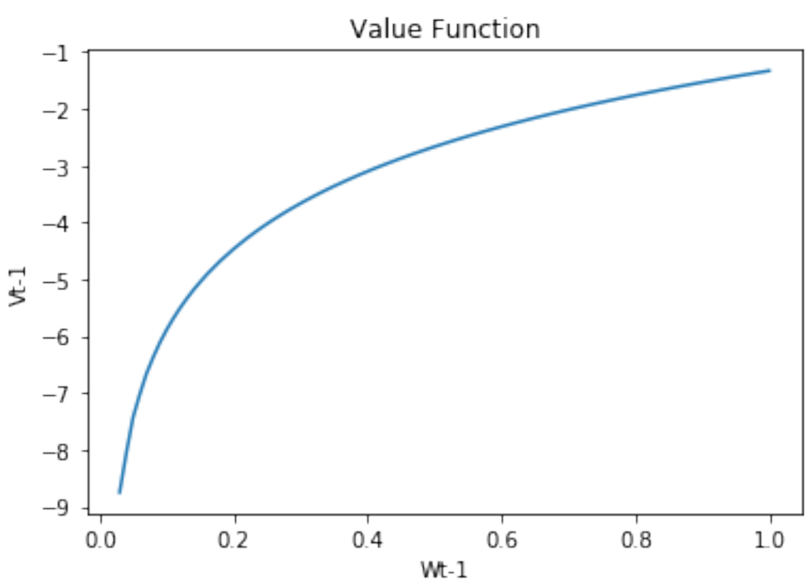
In [85]:
```
v_t_new = np.tile(v_t.reshape((1,N)),(N,1))
v_t_new[c_matrix <= 0] = -1e10 # To opt the condition W - W' < 0 out o
f consideration
v_t_minus1 = u + beta * v_t_new # utility matrix doesn't change
v_t_minus_1 = v_t_minus1.max(axis = 1)
index = np.argmax(v_t_minus1, axis = 1)
W_vec[index]
```

Out[85]:
```
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 [86]:
```
plt.plot(W_vec,W_vec[index])
plt.title("Policy Function")
plt.xlabel('Wt-1')
plt.ylabel('Wt')
plt.show()
```



In [87]:
```
plt.plot(W_vec[2:], v_t_minus_1[2:]) #To make the picture more readabl
e
plt.title("Value Function")
plt.xlabel('Wt-1')
plt.ylabel('Vt-1')
plt.show()
```



In [88]:
```
sigma_tminus1 = np.sum((v_t_minus_1 - v_t) ** 2)
sigma_tminus1
```

Out[88]:
```
6.56100003357169e+19
```

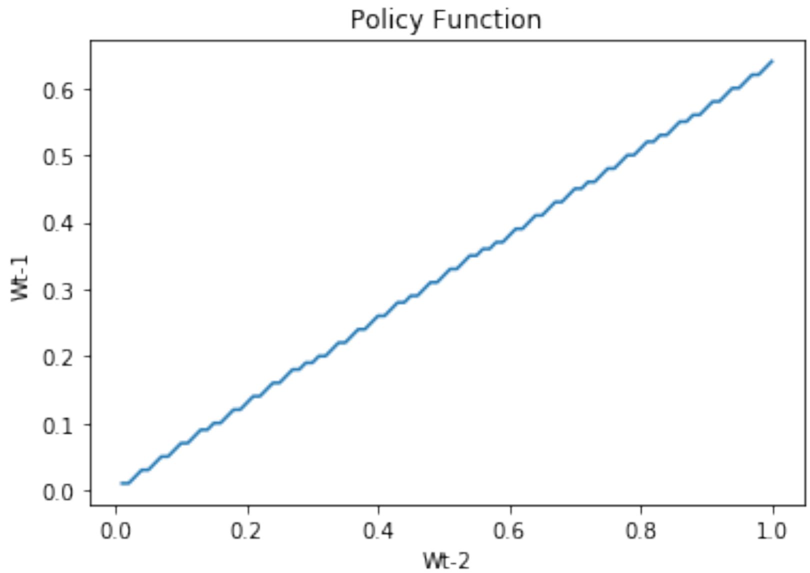
The sigma T-1 is much smaller than the sigma T.

Exercise 13

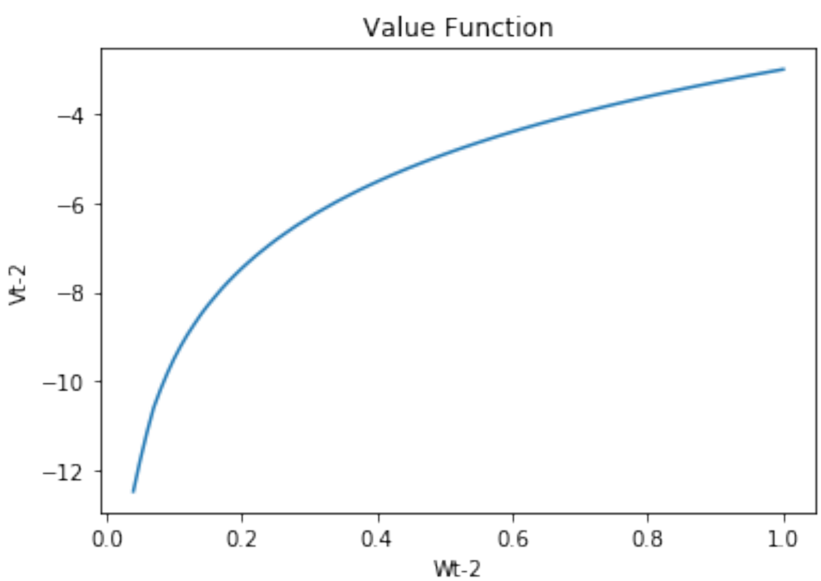
In [89]:
```
v_t_new = np.tile(v_t_minus_1.reshape((1,N)),(N,1))
v_t_new[c_matrix <= 0] = -1e10 # To opt the condition W - W' < 0 out o
f consideration
v_t_minus2 = u + beta * v_t_new # utility matrix doesn't change
v_t_minus_2 = v_t_minus2.max(axis = 1)
index = np.argmax(v_t_minus2, axis = 1)
W_vec[index]
```

Out[89]:
```
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 [90]:
```
plt.plot(W_vec,W_vec[index])
plt.title("Policy Function")
plt.xlabel('Wt-2')
plt.ylabel('Wt-1')
plt.show()
```



In [91]:
```
plt.plot(W_vec[3:], v_t_minus_2[3:])
plt.title("Value Function")
plt.xlabel('Wt-2')
plt.ylabel('Vt-2')
plt.show()
```



In [99]:
```
sigma_tminus2 = np.sum((v_t_minus_2 - v_t_minus_1) ** 2)
sigma_tminus2
```

Out[99]: 5.314410027193069e+19

As we can see,

$$\sigma_{T-2} < \sigma_{T-1} < \sigma_T$$

Exercise 14:

In [105]:
```
v_0 = v_t_add_1
i = 0
sigma = 1
v_1 = np.zeros(N).reshape(N,1)
while sigma >= 1e-9 and i < 500:
    v0 = v_1
    v_0 = np.tile(v_1.reshape((1,N)),(N,1))
    c_matrix = (np.tile(W_vec.reshape((N, 1)), (1, N)) -
         np.tile(W_vec.reshape((1, N)), (N, 1)))
    c_t = c_matrix <= 0
    v_0[c_t] = -1e10
    v1 = u + beta * v_0
    v_1 = v1.max(axis = 1)
    index = np.argmax(v1, axis = 1)
    sigma = np.sum((v_1 - v0) ** 2)
    i += 1
    print("iteration =", i, "sigma =", sigma)
print ("This covergency takes", i ,"iterations.")
```

```
iteration = 1 sigma = 8.100000041446531e+21
iteration = 2 sigma = 6.56100003357169e+19
iteration = 3 sigma = 5.314410027193069e+19
iteration = 4 sigma = 4.304672122026387e+19
iteration = 5 sigma = 3.486784418841374e+19
iteration = 6 sigma = 2.824295379261513e+19
iteration = 7 sigma = 2.2876792572018258e+19
iteration = 8 sigma = 1.853020198333479e+19
iteration = 9 sigma = 1.5009463606501175e+19
iteration = 10 sigma = 1.2157665521265955e+19
iteration = 11 sigma = 9.847709072225423e+18
iteration = 12 sigma = 7.976644348502596e+18
iteration = 13 sigma = 6.461081922287103e+18
iteration = 14 sigma = 5.233476357052553e+18
iteration = 15 sigma = 4.239115849212569e+18
iteration = 16 sigma = 3.4336838378621814e+18
iteration = 17 sigma = 2.7812839086683674e+18
```

```
iteration = 18 sigma = 2.2528399660213778e+18
iteration = 19 sigma = 1.8248003724773166e+18
iteration = 20 sigma = 1.4780883017066266e+18
iteration = 21 sigma = 1.197251524382368e+18
iteration = 22 sigma = 9.697737347497179e+17
iteration = 23 sigma = 7.855167251472716e+17
iteration = 24 sigma = 6.362685473692899e+17
iteration = 25 sigma = 5.1537752336912474e+17
iteration = 26 sigma = 4.17455793928991e+17
iteration = 27 sigma = 3.381391930824827e+17
iteration = 28 sigma = 2.7389274639681098e+17
iteration = 29 sigma = 2.2185312458141686e+17
iteration = 30 sigma = 1.7970103091094768e+17
iteration = 31 sigma = 1.4555783503786765e+17
iteration = 32 sigma = 1.1790184638067278e+17
iteration = 33 sigma = 9.550049556834494e+16
iteration = 34 sigma = 7.735540141035942e+16
iteration = 35 sigma = 6.265787514239115e+16
iteration = 36 sigma = 5.075287886533683e+16
iteration = 37 sigma = 4.110983188092282e+16
iteration = 38 sigma = 3.32989638235475e+16
iteration = 39 sigma = 2.6972160697073476e+16
iteration = 40 sigma = 2.1847450164629524e+16
iteration = 41 sigma = 1.7696434633349916e+16
iteration = 42 sigma = 1.4334112053013432e+16
iteration = 43 sigma = 1.1610630762940884e+16
iteration = 44 sigma = 9404610917982120.0
iteration = 45 sigma = 7617734843565517.0
iteration = 46 sigma = 6170365223288068.0
iteration = 47 sigma = 4997995830863340.0
iteration = 48 sigma = 4048376622999305.0
iteration = 49 sigma = 3279185064629437.5
iteration = 50 sigma = 2656139902349845.0
iteration = 51 sigma = 2151473320903374.8
iteration = 52 sigma = 1742693389931734.0
iteration = 53 sigma = 1411581645844705.2
iteration = 54 sigma = 1143381133134212.0
iteration = 55 sigma = 926138717838712.5
iteration = 56 sigma = 750172361449357.9
iteration = 57 sigma = 607639612773980.5
iteration = 58 sigma = 492188086346925.1
iteration = 59 sigma = 398672349941010.06
iteration = 60 sigma = 322924603452218.94
iteration = 61 sigma = 261568928796298.2
iteration = 62 sigma = 211870832325002.22
iteration = 63 sigma = 171615374183252.53
iteration = 64 sigma = 139008453088435.27
iteration = 65 sigma = 112596847001633.22
iteration = 66 sigma = 91203446071323.64
iteration = 67 sigma = 73874791317772.83
```

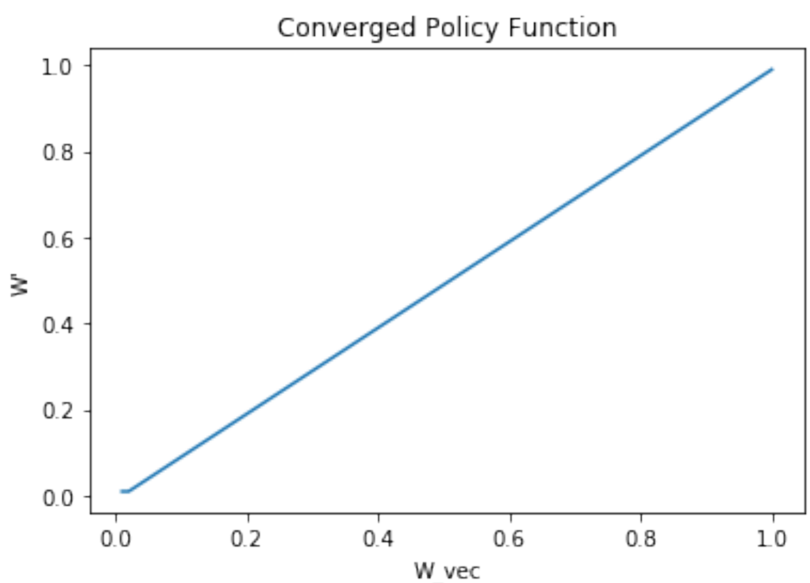
```
iteration = 68 sigma = 59838580967396.65
iteration = 69 sigma = 48469250583591.945
iteration = 70 sigma = 39260092972710.12
iteration = 71 sigma = 31800675307895.83
iteration = 72 sigma = 25758546999396.25
iteration = 73 sigma = 20864423069511.586
iteration = 74 sigma = 16900182686304.996
iteration = 75 sigma = 13689147975907.654
iteration = 76 sigma = 11088209860485.797
iteration = 77 sigma = 8981449986994.084
iteration = 78 sigma = 7274974489465.778
iteration = 79 sigma = 5892729336467.846
iteration = 80 sigma = 4773110762539.508
iteration = 81 sigma = 3866219717657.537
iteration = 82 sigma = 3131637971303.128
iteration = 83 sigma = 2536626756756.045
iteration = 84 sigma = 2054667672972.888
iteration = 85 sigma = 1664280815108.5134
iteration = 86 sigma = 1348067460238.3552
iteration = 87 sigma = 1091934642793.5035
iteration = 88 sigma = 884467060663.1531
iteration = 89 sigma = 716418319137.5505
iteration = 90 sigma = 580298838501.7767
iteration = 91 sigma = 470042059186.77734
iteration = 92 sigma = 380734067941.5837
iteration = 93 sigma = 308394595032.9489
iteration = 94 sigma = 249799621976.9002
iteration = 95 sigma = 202337693801.46634
iteration = 96 sigma = 163893531979.29382
iteration = 97 sigma = 132753760903.24619
iteration = 98 sigma = 107530546331.53915
iteration = 99 sigma = 87099742528.32265
iteration = 100 sigma = 70550791447.55199
iteration = 101 sigma = 0.0
This covergency takes 101 iterations.
```

Exercise 15:

In [96]:
```
W_vec[index]
```

Out[96]:
```
array([0.01, 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])
```

In [97]:
```
plt.plot(W_vec,W_vec[index])
plt.title("Converged Policy Function")
plt.xlabel("W_vec")
plt.ylabel("W'")
plt.show()
```



Exercise 16:

In [30]:
```
from scipy.stats import norm

M = 7
sig = 0.5
mu = 4 * sig
E = np.linspace(mu - 3 * sig, mu + 3 * sig, M)
E
```

Out[30]:
```
array([0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5])
```

In [31]:
```
Pr = []

for i in range(0, M):
    if i == 0:
        Pr.append(norm.cdf((E[0] + E[1]) / 2, mu, sig))
    elif i == M-1:
        Pr.append(1 - norm.cdf((E[-2] + E[-1]) / 2, mu, sig))
    else:
        Emin = (E[i - 1] + E[i]) / 2
        Emax = (E[i] + E[i + 1]) / 2
        Pr.append(norm.cdf(Emax, mu, sig) - norm.cdf(Emin, mu, sig))
Pr
```

Out[31]:
```
[0.006209665325776132,
 0.06059753594308194,
 0.2417303374571288,
 0.38292492254802624,
 0.2417303374571288,
 0.060597535943081926,
 0.006209665325776159]
```

In [32]:
```
for i in range(0, M):
    print("The probablity of", E[i], "is", Pr[i])
```

```
The probablity of 0.5 is 0.006209665325776132
The probablity of 1.0 is 0.06059753594308194
The probablity of 1.5 is 0.2417303374571288
The probablity of 2.0 is 0.38292492254802624
The probablity of 2.5 is 0.2417303374571288
The probablity of 3.0 is 0.060597535943081926
The probablity of 3.5 is 0.006209665325776159
```

Exercise 17:

In [33]:
```
c_cube0 = (W_vec.reshape(N, -1) - W_vec).reshape(N, 1, N)
c_cube = np.tile(c_cube0, (1, M, 1))
c_neg = c_cube <= 0
c_cube[c_cube <= 0] = 1e-10
u_cube = utility(c_cube)
for i in range(M):
    u_cube[:, i, :] = u_cube[:, i, :] * E[i]
```

In [34]:
```
Pr = np.array(Pr)
v_tadd1 = np.zeros((N, M))
expected_vtadd1 = (v_tadd1 @ Pr.reshape(-1, 1)).reshape(1, 1, N)
expected_v_tadd1 = np.tile(expected_vtadd1, (N, M ,1))
```

In [35]:
```
expected_v_tadd1[c_neg] = -1e10
```

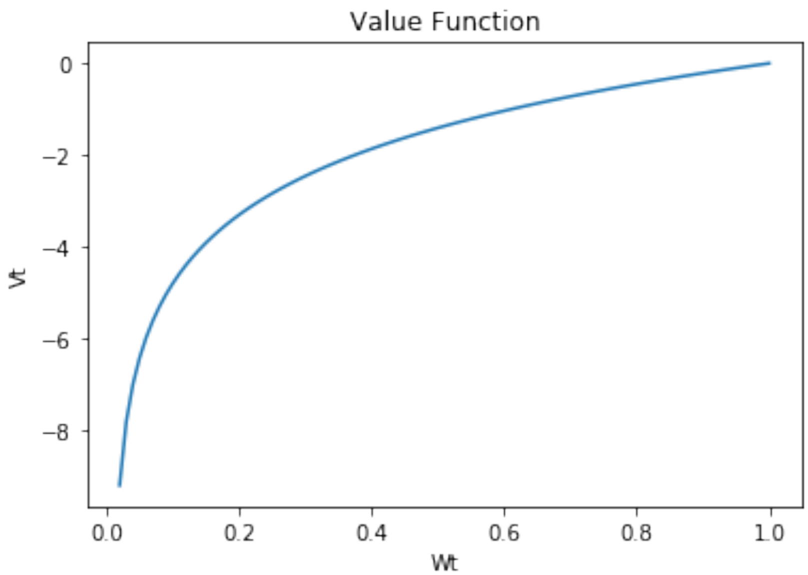
In [36]:
```
v_t_cube = u_cube + beta * expected_v_tadd1
vt_cube = np.max(v_t_cube, axis=2) #V(W, e)
index = np.argmax(v_t_cube, axis= 2)# W' = psi(W, e)
```

In [37]:
```
vt_cub = vt_cube @ Pr.reshape(-1, 1) #E(V(W, e)) = V'(W)
```

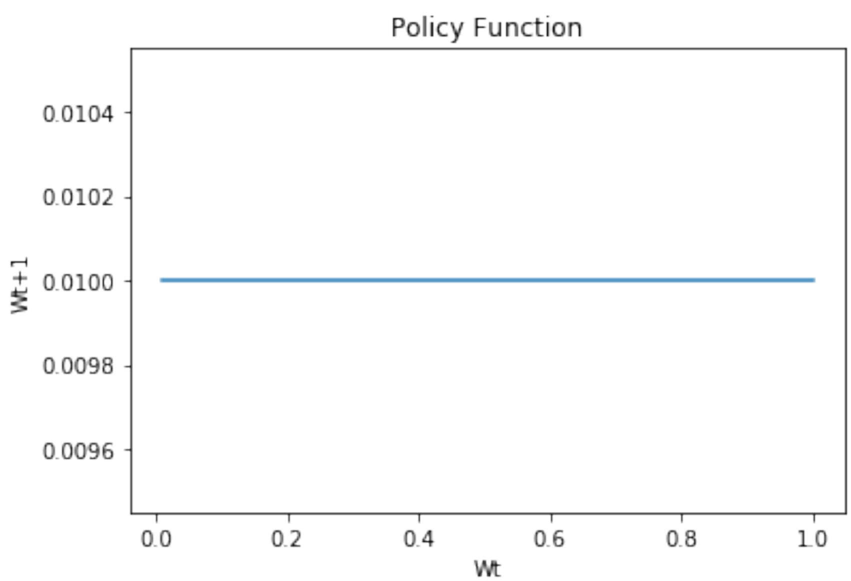
In [38]:
```
#index
```

In [39]:
```
W_tadd1 = np.zeros((N, M))
for i in range(M):
    W_tadd1[:, i] = W_vec[index[:, i]]
W_tadd1 = W_tadd1 @ Pr.reshape(-1, 1) #W' = E(psi(W, e))
```

In [41]:
```
plt.plot(W_vec[1:], vt_cub[1:]) #To make the picture more readable
plt.title("Value Function")
plt.xlabel('Wt')
plt.ylabel('Vt')
plt.show()
```



In [42]:
```
plt.plot(W_vec, W_tadd1)
plt.title("Policy Function")
plt.xlabel('Wt')
plt.ylabel('Wt+1')
plt.show()
```



Exercise 18:

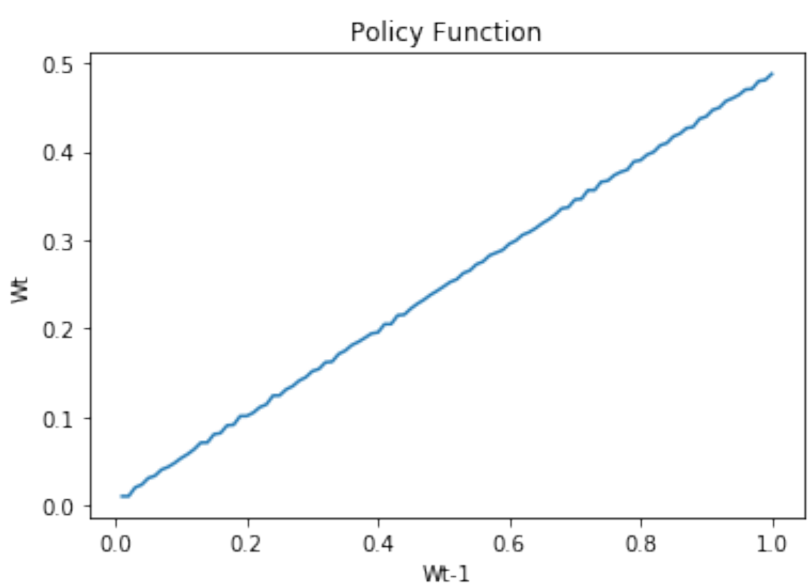
In [43]:
```
sigma_t_cube = np.sum((vt_cube) ** 2) #Vt+1 is np.zeros(N,M)
sigma_t_cube
```

Out[43]: 5.6700000580251445e+20

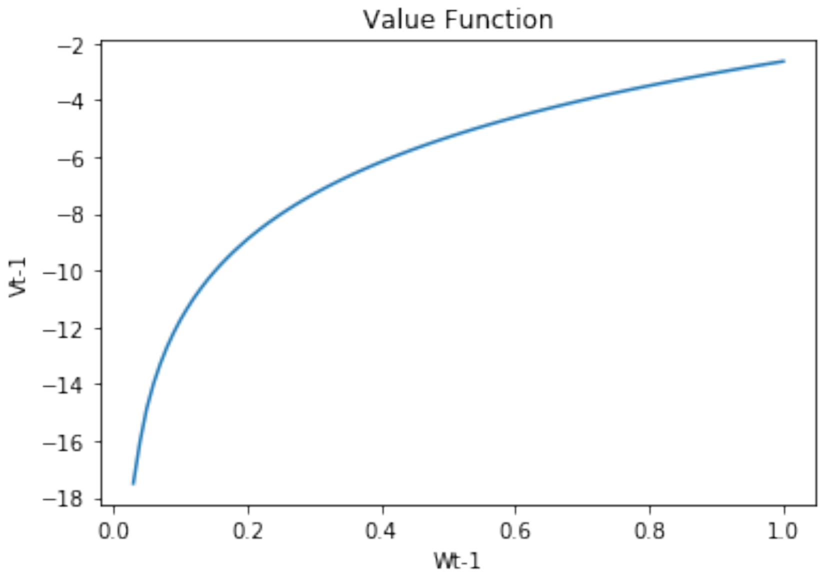
Exercise 19:

In [44]:
```
expected_vt = (vt_cube @ Pr.reshape(-1, 1)).reshape(1, 1, N)
expected_v_t = np.tile(expected_vt, (N, M ,1))
expected_v_t[c_neg] = -1e10

v_tminus1_cube = u_cube + beta * expected_v_t
vtminus1_cube = np.max(v_tminus1_cube, axis=2) #V(W, e)
index = np.argmax(v_tminus1_cube, axis= 2)# W' = psi(W, e)
vtminus1_cub = vtminus1_cube @ Pr.reshape(-1, 1) #E(V(W, e)) = V'(W)
```

In [45]:
```
W_t = np.zeros((N, M))
for i in range(M):
    W_t[:, i] = W_vec[index[:, i]]
W_t = W_t @ Pr.reshape(-1, 1) #W' = E(psi(W, e))

plt.plot(W_vec, W_t)
plt.title("Policy Function")
plt.xlabel('Wt-1')
plt.ylabel('Wt')
plt.show()
```



In [48]:
```
plt.plot(W_vec[2:], vtminus1_cub[2:]) #To make the picture more readab
le
plt.title("Value Function")
plt.xlabel('Wt-1')
plt.ylabel('Vt-1')
plt.show()
```



In [49]:
```
sigma_tminus1 = ((vtminus1_cube - vt_cube) ** 2).sum()
sigma_tminus1
```

Out[49]: 4.592700047000368e+20

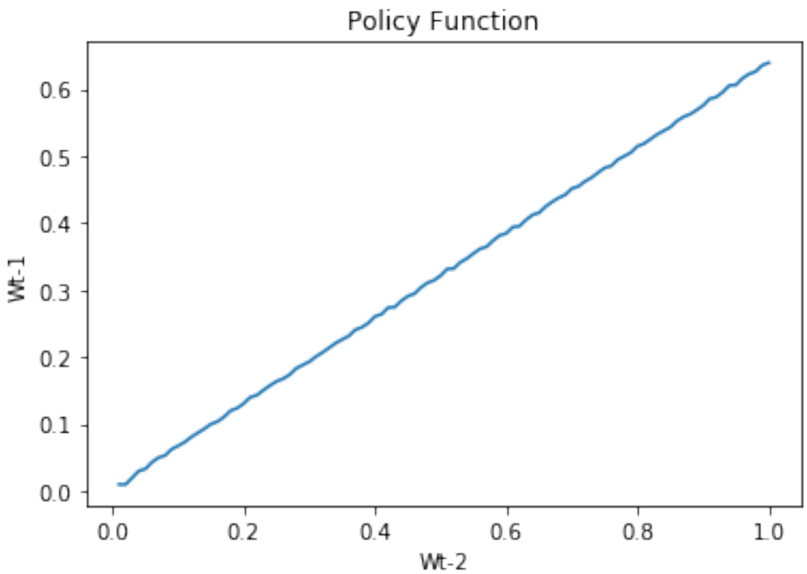
As we can see,

$$\sigma_{T-1} < \sigma_T$$

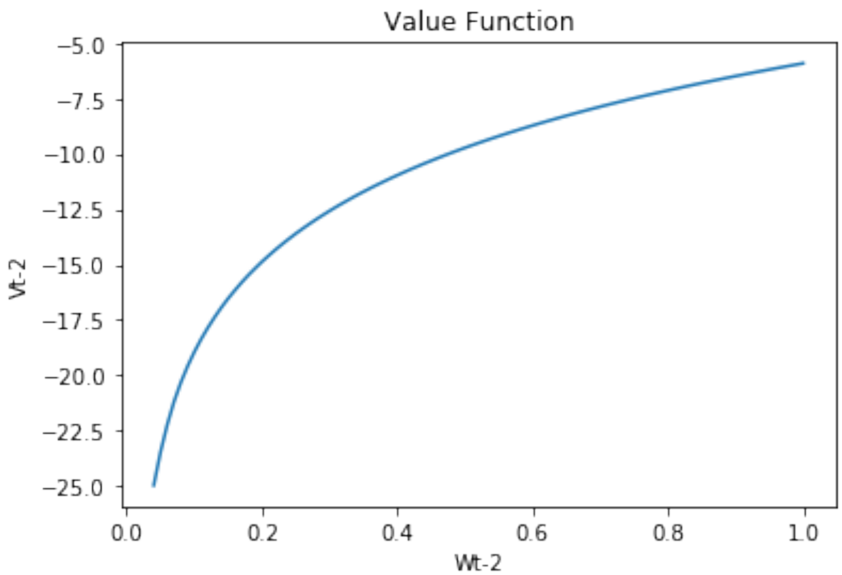
Exercise 20:

In [50]:
```
expected_vtminus1 = (vtminus1_cube @ Pr.reshape(-1, 1)).reshape(1, 1,
N)
expected_v_tminus1 = np.tile(expected_vtminus1, (N, M ,1))
expected_v_tminus1[c_neg] = -1e10

v_tminus2_cube = u_cube + beta * expected_v_tminus1
vtminus2_cube = np.max(v_tminus2_cube, axis=2) #V(W, e)
index = np.argmax(v_tminus2_cube, axis= 2)# W' = psi(W, e)
vtminus2_cub = vtminus2_cube @ Pr.reshape(-1, 1) #E(V(W, e)) = V'(W)
```

In [51]:
```
W_tminus1 = np.zeros((N, M))
for i in range(M):
    W_tminus1[:, i] = W_vec[index[:, i]]
W_tminus1 = W_tminus1 @ Pr.reshape(-1, 1) #W' = E(psi(W, e))

plt.plot(W_vec, W_tminus1)
plt.title("Policy Function")
plt.xlabel('Wt-2')
plt.ylabel('Wt-1')
plt.show()
```



In [53]:
```
plt.plot(W_vec[3:], vtminus2_cub[3:])
plt.title("Value Function")
plt.xlabel('Wt-2')
plt.ylabel('Vt-2')
plt.show()
```



In [54]:
```
sigma_tminus2 = ((vtminus2_cube - vtminus1_cube) ** 2).sum()
sigma_tminus2
```

Out[54]: 3.7200870380702984e+20

Obviously,

$$\sigma_{T-2} < \sigma_{T-1} < \sigma_T$$

Exercise 21:

In [55]:
```
i = 0
sigma = 1
v0_cube = np.zeros((N, M))
while sigma >= 1e-9 and i < 500:
    v1_cube = v0_cube
    expected_v1 = (v1_cube @ Pr.reshape(-1, 1)).reshape(1, 1, N)
    expected_v_1 = np.tile(expected_v1, (N, M ,1))
    expected_v_1[c_neg] = -1e10

    v_0_cube = u_cube + beta * expected_v_1
    v0_cube = np.max(v_0_cube, axis=2) #V(W, e)
    index = np.argmax(v_0_cube, axis= 2)# W' = psi(W, e)
    v0_cub = v0_cube @ Pr.reshape(-1, 1) #E(V(W, e)) = V'(W)
    sigma = ((v0_cube - v1_cube) ** 2).sum()
    i += 1
    print("iteration =", i, "sigma =", sigma)
print ("This covergency takes", i ,"iterations.")
```

```
iteration = 1 sigma = 5.6700000580251445e+20
iteration = 2 sigma = 4.592700047000368e+20
iteration = 3 sigma = 3.7200870380702984e+20
iteration = 4 sigma = 3.013270500836941e+20
iteration = 5 sigma = 2.4407491056779225e+20
iteration = 6 sigma = 1.977006775599117e+20
iteration = 7 sigma = 1.6013754882352847e+20
iteration = 8 sigma = 1.29711414547058e+20
iteration = 9 sigma = 1.0506624578311702e+20
iteration = 10 sigma = 8.51036590843248e+19
iteration = 11 sigma = 6.8933963858303115e+19
iteration = 12 sigma = 5.583651072522553e+19
iteration = 13 sigma = 4.5227573687432675e+19
iteration = 14 sigma = 3.6634334686820475e+19
iteration = 15 sigma = 2.9673811096324588e+19
iteration = 16 sigma = 2.4035786988022915e+19
iteration = 17 sigma = 1.9468987460298564e+19
iteration = 18 sigma = 1.576987984284184e+19
iteration = 19 sigma = 1.2773602672701886e+19
iteration = 20 sigma = 1.034661816488853e+19
iteration = 21 sigma = 8.380760713559713e+18
iteration = 22 sigma = 6.788416177983365e+18
iteration = 23 sigma = 5.498617104166525e+18
iteration = 24 sigma = 4.4538798543748864e+18
iteration = 25 sigma = 3.6076426820436567e+18
iteration = 26 sigma = 2.922190572455362e+18
iteration = 27 sigma = 2.366974363688844e+18
iteration = 28 sigma = 1.9172492345879634e+18
iteration = 29 sigma = 1.552971880016250 4e+18
iteration = 30 sigma = 1.2579072228131628e+18
```

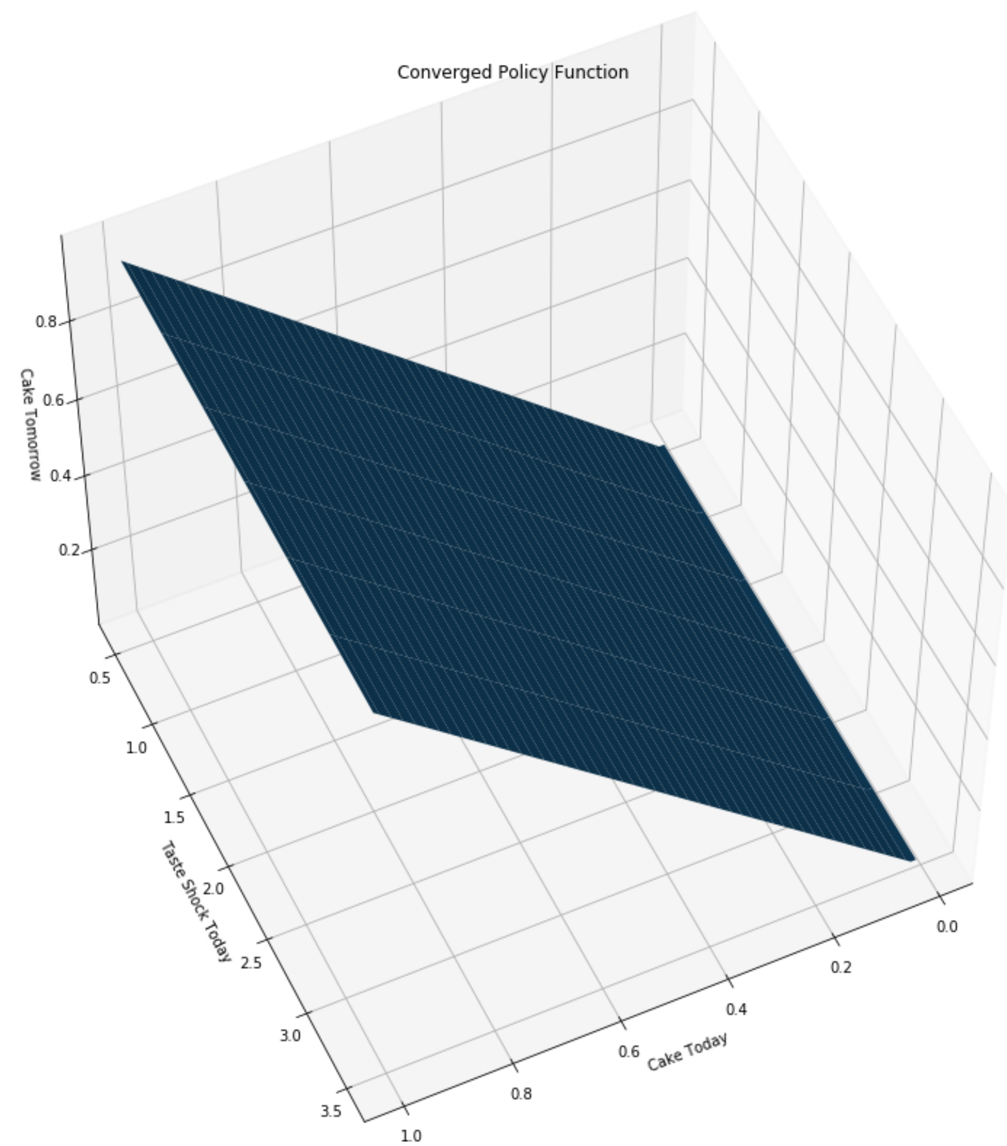
```
iteration = 31 sigma = 1.0189048504786616e+18
iteration = 32 sigma = 8.25312928887716e+17
iteration = 33 sigma = 6.685034723990497e+17
iteration = 34 sigma = 5.4148781264323046e+17
iteration = 35 sigma = 4.386051282410167e+17
iteration = 36 sigma = 3.5527015387522355e+17
iteration = 37 sigma = 2.8776882463893117e+17
iteration = 38 sigma = 2.3309274795753424e+17
iteration = 39 sigma = 1.8880512584560278e+17
iteration = 40 sigma = 1.5293215193493827e+17
iteration = 41 sigma = 1.2387504306730006e+17
iteration = 42 sigma = 1.0033878488451306e+17
iteration = 43 sigma = 8.12744157564556e+16
iteration = 44 sigma = 6.583227676272908e+16
iteration = 45 sigma = 5.332414417781057e+16
iteration = 46 sigma = 4.3192556784026584e+16
iteration = 47 sigma = 3.498597099506155e+16
iteration = 48 sigma = 2.833863650599989e+16
iteration = 49 sigma = 2.295429556985994e+16
iteration = 50 sigma = 1.8592979411586576e+16
iteration = 51 sigma = 1.5060313323385154e+16
iteration = 52 sigma = 1.2198853791942e+16
iteration = 53 sigma = 9881071571473046.0
iteration = 54 sigma = 8003667972893192.0
iteration = 55 sigma = 6482971058043506.0
iteration = 56 sigma = 5251206557015263.0
iteration = 57 sigma = 4253477311182384.5
iteration = 58 sigma = 3445316622057755.5
iteration = 59 sigma = 2790706463866804.5
iteration = 60 sigma = 2260472235732134.0
iteration = 61 sigma = 1830982510943050.8
iteration = 62 sigma = 1483095833863893.0
iteration = 63 sigma = 1201307625429775.0
iteration = 64 sigma = 973059176598139.4
iteration = 65 sigma = 788177933044514.2
iteration = 66 sigma = 638424125766077.2
iteration = 67 sigma = 517123541870543.7
iteration = 68 sigma = 418870068915161.06
iteration = 69 sigma = 339284755821301.06
iteration = 70 sigma = 274820652215274.22
iteration = 71 sigma = 222604728294392.16
iteration = 72 sigma = 180309829918477.47
iteration = 73 sigma = 146050962233986.3
iteration = 74 sigma = 118301279409548.27
iteration = 75 sigma = 95824036321753.17
iteration = 76 sigma = 77617469420638.89
iteration = 77 sigma = 62870150230735.98
iteration = 78 sigma = 50924821686914.32
iteration = 79 sigma = 41249105566418.5
iteration = 80 sigma = 33411775508816.523
```

```
iteration = 81 sigma = 27063538162158.547
iteration = 82 sigma = 21921465911365.18
iteration = 83 sigma = 17756387388222.133
iteration = 84 sigma = 14382673784475.812
iteration = 85 sigma = 11649965765440.787
iteration = 86 sigma = 9436472270021.875
iteration = 87 sigma = 7643542538731.971
iteration = 88 sigma = 6191269456386.519
iteration = 89 sigma = 5014928259685.982
iteration = 90 sigma = 4062091890357.76
iteration = 91 sigma = 3290294431200.983
iteration = 92 sigma = 2665138489283.05
iteration = 93 sigma = 2158762176328.333
iteration = 94 sigma = 1748597362833.7273
iteration = 95 sigma = 1416363863901.4849
iteration = 96 sigma = 1147254729764.4727
iteration = 97 sigma = 929276331110.9948
iteration = 98 sigma = 752713828198.5693
iteration = 99 sigma = 609698200835.3107
iteration = 100 sigma = 493855542659.8484
iteration = 101 sigma = 0.0
This covergency takes 101 iterations.
```

Exercise 22:

In [56]:
```
W_prime = np.zeros((N, M))
for i in range(M):
    W_prime[:, i] = W_vec[index[:, i]]
W_prime = W_prime @ Pr.reshape(-1, 1) #W' = E(psi(W, e))
```

In [63]:
```
from mpl_toolkits.mplot3d import Axes3D
X, Y = np.meshgrid(W_vec, E)
figure = plt.figure(figsize=(15, 15))
ax1 = figure.add_subplot(111, projection='3d')
ax1.plot_surface(X.T, Y.T, W_prime)
ax1.set_xlabel('Cake Today')
ax1.set_ylabel('Taste Shock Today')
ax1.set_zlabel('Cake Tomorrow')
ax1.set_title("Converged Policy Function")
ax1.view_init(elev=55, azim=65)
plt.show()
```


Converged Policy Function
Cake Tomorrow
0.8
0.6
0.4
0.2
Taste Shock Today
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Cake Today
0.0
0.2
0.4
0.6
0.8
1.0