Exercise 5.1
Condition characterizing the optimal amount of cake to be eaten in period 1:

$$\text{maximize } \sum_{t=1} B^0 u(c_1) \text{ such that } w_2 = w_1 - c_1 \text{ over } c_1 \in [0, w_1]$$

where c_1, is the consumption in period 1, $w_1]$ is the size of the cake in period 1 (given), and $w_2]$ is the size of the cake left for period 2.

The problem is equivalently written in terms of finding optimal value of w_2 (cake to be saved in period 2):

$$\text{maximize } \sum_{t=1} B^0 u(w_1 - w_2) \text{ over } w_2 = [0, w_1]$$

Exercise 5.2
In period $t = 2$, condition characterizing the optimal amount of cake to leave for the next period w_3:

$$\text{maximize } B^0 u(w_1 - w_2) + B^1 u(w_2 - w_3) \text{ over } w_3 = [0, w_2]$$

where w_1 is the size of the cake available in period 1, w_2 is the size of cake available in period 2.

Condition characterizing the optimal amount of cake to be left for the next period w_2 in period 1:

$$\text{maximize } B^0 u(u(w_1 - w_2)) \text{ over } w_2 \in [0, w_1]$$

where $w_1]$ is the size of the cake available in period 1.

Exercise 5.3 (a)
Condition characterizing the optimal w_2, w_3, w_4:

$$V_T(w_3) = maxu(w_3 - w_4) \text{ over } w_4$$
$$V_T(w_2) = maxu(w_2 - w_3) + Bu(w_3 - w_4) \text{ over } w_3, w_4$$
$$V_T(w_1) = maxu(w_1 - w_2) + Bu(w_2 - w_3) + B^2 u(w_3 - w_4) \text{ over } w_2, w_3, w_4$$

Exercise 5.3 (b)
B=0.9, $u(c_t) = lu(c_t), w_1 = 1$
$T = 3$ and last period implies $w_4 = 0$. Hence,

$V_T(w_3) = u(w_3)$
Now, $V_T(w_2) = maxu(w_2 - w_3) + Bu(w_3)$
Taking derivative with respect to w_3:

$u'(w_2 - w_3) = Bu'(w3)$
$\frac{1}{w_2 - w_3} = \frac{0.9}{w3}$

Therefore, $1.9w_3 = 0.9w_2$

Now, $V_T(w_1) = maxu(w_1 - w_2)overw_2 + Bu(w_2 - 9/19(w_2)) + B^2u(9/19(w_2))$

Taking derivative with respect to w_1
$\frac{1}{w_1 - w_2} = \frac{19B}{10w_2} + \frac{19B^2}{9w_2}$
Solving for w_2, using $B = 0.9$ and $w_1 = 1$, we get:

$w_2 = 0.62$
$w_3 = 0.3$
$w_4 = 0$

In [ ]:

```

```

In [ ]:

```

```

Exercise 5.8

Bellman equation for cake eating problem with a general utility function u(c) and infinite horizon:

V(W) = max{u(W - W') + B*V(W') } over W' belonging to [0, W]

B denotes beta

Note: W represents today time period and W' represents tomorrow's time period

Exercise 5.9 - 5.15

Exercise 5.9

In [345]:

```
import numpy as np
w_lb = 1e-2
w_ub = 1.0
N = 100
w_vec = np.linspace(w_lb, w_ub, N)
```

Exercise 5.10 and 5.11

In [346]:

```
beta = 0.9

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

# initial guess for value function
#v_init = utility(w_vec)
v_init = np.zeros(100)

def compute_v_t(v_initial):
    policy_func = np.zeros(N)
    dist_vec = np.zeros(N)
    #create utility matrix
    c_mat = (np.tile(w_vec.reshape((N,1)),(1,N)) - np.tile(w_vec.reshape((1,N)),(N,1)))
    c_pos = c_mat > 0
    c_mat[~c_pos] = 1e-7
    u_mat = utility(c_mat)
    vf_iter = 0
    for i in range(1):
        vf_iter += 1
        #one contraction mapping
        v_prime = np.tile(v_initial.reshape((1,N)),(N,1))
        v_prime[~c_pos] = -9e+4
        x = u_mat + beta*v_prime
        v_new = x.max(axis = 1)
        policy_index = np.argmax(x, axis = 1)
        policy_func = w_vec[policy_index]
        dist_vec[i] = ((v_new - v_initial)**2).sum()
        v_initial = v_new
    return v_initial, policy_func, dist_vec
```

In [347]:

```
v_time_t, policy_func_t, dist_func_t = compute_v_t(v_init)
```

Exercise 5.12

In [348]:

```
v_time_t_1, policy_func_t_1, dist_func_t_1 = compute_v_t(v_time_t)
```

Exercise 5.13

In [349]:

```
v_time_t_2, policy_func_t_2, dist_func_t_2 = compute_v_t(v_time_t_1)
print(dist_func_t[0], "delta t")
print(dist_func_t_1[0], "delta t_1")
print(dist_func_t_2[0], "delta t_2")
```

```
6563611570.214573 delta t
5316525743.271798 delta t_1
4306386030.006323 delta t_2
```

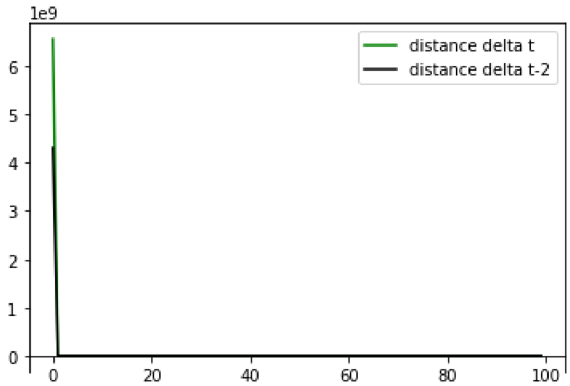
The distance linearly decreases as time changes from T to T-1 to T-2.

In [238]:

```
from matplotlib import pyplot as plt
ax = plt.gca()
ax.spines["bottom"].set_position("zero")
iterr = range(0, 100)
plt.plot(iterr, dist_func_t, color='green', label='distance delta t')
plt.plot(iterr, dist_func_t_2, color='black', label='distance delta t-2')
ax.legend()
plt.show
```

Out[238]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```

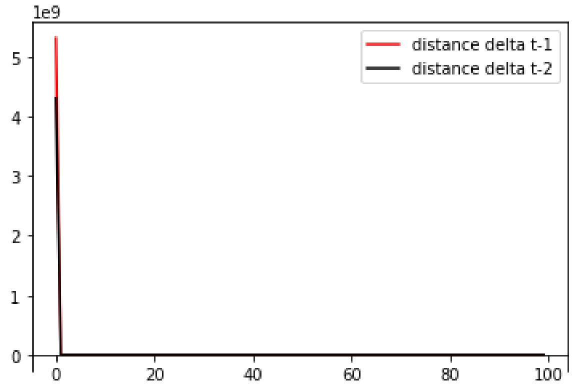


In [239]:

```
from matplotlib import pyplot as plt
ax = plt.gca()
ax.spines["bottom"].set_position("zero")
iterr = range(0, 100)
plt.plot(iterr, dist_func_t_1, color='red', label='distance delta t-1')
plt.plot(iterr, dist_func_t_2, color='black', label='distance delta t-2')
ax.legend()
plt.show
```

Out[239]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



Exercise 5.14

In [340]:

```
import numpy as np
w_lb = 1e-2
w_ub = 1.0
N = 100
w_vec = np.linspace(w_lb, w_ub, N)
beta = 0.9

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

# initial guess for value function
v_init = utility(w_vec)

def compute_v_t_loop(v_initial):
    policy_func = np.zeros(N)
    dist_vec = []
    #create utility matrix
    c_mat = (np.tile(w_vec.reshape((N,1)),(1,N)) - np.tile(w_vec.reshape((1,N)),(N,1)))
    c_pos = c_mat > 0
    c_mat[~c_pos] = 1e-7
    u_mat = utility(c_mat)
    maxiters = 500
    toler = 1e-9
    dist = 10.0
    vf_iter = 0
    while dist > toler and vf_iter < maxiters:
        vf_iter += 1
        #one contraction mapping
        v_prime = np.tile(v_initial.reshape((1,N)),(N,1))
        v_prime[~c_pos] = -9e+4
        x = u_mat + beta*v_prime
        v_new = x.max(axis = 1)
        policy_index = np.argmax(x, axis = 1)
        policy_func = w_vec[policy_index]
        dist_vec.append(((v_new - v_initial)**2).sum())
        #print(dist, vf_iter)
        #print("iter=" , vf_iter, ", distance =", dist)
        v_initial = v_new
    return v_initial, policy_func, dist_vec
```

In [342]:

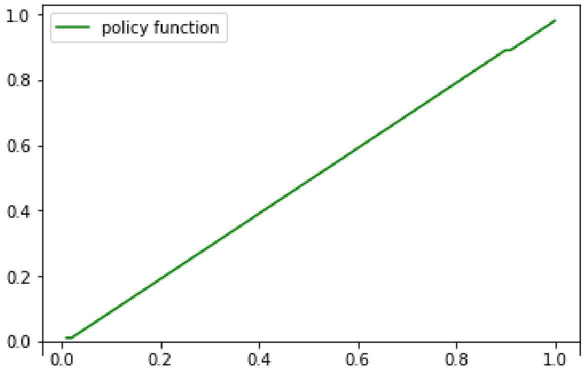
```
v_init = utility(w_vec)
v_t_loop, policy_t_loop, dist_t_loop = compute_v_t_loop(v_init)
print("the number of iterations are", vf_iter)
```

```
the number of iterations are 100
```

Exercise 5.15

In [339]:

```python
from matplotlib import pyplot as plt
ax = plt.gca()
ax.spines["bottom"].set_position("zero")
plt.plot(w_vec, policy_t_loop, color='green', label = 'policy function')
ax.legend()
plt.show()
```



Exercise 5.16 - 5.22

Exercise 5.16

In [351]:

```python
M = 7
sigma = 0.05
mu = 4*sigma
e_ub = mu + 3*sigma
e_lb = mu - 3*sigma
e = np.linspace(e_lb, e_ub, M)
print(e, "support of epsilon")
```

```
[0.05 0.1  0.15 0.2  0.25 0.3  0.35] support of epsilon
```

In [350]:

```python
from scipy import stats as st
mu = 0
sigma = 0.05
delta = 0.05 #difference between two consecutive 'e' values
gamma = np.zeros(M) #probability values
for i in range(M):
    a = e[i] - 0.5*delta
    if i != 6:
        b = e[i + 1] - 0.5*delta
    else:
        b = e[i] - 0.5*delta
    gamma[i] = st.norm.cdf(a, loc=mu, scale=sigma) - st.norm.cdf(b, loc=mu, scale=sigma)
print(gamma, "probability distribution values")
```

```
[-2.41730337e-01 -6.05975359e-02 -5.97703625e-03 -2.29231406e-04
 -3.37868356e-06 -1.89494025e-08  0.00000000e+00] probability distribution values
```

Exercise 5.17 and 5.18

In [282]:

```python
import numpy as np
w_lb = 1e-2
w_ub = 1.0
N = 100
w_vec = np.linspace(w_lb, w_ub, N)


beta = 0.9
```

```
def utility(c):
    util = np.log(c)
    return util

# initial guess for value function
#v_init = utility(w_vec)
v_init = np.zeros(shape = (N,M))
policy_func = np.zeros(shape = (N,M))
def compute_v_t_e(v_initial):
    dist_vec = np.zeros(N)
    #create utility matrix
    c_mat = (np.tile(w_vec.reshape((N,1)),(1,N)) - np.tile(w_vec.reshape((1,N)),(N,1)))
    c_pos = c_mat > 0
    c_mat[~c_pos] = 1e-7
    u_mat = utility(c_mat)
    u_mat_e = np.repeat(u_mat, M).reshape((N,N,M))*e
    vf_iter = 0
    for i in range(1):
        vf_iter += 1
        #one contraction mapping
        expected_v = (v_initial*gamma).sum(axis=1)
        x = np.swapaxes(np.swapaxes(u_mat_e, 1, 2) + beta*expected_v, 1, 2)
        x[np.triu_indices(N, k=1)] = -9e+4
        v_new = np.max(x, axis=1)
        policy_index = np.argmax(x, axis=1)
        policy_func = w_vec[policy_index]
        #v_prime = np.tile(v_initial.reshape((1,N)),(N,1))
        #v_prime[~c_pos] = -9e+4
        #x = u_mat + beta*v_prime
        #v_new = x.max(axis = 1)
        #policy_index = np.argmax(x, axis = 1)
        #policy_func = w_vec[policy_index]

        dist_vec[i] = ((v_new - v_initial)**2).sum()

        #print(dist, vf_iter)
        #print("iter=" , vf_iter, ", distance =", dist)
        v_initial = v_new
    return v_initial, policy_func, dist_vec
```

In [283]:

```
v_time_t_e, policy_t_e, dist_t_e = compute_v_t_e(v_init)
```

Exercise 5.19

In [284]:

```
v_time_t_1_e, policy_t_1_e, dist_t_1_e = compute_v_t_e(v_time_t_e)
```

Exercise 5.20

In [285]:

```
v_time_t_2_e, policy_t_2_e, dist_t_2_e = compute_v_t_e(v_time_t_1_e)
```

In [352]:

```
print("distance in case of V_t is = ", dist_t_e[0])
print("distance in case of V_t_1 is = ", dist_t_1_e[0])
print("distance in case of V_t_2 is = ", dist_t_2_e[0])
"The distance for T-2 sharply drops in comparison to T and T-1 distances"
```

```
distance in case of V_t is =  153.5516913256094
distance in case of V_t_1 is =  53.67853426975263
distance in case of V_t_2 is =  4.139059683986369
```

Out[352]:

```
'The distance for T-2 sharply drops in comparison to T and T-1 distances'
```

"The distance for T-2 sharply drops in comparison to T and T-1 distances"

Exercise 5.21

In [290]:

```
import numpy as np
w_lb = 1e-2
w_ub = 1.0
N = 100
w_vec = np.linspace(w_lb, w_ub, N)

beta = 0.9

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

# initial guess for value function
#v_init = utility(w_vec)
v_init = np.zeros(shape = (N,M))
policy_func = np.zeros(shape = (N,M))
def compute_v_t_e_loop(v_initial):
    dist_vec = []
    #create utility matrix
    c_mat = (np.tile(w_vec.reshape((N,1)),(1,N)) - np.tile(w_vec.reshape((1,N)),(N,1)))
    c_pos = c_mat > 0
    c_mat[~c_pos] = 1e-7
    u_mat = utility(c_mat)
    u_mat_e = np.repeat(u_mat, M).reshape((N,N,M))*e
    maxiters = 500
    toler = 1e-9
    dist = 10.0
    vf_iter = 0
    while dist > toler and vf_iter < maxiters:
        vf_iter += 1
        #one contraction mapping
        expected_v = (v_initial*gamma).sum(axis=1)
        x = np.swapaxes(np.swapaxes(u_mat_e, 1, 2) + beta*expected_v, 1, 2)
        x[np.triu_indices(N, k=1)] = -9e+4
        v_new = np.max(x, axis=1)
        policy_index = np.argmax(x, axis=1)
        policy_func = w_vec[policy_index]
        dist_vec.append(((v_new - v_initial)**2).sum())
        v_initial = v_new
    return v_initial, policy_func, dist_vec
```

In [323]:

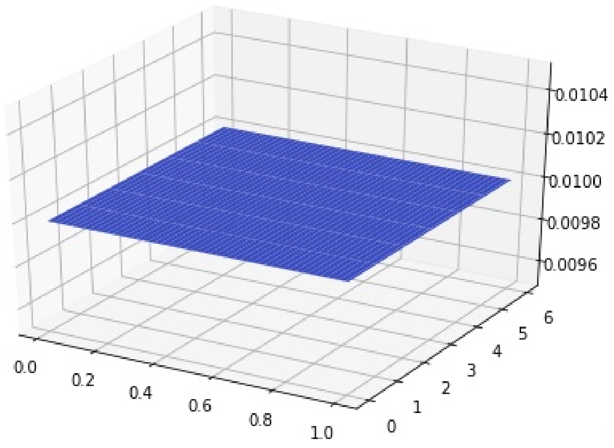
```
v_init = np.zeros(shape = (N,M))
v_t_e_loop, policy_t_e_loop, dist_t_e_loop = compute_v_t_e_loop(v_init)
```

Exercise 5.22

In [324]:

```
from matplotlib import pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
x = np.arange(0,N)
y = np.arange(0,M)
X,Y = np.meshgrid(x,y)
print(w_vec[X].shape)
print(Y.shape)
v_t_e_loop = policy_t_e_loop.reshape(7,100)
print(policy_t_e_loop.shape)
fig1 = plt.figure()
ax1 = Axes3D(fig1)
ax1.plot_surface(w_vec[X], Y, v_t_e_loop, cmap=cm.coolwarm)
plt.show()
```

```
(7, 100)
(7, 100)
(100, 7)
```



Reference: http://www.acme.byu.edu/wp-content/uploads/2014/09/Vol2Lab22Cake1.pdf