Exercise 5.1.

Here is the problem for T = 1:

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

The optimal amount of cake leave for W_2 is 0.

Exercise 5.2.

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

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

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

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

Exercise 5.3.

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

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

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

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

Take derivative of conditions, we get:

$$W_4 = 0$$

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

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

Since W_1 = 1, W_4 = 0 and the discount factor is β = 0.9, solve the equation and we get $W_2 = 0.631$, $W_3 = 0.299$. Calculate the consumption for each period, we get $c_1 = W_1 - W_2 = 0.369$, $c_2 = W_2 - W_3 = 0.332$, $c_3 = W_3 - W_4 = 0.299$.

Exercise 5.4.

The value function V_{T-1} is:

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

The condition that characterizes the optimal choice in period T-1 is:

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

Exercise 5.5.

When $u(x) = ln(x)$ and $V_T(\bar{W}) = u(\bar{W})$, solve the condition that characterizes the optimal choice in period T-1:

$$\psi_{T-1}(\bar{W}) = \frac{\beta}{1+\beta}\bar{W}$$
$$\psi_T(\bar{W}) = 0$$

$\psi_{T-1}(\bar{W})$ is not equal to $\psi_T(\bar{W})$.

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

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

$V_{T-1}(\bar{W})$ is not equal to $V_T(\bar{W})$.

Exercise 5.6.

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

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

Using the envelope theorem:

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

$$W_T = \beta(W_{T-1} - W_T)$$

Here is the analytical solution for $\psi_{T-2}(W_{T-2})$:

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

Here is the analytical solution for $V_{T-2}(W_{T-2})$:

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

Exercise 5.7.

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

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

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

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

Exercise 5.8.

When the horizon is infinite, the Bellman equation for the cake eating problem is:

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

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
```

Exercise 5.9.

In [2]:
```
N = 100
W = np.linspace(1e-2, 1, N)
```

Exercise 5.10.

In [3]:
```
beta = 0.9
def u(c):
    u = np.log(c)
    return u
```

The resulting policy function: $W' = \psi_T(W) = 0$

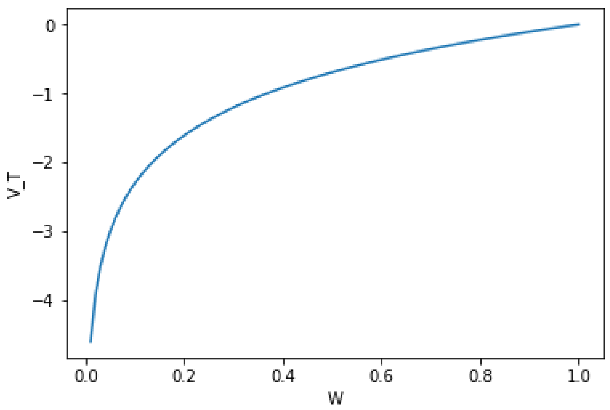
The value function: $V_T(W) = ln(W)$

In [4]:
```
V_T = u(W)
print(V_T)
```

```
[-4.60517019 -3.91202301 -3.5065579  -3.21887582 -2.99573227 -2.81341072
 -2.65926004 -2.52572864 -2.40794561 -2.30258509 -2.20727491 -2.12026354
 -2.04022083 -1.96611286 -1.89711998 -1.83258146 -1.77195684 -1.71479843
 -1.66073121 -1.60943791 -1.56064775 -1.51412773 -1.46967597 -1.42711636
 -1.38629436 -1.34707365 -1.30933332 -1.27296568 -1.23787436 -1.2039728
 -1.17118298 -1.13943428 -1.10866262 -1.07880966 -1.04982212 -1.02165125
 -0.99425227 -0.96758403 -0.94160854 -0.91629073 -0.89159812 -0.86750057
 -0.84397007 -0.82098055 -0.7985077  -0.77652879 -0.75502258 -0.73396918
 -0.71334989 -0.69314718 -0.67334455 -0.65392647 -0.63487827 -0.61618614
 -0.597837   -0.5798185  -0.56211892 -0.54472718 -0.52763274 -0.51082562
 -0.49429632 -0.4780358  -0.46203546 -0.4462871  -0.43078292 -0.41551544
 -0.40047757 -0.38566248 -0.37106368 -0.35667494 -0.34249031 -0.32850407
 -0.31471074 -0.30110509 -0.28768207 -0.27443685 -0.26136476 -0.24846136
 -0.23572233 -0.22314355 -0.21072103 -0.19845094 -0.18632958 -0.17435339
 -0.16251893 -0.15082289 -0.13926207 -0.12783337 -0.11653382 -0.10536052
 -0.09431068 -0.08338161 -0.07257069 -0.0618754  -0.05129329 -0.04082199
 -0.03045921 -0.02020271 -0.01005034  0.        ]
```

In [5]:
```
fig,ax = plt.subplots()
ax.plot(W, V_T)
ax.set_xlabel("W")
ax.set_ylabel("V_T")
```

Out[5]: `Text(0, 0.5, 'V_T')`



Exercise 5.11.

In [6]:
```
def d(V_T, V_Tp1):
    d = ((V_Tp1 - V_T) ** 2).sum()
    return d
diff = d(V_T, np.zeros(N))
diff
```

Out[6]: `178.92611065972804`

Exercise 5.12.

In [7]:
```
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-7
u_mat = u(c_mat)
V_prime = np.tile(V_T.reshape((1,N)), (N,1))
V_prime[~c_pos] = -9e+4
V_Tp1 = (u_mat + beta * V_prime).max(axis = 1)
diff = d(V_T, V_Tp1)
W_index = np.argmax(u_mat + beta * V_prime, axis=1)
W_prime = W[W_index]
W_prime
```

Out[7]:
```
array([0.01, 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, 0.09, 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, 0.18, 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, 0.27, 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, 0.36,
       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, 0.45, 0.46, 0.46, 0.47,
       0.47])
```

In [8]:
```
diff = d(V_T, V_Tp1)
diff
```

Out[8]:
```
6562865744.5285635
```

Exercise 5.13.

In [9]:
```
V_T = V_Tp1
V_prime = np.tile(V_T.reshape((1,N)), (N,1))
V_prime[~c_pos] = -9e+4
V_Tp1 = (u_mat + beta * V_prime).max(axis = 1)
diff = d(V_T, V_Tp1)
W_index = np.argmax(u_mat + beta * V_prime, axis=1)
W_prime = W[W_index]
diff = d(V_T, V_Tp1)
diff
```

Out[9]:
```
5315921432.356884
```

Exercise 5.14.

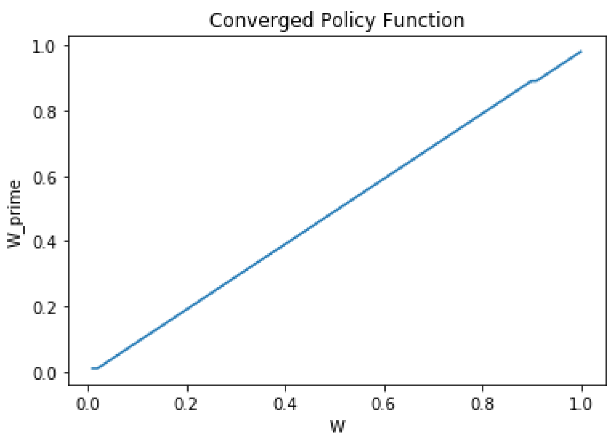
In [10]:
```python
maxiters = 500
toler = 1e-9
diff = 10.0
VF_iter = 0

while diff > toler and VF_iter < maxiters:
    VF_iter += 1
    V_prime = np.tile(V_T.reshape((1,N)), (N,1))
    V_prime[~c_pos] = -9e+4
    V_Tp1 = (u_mat + beta * V_prime).max(axis = 1)
    diff = d(V_T, V_Tp1)
    W_index = np.argmax(u_mat + beta * V_prime, axis=1)
    W_prime = W[W_index]
    print('Number of iteration=', VF_iter, ', distance =', diff)
    V_T = V_Tp1
print('V(W)', V_T)
```

```
Number of iteration= 1 , distance = 5315921432.356884
Number of iteration= 2 , distance = 4305896471.418966
Number of iteration= 3 , distance = 3487776216.5675955
Number of iteration= 4 , distance = 2825098788.019544
Number of iteration= 5 , distance = 2288330056.357764
Number of iteration= 6 , distance = 1853547373.8090138
Number of iteration= 7 , distance = 1501373394.0455368
Number of iteration= 8 , distance = 1216112465.4304204
Number of iteration= 9 , distance = 985051109.6286845
Number of iteration= 10 , distance = 797891408.7166361
Number of iteration= 11 , distance = 646292049.0098146
Number of iteration= 12 , distance = 523496566.09326845
Number of iteration= 13 , distance = 424032223.8060949
Number of iteration= 14 , distance = 343466105.6810028
Number of iteration= 15 , distance = 278207549.28964245
Number of iteration= 16 , distance = 225348118.14783967
Number of iteration= 17 , distance = 182531978.5237014
Number of iteration= 18 , distance = 147850905.07819986
Number of iteration= 19 , distance = 119759235.2952075
```

Exercise 5.15.

In [11]:
```python
fig,ax = plt.subplots()
ax.plot(W, W_prime)
ax.set_xlabel("W")
ax.set_ylabel("W_prime")
ax.set_title("Converged Policy Function")
```

Out[11]: Text(0.5, 1.0, 'Converged Policy Function')



Exercise 5.16.

In [12]:
```
sigma = 0.05
M = 7
mu = 4 * sigma
e_ub = mu+3 * sigma
e_lb = mu-3 * sigma
epsilon = np.linspace(e_lb, e_ub, M)
epsilon
```

Out[12]:
```
array([0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35])
```

In [13]:
```
f = lambda x: norm(loc=mu, scale=sigma).pdf(x)
gamma = f(epsilon)
gamma
```

Out[13]:
```
array([0.08863697, 1.07981933, 4.83941449, 7.97884561, 4.83941449,
       1.07981933, 0.08863697])
```

Exercise 5.17.

In [14]:
```
W_lb = 1e-2
W_ub = 1.0
N = 100
W = np.linspace(W_lb, W_ub, N)

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-7
u_mat = u(c_mat)
eu_cube = np.array([u_mat * e for e in epsilon])
```

In [15]:
```
V_T = np.zeros((N,M))

EV_prime = V_T @ gamma.reshape((M,1))
EV_prime_mat = np.tile(EV_prime.reshape((1,N)), (N,1))
EV_prime_mat[~c_pos] = -9e+4
EV_prime_cube = np.array([EV_prime_mat for e in range(M)])
```

In [16]:
```
V_Tp1_cube = eu_cube + beta * EV_prime_cube
V_Tp1 = np.zeros((N,M))
W_Tp1 = np.zeros((N,M))
for i in range(N):
    sheet = V_Tp1_cube[:, i, :]
    V_Tp1[i] = sheet.max(axis=1)
    W_index = np.argmax(sheet, axis=1)
    W_Tp1[i] = W[W_index]
```

Exercise 5.18.

In [17]:
```
def d(V_T, V_Tp1):
    d = np.sum((V_T-V_Tp1) ** 2)
    return d
d(V_T, V_Tp1)
```

Out[17]:
```
45930655737.64533
```

Exercise 5.19.

In [18]:
```
V_T = V_Tp1

EV_prime = V_T @ gamma.reshape((M,1))
EV_prime_mat = np.tile(EV_prime.reshape((1,N)), (N,1))
EV_prime_mat[~c_pos] = -9e+4
EV_prime_cube = np.array([EV_prime_mat for e in range(M)])
V_Tp1_cube = eu_cube + beta * EV_prime_cube
V_Tp1 = np.zeros((N,M))
W_Tp1 = np.zeros((N,M))
for i in range(N):
    sheet = V_Tp1_cube[:, i, :]
    V_Tp1[i] = sheet.max(axis=1)
    W_index = np.argmax(sheet, axis=1)
    W_Tp1[i] = W[W_index]
print(d(V_T, V_Tp1))
```

```
45929630360.5779
```

Exercise 5.20.

In [19]:
```
V_T = V_Tp1

EV_prime = V_T @ gamma.reshape((M,1))
EV_prime_mat = np.tile(EV_prime.reshape((1,N)), (N,1))
EV_prime_mat[~c_pos] = -9e+4
EV_prime_cube = np.array([EV_prime_mat for e in range(M)])
V_Tp1_cube = eu_cube + beta * EV_prime_cube
V_Tp1 = np.zeros((N,M))
W_Tp1 = np.zeros((N,M))
for i in range(N):
    sheet = V_Tp1_cube[:,i,:]
    V_Tp1[i] = sheet.max(axis=1)
    W_index = np.argmax(sheet, axis=1)
    W_Tp1[i] = W[W_index]
print(d(V_T, V_Tp1))
```

```
45917033863.3907
```

Exercise 5.21.

In [20]:
```
maxiters = 500
toler = 1e-9
diff = 10.0
VF_iter = 0
V_T = np.zeros((N,M))

while diff > toler and VF_iter < maxiters:
    VF_iter += 1
    EV_prime = V_T @ gamma.reshape((M,1))
    EV_prime_mat = np.tile(EV_prime, (1,N))
    EV_prime_mat[~c_pos] = -9e+4
    EV_prime_cube = np.array([EV_prime_mat for e in range(M)])
    V_Tp1_cube = eu_cube + beta * EV_prime_cube
    V_Tp1 = np.zeros((N,M))
    W_Tp1 = np.zeros((N,M))
    for i in range(N):
        sheet = V_Tp1_cube[:,i,:]
        V_Tp1[i] = sheet.max(axis=1)
        W_index = np.argmax(sheet, axis=1)
        W_Tp1[i] = W[W_index]
    diff = d(V_T, V_Tp1)
    print('Number of iteration = ', VF_iter, ', distance = ', diff)
    V_T = V_Tp1
W_Tp1
```

```
Number of iteration =  1 , distance =  45930655737.64533
Number of iteration =  2 , distance =  16223.391772312192
Number of iteration =  3 , distance =  5253534.138986075
Number of iteration =  4 , distance =  1701223846.2116983
Number of iteration =  5 , distance =  518813656513.1142
Number of iteration =  6 , distance =  2186621133313.0828
Number of iteration =  7 , distance =  263770458241.2872
Number of iteration =  8 , distance =  533625542.1073715
Number of iteration =  9 , distance =  0.0
```

Exercise 5.22.

In [21]:

```
X, Y = np.meshgrid(W, epsilon)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X.T, Y.T, W_Tp1)
ax.set_xlabel('W')
ax.set_ylabel('epsilon')
ax.set_zlabel('Optimal Policy')
ax.set_title("Converged Policy Function")
ax.view_init(elev=30,azim=30)
plt.show()
```