MACS30150 Assignment 3

Dr. Richard Evans
Submitted by: Junho Choi
Due January 30, 2019 (11:30 AM)

Exercise 5.1

If the individual lives only for one period, he or she will want to consume all of cake to maximize one's utility. That is, it would simply be the case in which $c_1 = W_1$ and one would achieve the utility of $u(W_1)$. Because the individual does not live in the next period (and cannot get the utility in that period), he or she would not leave anything for the future; therefore, $W_2 = 0$.

So in summary, $c_1 = W_1$ and $W_2 = 0$.

Exercise 5.2

Now, once again, because the person will not live in $T = 3$, it would be that he or she will not leave anything to consume in the periods 3 and onward. Therefore, $W_3 = 0$. This would mean that the individual consumes whatever one has saved from period 1 (i.e. $c_2 = W_2$). Since $c_1 = W_1 - W_2$ by definition, the individual's problem becomes

$$\max_{W_2} u(W_1 - W_2) + \beta u(W_2)$$

where W_1 was given from the start. By the first order conditions, it should be that

$$\beta u'(W_2) = u'(W_1 - W_2) \quad \Leftrightarrow \quad \beta = \frac{u'(W_1 - W_2)}{u'(W_2)} \tag{A}$$

and condition (A) will characterize how W_2 is chosen.

So in summary, $W_3 = 0$, and W_2 is chosen such that $\beta = \frac{u'(W_1 - W_2)}{u'(W_2)}$.

Exercise 5.3

Once again, because the person will not be alive in $T = 4$, we have $W_4 = 0$; that is, nothing is saved for period 4. As such, the consumer will consume all of W_3 in period 3. Knowing that $c_t + W_{t+1} = W_t$ for $t \in \{1, 2, 3\}$, the individual's utility maximization problem becomes

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

By the first order conditions, we have

$$\beta^2 u'(W_3) = \beta u'(W_2 - W_3) \quad \Leftrightarrow \quad \beta = \frac{u'(W_2 - W_3)}{u'(W_3)} \tag{B}$$

$$\beta u'(W_2 - W_3) = u'(W_1 - W_2) \quad \Leftrightarrow \quad \beta = \frac{u'(W_1 - W_2)}{u'(W_2 - W_3)} \tag{C}$$

Assuming that $u'(\cdot) > 0$ and $u''(\cdot) < 0$, and with the knowledge that $\beta = 0.9 < 1$, we have

$$u'(W_3) > u'(W_2 - W_3) \quad \Rightarrow \quad c_3 = W_3 < W_2 - W_3 = c_2$$
$$u'(W_2 - W_3) > u'(W_1 - W_2) \quad \Rightarrow \quad c_2 = W_2 - W_3 < W_1 - W_2 = c_1$$

so $c_1 > c_2 > c_3$ and $0.5W_3 < W_2 < 0.5(W_3 + W_1)$.

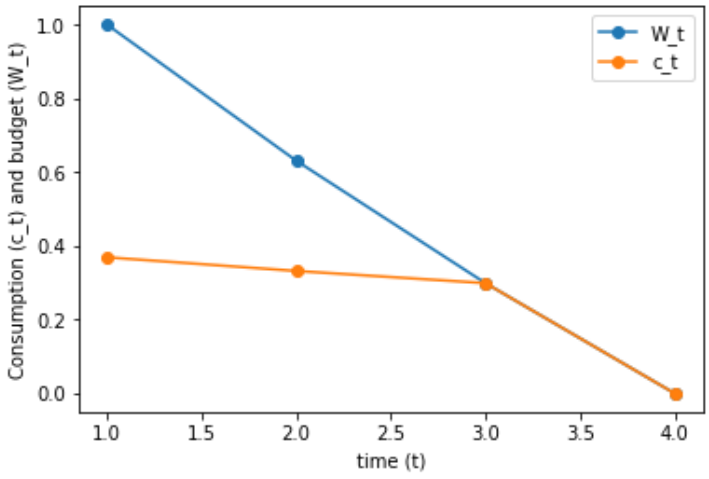
To illustrate this further, let us consider a log utility function $u(\cdot) = \ln(\cdot)$. Then, we have (from (B) and (C) above, and with $W_1 = 1$)

$$0.9(W_2 - W_3) = W_3 \quad \Leftrightarrow \quad W_3 = \frac{9}{19} W_2 \tag{B'}$$

$$0.9(1 - W_2) = W_2 - W_3 \quad \Leftrightarrow \quad W_3 = 1.9W_2 - 0.9 \tag{C'}$$

and combining (B') and (C') gets us $W_2 \approx 0.631$, $W_3 \approx 0.299$, and $W_4 = 0$. The values for consumption in each period, then, will be $c_3 \approx 0.299$, $c_2 \approx 0.332$, and $c_1 \approx 0.369$ which satisfies $c_1 > c_2 > c_3$ as shown above. This result is also summarized in the below figure (Figure 1).

Figure 1: Consumption (c_t) and Budget (W_t) for $t = 1, 2, 3, 4$



Exercise 5.4

The function $V_{T-1}(W_{T-1})$ can be written as follows

$$V_{T-1}(W_{T-1}) \equiv \max_{W_T} \; u(W_{T-1} - W_T) + \beta V_T(W_T) \tag{4}$$

which is expression (5.7) in Evans (2019, p. 47). But we know that $V_T(W_T)$ can be written as $V_T(W_T) = u(W_T - \psi_T(W_T))$ and $\psi_T(W_T)$ was solved to be 0, as the individual will yield no utility from consumption in a period that he or she no longer lives in. So expression (4) above becomes

$$V_{T-1}(W_{T-1}) = \max_{W_T} \; u(W_{T-1} - W_T) + \beta u(W_T) \tag{4-1}$$

and we assume that the value of W_T that solves the above maximization problem in (4-1) – let it be denoted W_T^* – can be written in as a function of W_{T-1} (i.e. $W_T^* = \psi_{T-1}(W_{T-1})$). Therefore, using this, we can rewrite (A') as

$$V_{T-1}(W_{T-1}) = u(W_{T-1} - \psi_{T-1}(W_{T-1})) + \beta u(\psi_{T-1}(W_{T-1})) \tag{4-2}$$

which represents the value function $V_{T-1}(W_{T-1})$ in terms of W_{T-1} and $\psi_{T-1}(W_{T-1})$ as the question asks. Let us now try to solve the maximization problem in (4-1) and try to characterize $\psi_{T-1}(W_{T-1})$. The first order conditions tell us that

$$u'(W_{T-1} - W_T^*) = \beta u'(W_T^*) \quad \Leftrightarrow \quad \beta u'(\psi_{T-1}(W_{T-1})) = u'(W_{T-1} - \psi_{T-1}(W_{T-1}))$$

and this would be the condition that characterizes the optimal choice.

In summary, the condition that characterizes the optimal choice in period $T-1$ is

$$\beta u'(\psi_{T-1}(W_{T-1})) = u'(W_{T-1} - \psi_{T-1}(W_{T-1})) \tag{4-3}$$

and V_{T-1} expressed in terms of $\psi_{T-1}(W_{T-1})$ (and also W_{T-1}) is written above in (4-2).

Exercise 5.5

Let us assume first that $\bar{W} \geq 0$ as cake size should be nonnegative. We know, since the individual ceases to exist in period $T+1$, that $\psi_T(W_T) = 0$ for all $W_T \geq 0$. If $W_{T-1} = 0$, there will be no cake to pass onto period T, and therefore we have $\psi_{T-1}(W_{T-1}) = 0$ when $W_{T-1} = 0$. As such, I will try to show that

$$\psi_{T-1}(\bar{W}) \neq \psi_T(\bar{W}) = 0 \quad \forall \; \bar{W} > 0 \tag{5}$$

and using the fact that the cake size should, again, be nonnegative, we should have $\psi_{T-1}(\bar{W}) \geq 0$. Combined with this, (5) becomes

$$\psi_{T-1}(\bar{W}) > 0 \quad \forall\, \bar{W} > 0 \tag{5-1}$$

Recall that $\psi_{T-1}(\bar{W})$ should be a solution to the below maximization problem:

$$V_{T-1}(\bar{W}) = \max_{W_T} \; u(\bar{W} - W_T) + \beta u(W_T) \tag{5-2}$$

and with the knowledge that $u(\cdot) = \ln(\cdot)$, we have the following first order condition:

$$\begin{aligned} & \frac{\beta}{\psi_{T-1}(\bar{W})} = \frac{1}{\bar{W} - \psi_{T-1}(\bar{W})} \\ \Leftrightarrow \quad & \beta\bar{W} - \beta\psi_{T-1}(\bar{W}) = \psi_{T-1}(\bar{W}) \end{aligned}$$

which is solved to be

$$\psi_{T-1}(\bar{W}) = \frac{\beta\bar{W}}{1+\beta} \tag{5-3}$$

but for $\beta \in (0, 1]$ and $\bar{W} > 0$, this is always greater than 0. So we have proven the expression in (5).

Now that we know the values of $\psi_{T-1}(\bar{W})$ and $\psi_T(\bar{W})$, let us plug these into the equations for $V_{T-1}(\bar{W})$ and $V_T(\bar{W})$. We then get, for $V_T(\bar{W})$,

$$\begin{aligned} V_T(\bar{W}) &= u(\bar{W} - \psi_T(\bar{W})) = u(\bar{W}) \\ &= \ln \bar{W} \end{aligned}$$

and for $V_{T-1}(\bar{W})$,

$$\begin{aligned} V_{T-1}(\bar{W}) &= u(\bar{W} - \psi_{T-1}(\bar{W})) + \beta u(\psi_{T-1}(\bar{W}) - \psi_T(\psi_{T-1}(\bar{W}))) \\ &= u(\bar{W} - \psi_{T-1}(\bar{W})) + \beta u(\psi_{T-1}(\bar{W})) \\ &= \ln\left(\bar{W} - \frac{\beta\bar{W}}{1+\beta}\right) + \beta \ln \frac{\beta\bar{W}}{1+\beta} \\ &= \ln\left(\frac{\bar{W}}{1+\beta}\right) + \beta \ln \frac{\beta\bar{W}}{1+\beta} \\ &= (1+\beta)(\ln \bar{W} - \ln(1+\beta)) + \beta \ln \beta \end{aligned}$$

but notice that, then,

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

which is nonzero for all $\bar{W} > 0$ and $\beta \in (0, 1]$. Notice that if we do consider $\bar{W} = 0$, we will have both $V_{T-1}(\bar{W})$ and $V_T(\bar{W})$ go to $-\infty$.

Therefore, we have shown that for any $\bar{W} > 0$,

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

Exercise 5.6

We can first right the value function for $T-2$ (denoted $V_{T-2}(\cdot)$, given W_{T-2}, as follows:

$$V_{T-2}(W_{T-2}) = \max_{W_{T-1}, W_T, W_{T+1}} u(W_{T-2} - W_{T-1}) + \beta u(W_{T-1} - W_T) + \beta^2 u(W_T - W_{T+1}) \quad (6)$$

but using the functions ψ_{T-2}, ψ_{T-1}, and ψ_T, we can write (6) as follows:

$$\begin{aligned} V_{T-2}(W_{T-2}) = \max_{W_{T-1}} \; & u(W_{T-2} - W_{T-1}) + \beta u(W_{T-1} - \psi_{T-1}(W_{T-1})) \\ & + \beta^2 u(\psi_{T-1}(W_{T-1}) - \psi_T(\psi_{T-1}(W_{T-1}))) \end{aligned} \quad (6\text{-}1)$$

We know that $\psi_T(\psi_{T-1}(W_{T-1})) = 0$ from the previous exercises. Then we can rewrite (6-1) as:

$$\begin{aligned} V_{T-2}(W_{T-2}) = \max_{W_{T-1}} \; & u(W_{T-2} - W_{T-1}) + \beta u(W_{T-1} - \psi_{T-1}(W_{T-1})) \\ & + \beta^2 u(\psi_{T-1}(W_{T-1})) \end{aligned} \quad (6\text{-}2)$$

and noting $\frac{\partial u}{\partial W_{T-1}}$ as $u'(\cdot)$, the first order condition becomes

$$\begin{aligned} [W_{T-1}]: \quad u'(W_{T-2} - W^*_{T-1}) = & \beta u'(W^*_{T-1} - \psi_{T-1}(W^*_{T-1})) \\ & - \beta \psi'_{T-1}(W^*_{T-1}) u'(W^*_{T-1} - \psi_{T-1}(W^*_{T-1})) \\ & + \beta^2 \psi'_{T-1}(W^*_{T-1}) u'(\psi_{T-1}(W^*_{T-1})) \end{aligned} \quad (6\text{-}3)$$

Notice the last two terms on the RHS of (6-3). We can organize them as follows:

$$\beta \psi'_{T-1}(W^*_{T-1})(\beta u'(\psi_{T-1}(W^*_{T-1})) - u'(W^*_{T-1} - \psi_{T-1}(W^*_{T-1}))) \quad (6\text{-}4)$$

but we know that $\beta \psi'_{T-1}(W^*_{T-1}) \neq 0$. Also, from (4-3) in Exercise 5.4, we can see that

$$\beta u'(\psi_{T-1}(W^*_{T-1})) - u'(W^*_{T-1} - \psi_{T-1}(W^*_{T-1})) = 0$$

and therefore the entire expression in (6-4) becomes 0. The first order condition then simplifies to

$$[W_{T-1}]: \quad u'(W_{T-2} - W^*_{T-1}) = \beta u'(W^*_{T-1} - \psi_{T-1}(W^*_{T-1})) \quad (6\text{-}3')$$

and this would characterize the "analytical" solution for $W^*_{T-1} = \psi_{T-2}(W_{T-2})$. Also, with

the knowledge that $u(\cdot) = \ln(\cdot)$, we can rewrite (6-3') as

$$\begin{aligned} & \frac{1}{W_{T-2} - W_{T-1}^*} = \frac{\beta}{W_{T-1}^* - \psi_{T-1}(W_{T-1}^*)} \\ \Leftrightarrow \quad & W_{T-1}^* - \psi_{T-1}(W_{T-1}^*) = \beta(W_{T-2} - W_{T-1}^*) \\ \Leftrightarrow \quad & W_{T-2} = \frac{W_{T-1}^* - \psi_{T-1}(W_{T-1}^*)}{\beta} + W_{T-1}^* \end{aligned}$$

and using Exercise 5.5's (5.3), $\psi_{T-1}(W_{T-1}^*) = \frac{\beta W_{T-1}^*}{1+\beta}$. Using this information, collecting the terms for W_{T-1}^* and $W_T^* = \psi_{T-1}(W_{T-1}^*)$ will yield:

$$W_{T-1}^* = \Big(\frac{\beta + \beta^2}{1 + \beta + \beta^2}\Big) W_{T-2} \quad \text{and} \quad W_T^* = \Big(\frac{\beta^2}{1 + \beta + \beta^2}\Big) W_{T-2} \tag{6-A}$$

Let us now plug information in (6-A) into (6-2). We then yield

$$V_{T-2}(W_{T-2}) = u\Big(\frac{W_{T-2}}{1 + \beta + \beta^2}\Big) + \beta u\Big(\frac{\beta W_{T-2}}{1 + \beta + \beta^2}\Big) + \beta^2 u\Big(\frac{\beta^2 W_{T-2}}{1 + \beta + \beta^2}\Big) \tag{6-B}$$

and using $u(\cdot) = \ln(\cdot)$, this can be organized as (although I believe (6-B) is a bit more concise)

$$\begin{aligned} V_{T-2}(W_{T-2}) = (1 + \beta + \beta^2) \ln W_{T-2} + (\beta + \beta^2) \ln \beta \\ - (1 + \beta + \beta^2) \ln (1 + \beta + \beta^2) \end{aligned} \tag{6-B'}$$

Therefore we have characterized the solution for $\psi_{T-2}(W_{T-2})$ in (6-3'), and have found analytical (and/or numerical) solutions to $\psi_{T-2}(W_{T-2})$ and V_{T-2} in (6-A), (6-B) and (6-B').

Exercise 5.7

I admit that my write-up to this part is really long; for a summary of results, please look at the paragraph beginning with "**In summary**" on page 10.

For the sake of clarification, let us try to solve for the same problem as before but in period $T - 3$. The problem becomes

$$\begin{aligned} V_{T-3}(W_{T-3}) = \max_{W_{T-2}} \; & u(W_{T-3} - W_{T-2}) + \beta u(W_{T-2} - \psi_{T-2}(W_{T-2})) \\ & + \beta^2 u(\psi_{T-2}(W_{T-2}) - \psi_{T-1}(\psi_{T-2}(W_{T-2}))) \\ & + \beta^3 u(\psi_{T-1}(\psi_{T-2}(W_{T-2}))) \end{aligned}$$

and the first order condition (FOC) will yield

$$\begin{aligned}[W_{T-2}]: \quad u'(W_{T-3}-W_{T-2}) =& \beta u'(W_{T-2}-\psi_{T-2}(W_{T-2})) \\ & -\beta\psi'_{T-2}(W_{T-2})u'(W_{T-2}-\psi_{T-2}(W_{T-2})) \\ & +\beta^2\psi'_{T-2}(W_{T-2})u'(\psi_{T-2}(W_{T-2})-\psi_{T-1}(\psi_{T-2}(W_{T-2}))) \\ & -\beta^2\psi'_{T-1}(\psi_{T-2}(W_{T-2}))u'(\psi_{T-2}(W_{T-2})-\psi_{T-1}(\psi_{T-2}(W_{T-2}))) \\ & +\beta^3\psi'_{T-1}(\psi_{T-2}(W_{T-2}))\psi'_{T-2}(W_{T-2})u'(\psi_{T-1}(\psi_{T-2}(W_{T-2})))\end{aligned}$$

yet we can notice that the last four terms cancel each other out by using (6-3') from Exercise 5.6 and (4-3) from Exercise 5.4. Therefore, the "cleaned-up" FOC will become

$$[W_{T-2}]: \quad u'(W_{T-3}-W^*_{T-2}) = \beta u'(W_{T-2}-\psi_{T-2}(W^*_{T-2})) \tag{7}$$

which very much resembles the aforementioned equations (and FOCs) (6-3') and (4-3). Therefore, in this line, I argue that for any $s \in \mathbb{N}$, it must be that the FOC to solving the optimal $W^*_{T-s} = \psi_{T-(s+1)}(W_{T-(s+1)})$ is characterized by

$$[W_{T-s}]: \quad u'(W_{T-(s+1)}-W^*_{T-s}) = \beta u'(W^*_{T-s}-\psi_{T-s}(W^*_{T-s})) \tag{7'}$$

and I will complete the proof using mathematical induction. The base case, $s=1$, has already been shown in (6-3') in Exercise 5.6. Let us assume that, for an arbitrary $s' \geq 1$, the similar should hold. That is,

$$[W_{T-s'}]: \quad u'(W_{T-(s'+1)}-W^*_{T-s'}) = \beta u'(W^*_{T-s'}-\psi_{T-s'}(W^*_{T-s'}))$$

and now we simply need to show that the similar should hold for $s = s'+1$. That is, we want to show that

$$[W_{T-(s'+1)}]: \quad u'(W_{T-(s'+2)}-W^*_{T-(s'+1)}) = \beta u'(W^*_{T-(s'+1)}-\psi_{T-(s'+1)}(W^*_{T-(s'+1)})) \tag{7-1}$$

Let us begin by writing the maximization problem in period $T-(s'+2)$. This will be written as

$$V_{T-(s'+2)}(W_{T-(s'+2)}) = \max_{W_{T-(s'+1)}} u(W_{T-(s'+2)}-W_{T-(s'+1)}) + \beta V_{T-(s'+1)}(W_{T-(s'+1)}) \tag{7-2}$$

where

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

By using the inductive assumptions, it can be shown that the derivative of $V_{T-(s'+1)}(W_{T-(s'+1)})$

with respect to $W_{T-(s'+1)}$ is as follows:

$$V'_{T-(s'+1)}(W_{T-(s'+1)}) = u'(W_{T-(s'+1)} - \psi_{T-(s'+1)}(W_{T-(s'+1)}))$$

Let us use this in the maximization problem described in (7-1). The first order conditions will yield us

$$0 = -u(W_{T-(s'+2)} - W^*_{T-(s'+1)}) + \beta u'(W^*_{T-(s'+1)} - \psi_{T-(s'+1)}(W^*_{T-(s'+1)}))$$

which by re-organizing becomes (7-1), which is what we wanted to show. Therefore, by induction, we have proven that the analytical solution of $W^*_{T-(s-1)} = \psi_{T-s}(W_{T-s})$ is characterized by

$$u'(W_{T-s} - W^*_{T-(s-1)}) = \beta u'(W^*_{T-(s-1)} - \psi_{T-(s-1)}(W^*_{T-(s-1)})) \tag{7-A}$$

for any $s \in \mathbb{N}$. Note that we also need to combine the result from (4-3') to yield this result for all $s \in \mathbb{N}$. Reorganizing the equation using $u(\cdot) = \ln(\cdot)$ will get us

$$W_{T-s} = \frac{W^*_{T-(s-1)} - \psi_{T-(s-1)}(W^*_{T-(s-1)})}{\beta} + W^*_{T-(s-1)} \tag{7-A'}$$

To observe a pattern, let us try to numerically solve for W^*_{T-2} and W^*_{T-3} by plugging in $s = 3$ and $s = 4$ above respectively. Using the fact that $\psi_{T-2}(W_{T-2}) = \frac{\beta+\beta^2}{1+\beta+\beta^2} W_{T-2}$, we can show that

$$\begin{aligned}
[s=3]: \quad & \psi_{T-3}(W_{T-3}) \equiv W^*_{T-2} = \Big(\frac{\beta+\beta^2+\beta^3}{1+\beta+\beta^2+\beta^3}\Big) W_{T-3} \\
[s=4]: \quad & \psi_{T-4}(W_{T-4}) \equiv W^*_{T-3} = \Big(\frac{\beta+\beta^2+\beta^3+\beta^4}{1+\beta+\beta^2+\beta^3+\beta^4}\Big) W_{T-4}
\end{aligned}$$

and we see a pattern emerge here. I argue, by induction, that

$$\psi_{T-s}(W_{T-s}) \equiv W^*_{T-(s-1)} = \Big(1 - \frac{1}{\sum_{i=0}^{s} \beta^i}\Big) W_{T-s} \tag{7-B}$$

and to begin with, we have already shown the base case for $s = 1, 2, 3, 4$. Now, suppose that (7-B) holds for some $s = s'$. We need to show that this holds for $s = s' + 1$. Using (7-A'), we see that

$$\begin{aligned}
W_{T-(s'+1)} &= \frac{W^*_{T-s'} - \psi_{T-s'}(W^*_{T-s'})}{\beta} + W^*_{T-s'} \\
&= \Big(\frac{1}{\beta(\sum_{i=0}^{s'} \beta^i)} + 1\Big) W^*_{T-s'} \\
\Leftrightarrow \quad W^*_{T-s'} &= \Big(\frac{\beta+\beta^2+\cdots+\beta^{s'+1}}{1+\beta+\beta^2+\cdots+\beta^{s'+1}}\Big) W_{T-(s'+1)} = \Big(1 - \frac{1}{\sum_{i=0}^{s'+1} \beta^i}\Big) W_{T-(s'+1)}
\end{aligned}$$

and so we have

$$\psi_{T-(s'+1)}(W_{T-(s'+1)}) \equiv W^*_{T-s'} = \Big(1 - \frac{1}{\sum_{i=0}^{s'+1}\beta^i}\Big)W_{T-(s'+1)} \tag{7-B'}$$

which is what we wanted to show. Therefore, we have shown that (7-B) holds for all $s \in \mathbb{N}$. Notice that if we assume $\beta \in (0,1)$, we have $\lim_{s\to\infty}\sum_{i=0}^{s}\beta^i = \frac{1}{1-\beta}$. Therefore,

$$\lim_{s\to\infty}\psi_{T-s}(W_{T-s}) = \Big(1 - \frac{1}{(1-\beta)^{-1}}\Big)\lim_{s\to\infty}W_{T-s} = \lim_{s\to\infty}\beta W_{T-s}$$

and so when $s \to \infty$, we will have $\psi(W_{T-s}) = \beta W_{T-s}$.

Now, let us look at $V_{T-s}(W_{T-s})$. Again, for motivation, let us examine $V_{T-3}(W_{T-3})$. With Exercise 5.6's (6-B), we can write

$$\begin{aligned} V_{T-3}(W_{T-3}) &= u(W_{T-3} - W^*_{T-2}) + \beta V_{T-2}(W^*_{T-2}) \\ &= u\Big(\frac{W_{T-3}}{1+\beta+\beta^2+\beta^3}\Big) + \beta u\Big(\frac{\beta W_{T-3}}{1+\beta+\beta^2+\beta^3}\Big) \\ &\quad + \beta^2 u\Big(\frac{\beta^2 W_{T-3}}{1+\beta+\beta^2+\beta^3}\Big) + \beta^3 u\Big(\frac{\beta^3 W_{T-3}}{1+\beta+\beta^2+\beta^3}\Big) \end{aligned}$$

and again, we see a pattern here. I will argue, and prove again by induction, that for any $s \in \mathbb{N}$,

$$V_{T-s}(W_{T-s}) = \sum_{j=0}^{s}\beta^j u\Big(\frac{\beta^j W_{T-s}}{\sum_{i=0}^{s}\beta^i}\Big) \tag{7-C}$$

Having the base case for $s = 1, 2, 3$, let us assume that (7-C) holds for an arbitrary $s = s'$. We want to show that it holds for $s = s' + 1$ as well. Then, since we know that $W^*_{T-s'} = \Big(1 - \frac{1}{\sum_{i=0}^{s'+1}\beta^i}\Big)W_{T-(s'+1)}$, we can write the following:

$$\begin{aligned} V_{T-(s'+1)}(W_{T-(s'+1)}) &= u(W_{T-(s'+1)} - W^*_{T-s'}) + \beta V_{T-s'}(W^*_{T-s'}) \\ &= u\Big(\frac{W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) + \beta\sum_{j=0}^{s'}\beta^j u\Big(\frac{\beta^j W^*_{T-s'}}{\sum_{i=0}^{s'}\beta^i}\Big) \end{aligned} \tag{7-C'}$$

yet notice that

$$u\Big(\frac{\beta^j W^*_{T-s'}}{\sum_{i=0}^{s'}\beta^i}\Big) = u\Big(\frac{\frac{\beta^{j+1}(1+\beta+\beta^2+\cdots+\beta^{s'})}{1+\beta+\beta^2+\cdots+\beta^{s'+1}}}{\sum_{i=0}^{s'}\beta^i}W_{T-(s'+1)}\Big) = u\Big(\frac{\beta^{j+1}W^*_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big)$$

and so (7-C') becomes

$$\begin{aligned} V_{T-(s'+1)}(W_{T-(s'+1)}) &= u\Big(\frac{W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) + \sum_{j=0}^{s'}\beta^{j+1}u\Big(\frac{\beta^{j+1}W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) \\ &= u\Big(\frac{W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) + \sum_{j=1}^{s'+1}\beta^{j}u\Big(\frac{\beta^{j}W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) \\ &= \sum_{j=0}^{s'+1}\beta^{j}u\Big(\frac{\beta^{j}W_{T-(s'+1)}}{\sum_{i=0}^{s'+1}\beta^i}\Big) \end{aligned} \tag{7-C"}$$

which is exactly what we wanted to show.

Knowing $u(\cdot) = \ln(\cdot)$, (7-C) can be rewritten as:

$$V_{T-s}(W_{T-s}) = \sum_{j=0}^{s}\beta^j\Big[\ln W_{T-s} + j\ln\beta - \ln\Big(\sum_{i=0}^{s}\beta^i\Big)\Big] \tag{7-D}$$

and let us see what happens if we take the limit $s \to \infty$. Notice that $\sum_{j=0}^{\infty}\beta^j = \frac{1}{1-\beta}$ and $\sum_{j=0}^{\infty} j\beta^j = \frac{\beta}{(1-\beta)^2}$ as $\beta \in (0,1)$. Then we can rewrite (7-D) as follows:

$$\text{as } s\to\infty, \quad V_{T-s}(W_{T-s}) \to V(W_{T-s}) = \frac{\ln W_{T-s}}{1-\beta} + \frac{\beta\ln\beta}{1-\beta^2} + \frac{\ln(1-\beta)}{1-\beta} \tag{7-D'}$$

In summary, for all $s \in \mathbb{N}$, $W^*_{T-(s-1)} = \psi_{T-s}(W_{T-s})$ is characterized by (7'-1):

$$u'(W_{T-s} - W^*_{T-(s-1)}) = \beta u'(W^*_{T-(s-1)} - \psi_{T-(s-1)}(W^*_{T-(s-1)})) \tag{7'-1}$$

and using $u(\cdot) = \ln(\cdot)$, we can write $\psi_{T-s}(W_{T-s})$ as in (7-B) (reproduced below):

$$\psi_{T-s}(W_{T-s}) \equiv W^*_{T-(s-1)} = \Big(1 - \frac{1}{\sum_{i=0}^{s}\beta^i}\Big)W_{T-s} \tag{7-B}$$

and $V_{T-s}(W_{T-s})$ as in (7-C) (reproduced) below:

$$V_{T-s}(W_{T-s}) = \sum_{j=0}^{s}\beta^j u\Big(\frac{\beta^j W_{T-s}}{\sum_{i=0}^{s}\beta^i}\Big) \tag{7-C}$$

Taking the limit $s \to \infty$ will yield us the result in (7-D') for $V_{T-s}(W_{T-s})$ (reproduced below):

$$V_{T-s}(W_{T-s}) \to V(W_{T-s}) = \frac{\ln W_{T-s}}{1-\beta} + \frac{\beta\ln\beta}{1-\beta^2} + \frac{\ln(1-\beta)}{1-\beta} \tag{7-D'}$$

and $\psi_{T-s}(W_{T-s}) \to \psi(W_{T-s}) = \beta W_{T-s}$.

Exercise 5.8

I will write the Bellman equation for the case $s \to \infty$ in (8) below:

$$\lim_{s\to\infty} V_{T-s}(W_{T-s}) = V(W_{T-s}) = \max_{W_{T-(s-1)}} u(W_{T-s} - W_{T-(s-1)}) + V(W_{T-(s-1)}) \qquad (8)$$

Exercise 5.9 to Exercise 5.22

Attached on the next page. Also refer to the accompanied `.ipynb` file.

References

Evans, Richard. 2019. "Chapter 5: Dynamic Programming." Unpublished manuscript.

PS3_junho3_part2

January 30, 2019

0.0.1 Exercise 5.9

In the below section, I have used *numpy*'s *linspace* to generate vector W that contains 100 elements from 0.01 to 1, evenly spaced (with increments of 0.01).

```
In [717]: import numpy as np
```

```
In [718]: N = 100
          W_max, W_min = 1, 0.01
          W = np.linspace(W_min, W_max, N)
          print(W)
```

```
[ 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.  ]
```

0.0.2 Exercise 5.10

Using (5.9) from Evans (2019, p. 49), we can characterize the problem for $t = T + 1$ as follows:

$$V_T(W) \equiv C(V_{T+1}(W)) \equiv \max_{W' \in [0,W]} u(W - W') + \beta V_{T+1}(W') \tag{1}$$

which is equivalent to (5.34) on Evans (2019, p. 68) as expectedly. We now follow the steps in Section 5.9 of Evans (2019, pp. 68-69). First, in the below section, I generate all the possible combinations of $W - W'$ (denoted *W_Wp*)that will be used in the future steps. Note that for W and W', I will use the vector created in Exercise 5.9.

```
In [745]: W_possibles = np.tile(W.reshape((N, 1)), (1, N))
          print(W_possibles)
          print()
          W_Wp = W_possibles - W_possibles.transpose()
          print(W_Wp)
```

```
[[ 0.01  0.01  0.01 ...,  0.01  0.01  0.01]
 [ 0.02  0.02  0.02 ...,  0.02  0.02  0.02]
 [ 0.03  0.03  0.03 ...,  0.03  0.03  0.03]
 ...,
 [ 0.98  0.98  0.98 ...,  0.98  0.98  0.98]
 [ 0.99  0.99  0.99 ...,  0.99  0.99  0.99]
 [ 1.    1.    1.   ...,  1.    1.    1.  ]]

[[ 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.  ]]
```

Notice that in the above matrix (in numpy array form), there are nonpositive values. On such values, the natural logarithm function will be undefined; therefore, we switch such numbers with a very small positive number; Evans (2019, p. 70) suggests 10^{-10}, and this is what I will use.

```
In [746]: verysmallpos = 10 ** (-10)
          nonpos = W_Wp <= 0
          W_Wp[nonpos] = verysmallpos
```

We know that, as the question itself has elaborated, $V_{T+1}(W')$ is a column vector of N zeroes. Therefore, the maximization problem will not need to take the part $\beta V_{T+1}(W')$ into consideration. Just in case, however, I create the vector *vTp1* (representing $V_{T+1}(W')$) in the below section.

```
In [721]: vTp1 = np.array([0] * N)
```

I define the function *ln_w_np* below to evaluate natural logarithms. Note that since natural logarithms are undefined over nonpositive values, they will take the output of a very large negative number; Evans (2019, p. 70) suggests the use of -10^{10}, and I will use this.

```
In [722]: from numpy import log as ln
          import math
          from matplotlib import pyplot as plt

In [762]: verylargeneg = (-1) * (10 ** 10)

          def ln_w_np(xval):
              try:
                  rtnval = ln(xval)
              except ValueError:
                  rtnval = verylargeneg
              return rtnval

In [763]: verylargeneg / 10000
```

```
Out[763]: -1000000.0
```

Let us now define a (rather trivial) function that will give us the utility function (denoted *Ufn*). But $u(\cdot) = \ln(\cdot)$, so we can write the following.

```
In [724]: def Ufn(mat):
              return ln_w_np(mat)
```

Then *W_Wp* evaluated by the utility function will be as follows:

```
In [725]: U = Ufn(W_Wp)
          print("Dimensions are:", U.shape)
          print(U)
```

```
Dimensions are: (100, 100)
[[ -2.30258509e+01  -2.30258509e+01  -2.30258509e+01 ...,  -2.30258509e+01
   -2.30258509e+01  -2.30258509e+01]
 [ -4.60517019e+00  -2.30258509e+01  -2.30258509e+01 ...,  -2.30258509e+01
   -2.30258509e+01  -2.30258509e+01]
 [ -3.91202301e+00  -4.60517019e+00  -2.30258509e+01 ...,  -2.30258509e+01
   -2.30258509e+01  -2.30258509e+01]
 ...,
 [ -3.04592075e-02  -4.08219945e-02  -5.12932944e-02 ...,  -2.30258509e+01
   -2.30258509e+01  -2.30258509e+01]
 [ -2.02027073e-02  -3.04592075e-02  -4.08219945e-02 ...,  -4.60517019e+00
   -2.30258509e+01  -2.30258509e+01]
 [ -1.00503359e-02  -2.02027073e-02  -3.04592075e-02 ...,  -3.91202301e+00
   -4.60517019e+00  -2.30258509e+01]]
```

We now maximize over the rows (which is the direction that W' took), and the result will be what is represented in vT; this will be the result of evaluating W with the value function V_T (i.e. $V_T(W)$). The W' values that returned the maxima to calculate the value function is the policy function ψ_T evaluated at W (i.e. $W' = \psi_T(W)$); this will be represented in pT. Notice that as there should be no cake transferred to period $T+1$, W' takes the smallest value possible, which is 0.01 for our configurations.

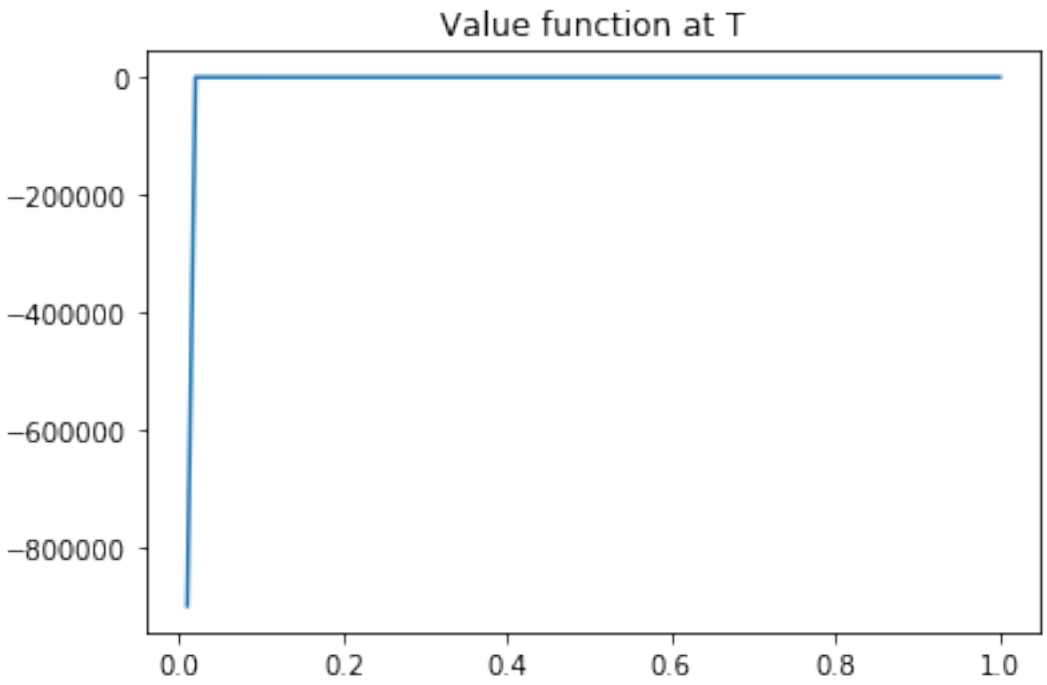
To promote future uses, we define function *valpol* for calculating value and policy function approximations. This will take the matrix of possible values of $W - W'$ (which is W_Wp in our case), $V(W')$, and β value initialized to be 0.9. Note that the upper triangular part of the overall matrix (i.e. $U(W - W') + \beta V(W')$) is where $W - W'$ is negative, and so it should only return the smallest W' possible (0.01 in our case). Therefore, they are punished with a large negative number; in my case, I have put in $-1,000,000$.

```
In [895]: def valpol(W_Wp, VW, beta=0.9):

              ## evaluating utility function
              U = Ufn(W_Wp)

              ## repeating VW N times
```

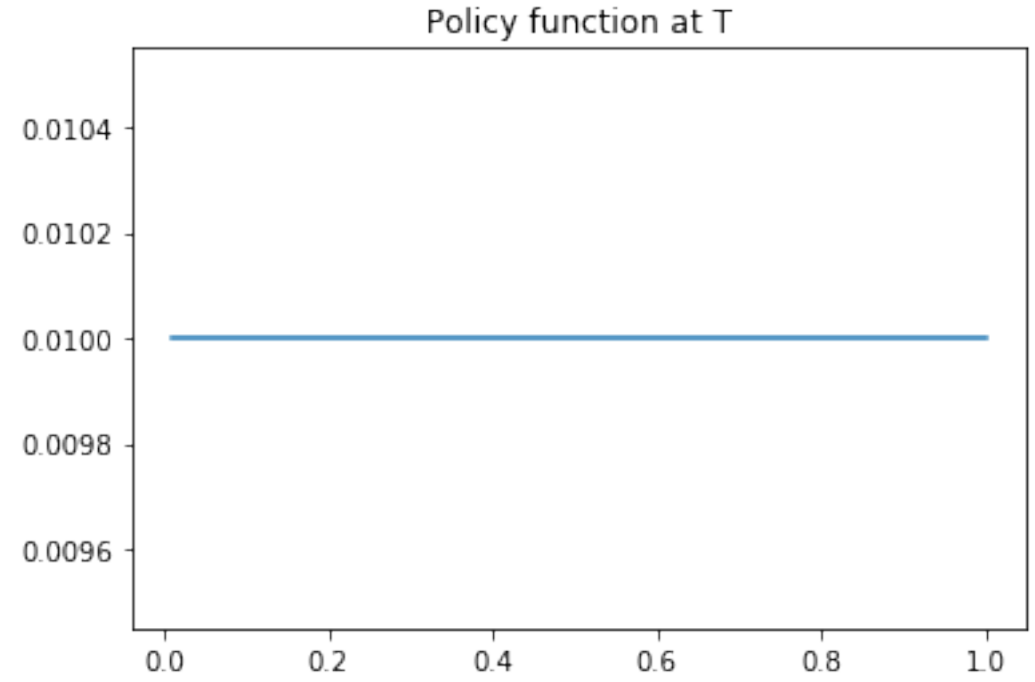
```
    VW_mat = np.tile(VW, (N, 1))

    ## should not account for W-W' that is negative, so do this
    ## putting in verylargeneg produced errors; try this instead
    VW_mat[nonpos] = verylargeneg / 10000

    ## contraction
    contraction = U + (beta * VW_mat)

    ## value function evaluation, one more iteration
    ## note that W' was on the columns, so axis=1
    VW_iter = contraction.max(axis=1)

    ## policy function evaluation, one more iteration
    polidx = np.argmax(contraction, axis=1)
    pol_iter = W[polidx]

    return VW_iter, pol_iter, polidx
```

Using the above function, $V_T(W')$ and $W' = \psi_T(W)$ can be found as follows; their graphs are presented below.

```
In [897]: ## value and policy at T
          vT, pT, a = valpol(W_Wp, vTp1)
          a.shape

Out[897]: (100,)

In [766]: ## value at T
          plt.plot(W,vT)
          plt.title("Value function at T")
          plt.show()
```



In [767]:
```
## policy at T
plt.plot(W,pT)
plt.title("Policy function at T")
plt.show()
```



0.0.3 Exercise 5.11

In the below section, I define function *norm_val_fns* to calculate the sum of squared differences for $V_t(W)$ and $V_{t+1}(W')$. The said function requires 2 inputs, v and *v_plus_1* to indicate $V_t(W)$ and $V_{t+1}(W')$ (in lists or numpy arrays).

```
In [768]: def norm_val_fns(v1, v2):

              err_msg = "v1 and v2 should be of the same length"
              assert len(v1) == len(v2), err_msg

              if type(v1) == list:
                  v1 = np.array(v1)

              if type(v2) == list:
                  v2 = np.array(v2)

              v1_2 = v1 - v2
              mult = v1_2 * v1_2

              return np.sum(mult)
```

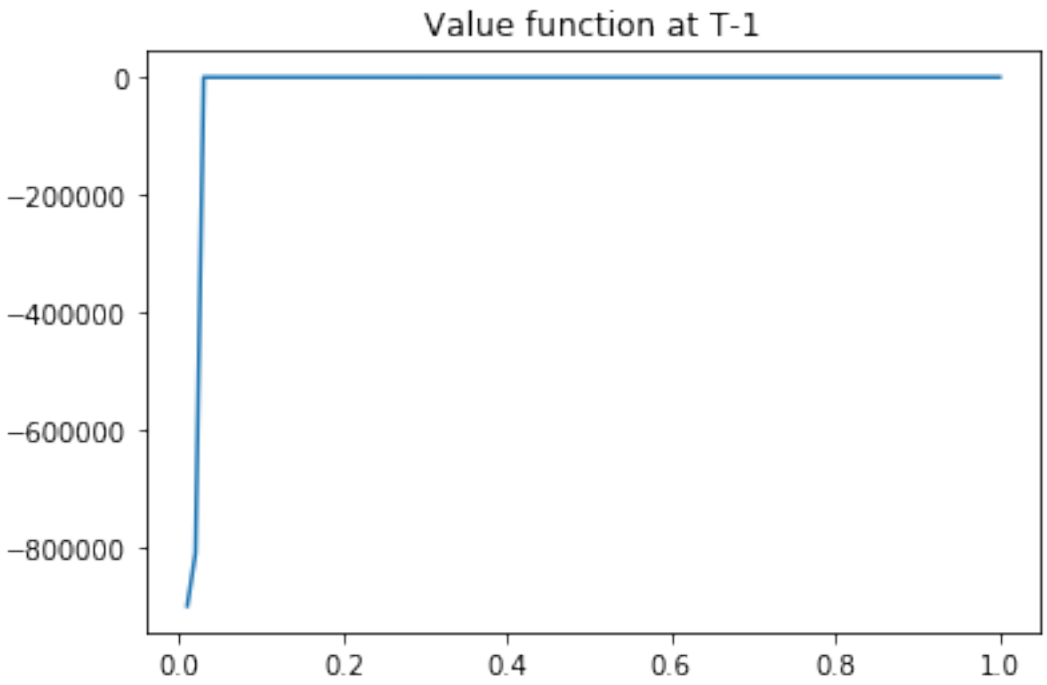
Using the above function, δ_T can be calculated as $\delta_T \approx 8.1 \times 10^{11}$. Note that the large distance is most likely due to punishing the upper triangle part of the matrix with a negative number with very large absolute value (i.e. $-1,000,000$.)

```
In [769]: d_T = norm_val_fns(vT, vTp1)
          print('delta_T :', d_T)

delta_T : 810041447241.0
```

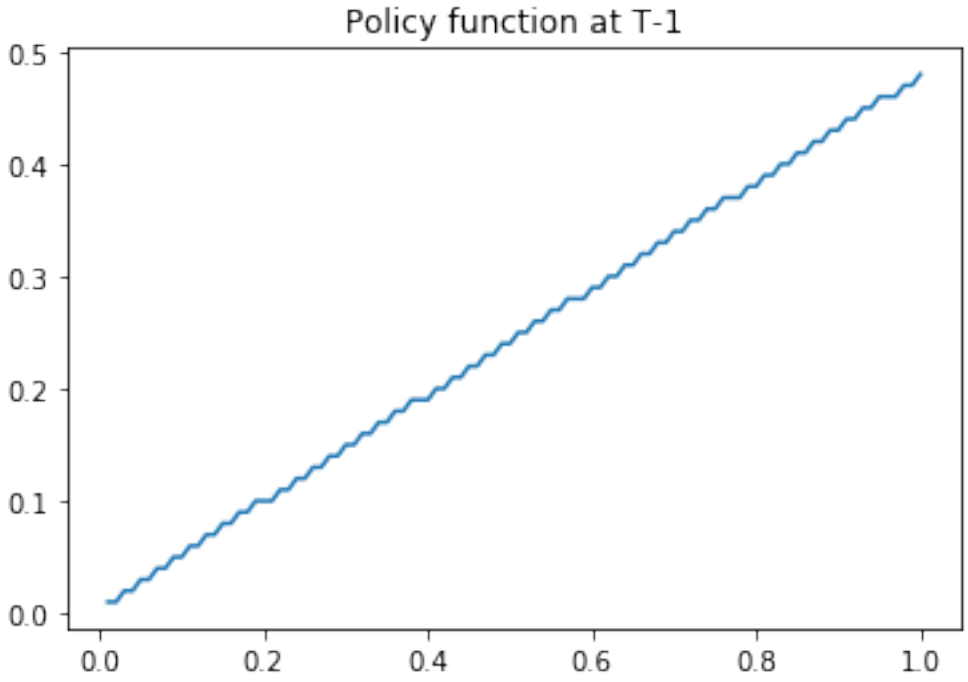
0.0.4 Exercise 5.12

Using *valpol* and the past value for value function approximation at time T, we can calculate V_{T-1} and ψ_{T-1} as follows.

```
In [770]: vTm1, pTm1 = valpol(W_Wp, vT)

In [771]: ## value at T-1
          plt.plot(W,vTm1)
          plt.title("Value function at T-1")
          plt.show()
```



In [773]:
```
## policy at T-1
plt.plot(W,pTm1)
plt.title("Policy function at T-1")
plt.show()
```



Distance $\delta_{T-1} \approx 6.6 \times 10^{11}$ has decreased slightly than before ($\delta_T \approx 8.1 \times 10^{11}$).

```
In [774]: d_Tm1 = norm_val_fns(vTm1, vT)
          print('delta_(T-1) :', d_Tm1)
```

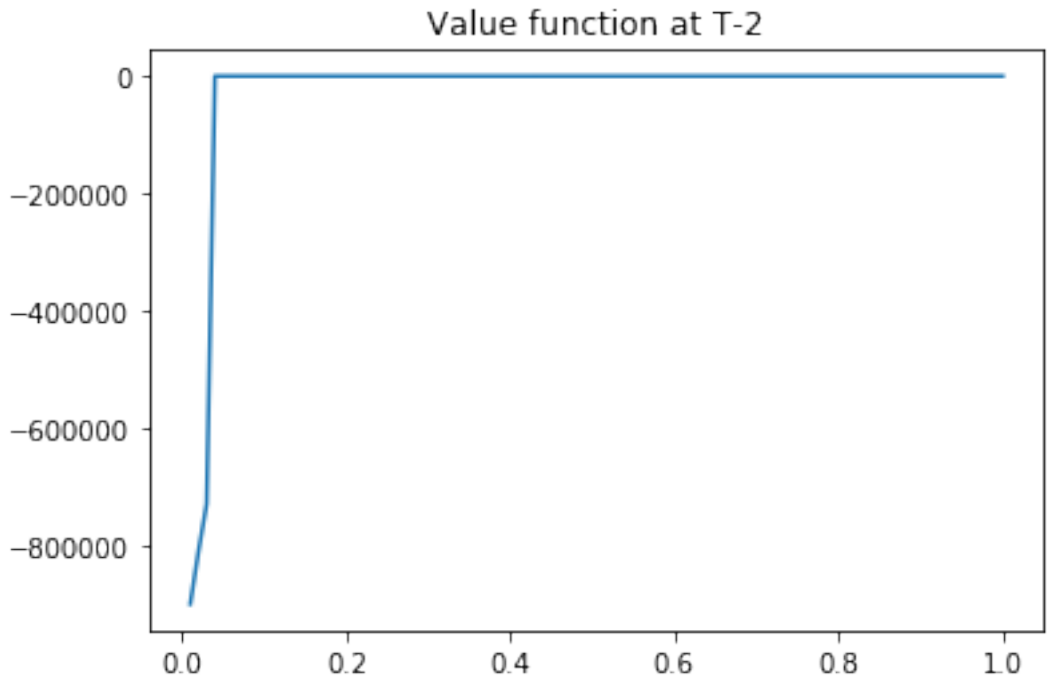
```
delta_(T-1) : 656133572636.0
```

0.0.5 Exercise 5.13

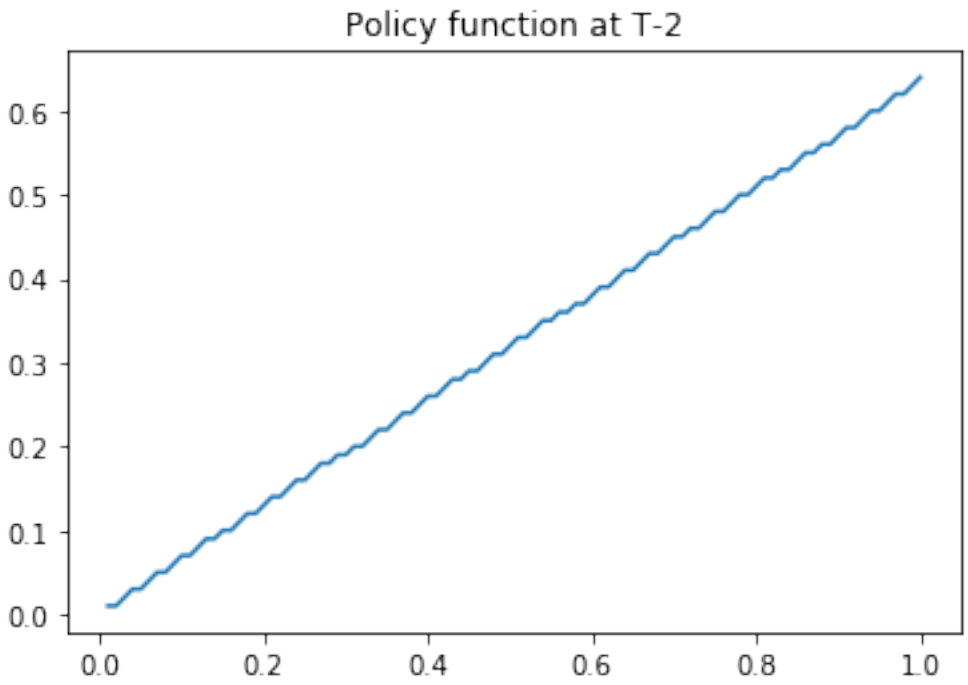
Once again, we can calculate the value and policy function evaluations at $T-2$ (i.e. V_{T-2} and ψ_{T-2}) as follows.

```
In [788]: vTm2, pTm2 = valpol(W_Wp, vTm1)
```

```
In [790]: ## value at T-2
          plt.plot(W,vTm2)
          plt.title("Value function at T-2")
          plt.show()
```




```
In [792]: ## policy at T-2
          plt.plot(W,pTm2)
          plt.title("Policy function at T-2")
          plt.show()
```



Distance $\delta_{T-2} \approx 5.3 \times 10^{11}$ has decreased in comparison to $\delta_{T-1} \approx 6.6 \times 10^{11}$ and $\delta_T \approx 8.1 \times 10^{11}$. Let us further iterate to see (or fail to see) convergence.

```
In [793]: d_Tm2 = norm_val_fns(vTm2, vTm1)
          print('delta_(T-2) :', d_Tm2)
```

```
delta_(T-2) : 531468194013.0
```

0.0.6 Exercise 5.14

In the below function *dynprog*, I reiterate the step-by-step contraction mapping over and over again until the distance is small enough that the system converges. I use the value of 10^{-9} for the distance that is "small enough."

```
In [784]: small = 1e-9
```

```
          def dynprog(W_Wp, V, beta=0.9):
              W_l = W_Wp.shape[0]
              Wp_l = W_Wp.shape[1]

              err_msg = "Dimensions should be the same"
              assert W_l == Wp_l and Wp_l == len(V), err_msg

              s = 1
              inputV = V
```

```
    d = small + 1

    while d >= small:
        opener = "Iteration "+str(s)+":"
        newV, newP = valpol(W_Wp, inputV, beta)
        d = norm_val_fns(newV, inputV)
        print(opener, "distance =", d)
        inputV = newV
        s += 1

    print("--------------------")
    print("Success at iteration "+str(s-1)+":", "distance =", d)

    return newV, newP
```

In the below section, we see that it takes 101 iterations to complete the entire contraction operation.

```
In [785]: val_cvg, pol_cvg = dynprog(W_minus_W_prime, val_Tp1)

Iteration 1: distance = 810041447241.0
Iteration 2: distance = 656133572636.0
Iteration 3: distance = 531468194013.0
Iteration 4: distance = 430489237261.0
Iteration 5: distance = 348696282256.0
Iteration 6: distance = 282443988679.0
Iteration 7: distance = 228779630868.0
Iteration 8: distance = 185311501031.0
Iteration 9: distance = 150102315856.0
Iteration 10: distance = 121582875859.0
Iteration 11: distance = 98482129458.5
Iteration 12: distance = 79770524871.2
Iteration 13: distance = 64614125153.5
Iteration 14: distance = 52337441380.7
Iteration 15: distance = 42393327523.6
Iteration 16: distance = 34338595298.4
Iteration 17: distance = 27814262195.4
Iteration 18: distance = 22529552381.4
Iteration 19: distance = 18248937431.8
Iteration 20: distance = 14781639322.2
Iteration 21: distance = 11973127853.1
Iteration 22: distance = 9698233563.05
Iteration 23: distance = 7855569187.98
Iteration 24: distance = 6363011044.08
Iteration 25: distance = 5154038947.42
Iteration 26: distance = 4174771549.03
Iteration 27: distance = 3381564956.26
Iteration 28: distance = 2739067616.05
```

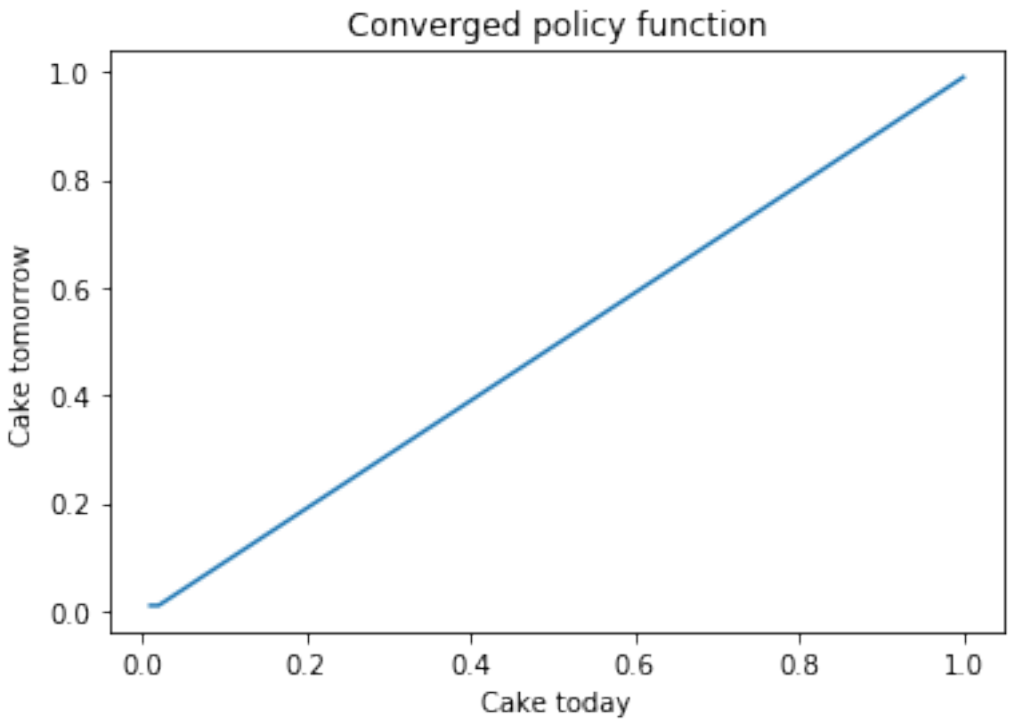
```
Iteration 29: distance = 2218644770.4
Iteration 30: distance = 1797102265.37
Iteration 31: distance = 1455652836.23
Iteration 32: distance = 1179078798.58
Iteration 33: distance = 955053828.034
Iteration 34: distance = 773593601.849
Iteration 35: distance = 626610818.6
Iteration 36: distance = 507554764.133
Iteration 37: distance = 411119359.982
Iteration 38: distance = 333006682.59
Iteration 39: distance = 269735413.876
Iteration 40: distance = 218485686.194
Iteration 41: distance = 176973406.749
Iteration 42: distance = 143348460.378
Iteration 43: distance = 116112253.798
Iteration 44: distance = 94050926.4522
Iteration 45: distance = 76181251.286
Iteration 46: distance = 61706814.3872
Iteration 47: distance = 49982520.4862
Iteration 48: distance = 40485842.4144
Iteration 49: distance = 32793533.1644
Iteration 50: distance = 26562762.6614
Iteration 51: distance = 21515838.544
Iteration 52: distance = 17427829.9996
Iteration 53: distance = 14116543.0695
Iteration 54: distance = 11434400.6479
Iteration 55: distance = 9261865.27879
Iteration 56: distance = 7502111.62159
Iteration 57: distance = 6076711.1521
Iteration 58: distance = 4922136.76448
Iteration 59: distance = 3986931.50376
Iteration 60: distance = 3229415.23529
Iteration 61: distance = 2615827.05135
Iteration 62: distance = 2118820.61478
Iteration 63: distance = 1716245.39472
Iteration 64: distance = 1390159.45959
Iteration 65: distance = 1126029.84571
Iteration 66: distance = 912084.851218
Iteration 67: distance = 738789.399192
Iteration 68: distance = 598420.075085
Iteration 69: distance = 484720.914662
Iteration 70: distance = 392624.58766
Iteration 71: distance = 318026.554633
Iteration 72: distance = 257602.140561
Iteration 73: distance = 208658.355962
Iteration 74: distance = 169013.881248
Iteration 75: distance = 136901.848497
Iteration 76: distance = 110891.09234
```

```
Iteration 77: distance = 89822.3682302
Iteration 78: distance = 72756.6906315
Iteration 79: distance = 58933.4818508
Iteration 80: distance = 47736.6710668
Iteration 81: distance = 38667.2401787
Iteration 82: distance = 31320.9876622
Iteration 83: distance = 25370.5110143
Iteration 84: distance = 20550.6050721
Iteration 85: distance = 16646.4646846
Iteration 86: distance = 13484.096096
Iteration 87: distance = 10922.5530993
Iteration 88: distance = 8847.68287051
Iteration 89: distance = 7167.01967214
Iteration 90: distance = 5805.64666365
Iteration 91: distance = 4702.91194555
Iteration 92: distance = 3809.65263538
Iteration 93: distance = 3086.08473681
Iteration 94: distance = 2499.94020953
Iteration 95: distance = 2025.12876605
Iteration 96: distance = 1640.46033978
Iteration 97: distance = 1328.79107697
Iteration 98: distance = 1076.2305423
Iteration 99: distance = 871.522651457
Iteration 100: distance = 705.54401115
Iteration 101: distance = 0.0
--------------------
Success at iteration 101: distance = 0.0
```

0.0.7 Exercise 5.15

We can plot the converged policy function as follows.

```
In [798]: plt.plot(W, pol_cvg)
          plt.title("Converged policy function")
          plt.xlabel("Cake today")
          plt.ylabel("Cake tomorrow")
          plt.show()
```



0.0.8 Exercise 5.16

For this exercise, I borrow the *ncNormal* function that I have created in the previous problem set for the creation of vectors ϵ and $\Gamma(\epsilon)$. Note that we set the parameters first in the section directly below.

```
In [521]: M = 7
          sigma = 0.5
          mu = sigma * 4
          k = 3

In [527]: from scipy.stats import norm as nr
          import numpy as np
          import math

          def ncNormal(mu, sig, N, k):
              '''
              Given mean and standard deviation of a normal distribution, along with
              the number of nodes (to be equally spaced) and number of standard
              deviations to be away from the mean, returns vector of weights and
              vector of nodes for Newton-Cotes quadrature method.

              input:
              mu: mean of the normal distribution
```

```
    sig: standard deviation of the normal distribution
    N: number of nodes
    k: number of standard deviations to be away from the mean

    output:
    tuple (double) containing
    - vector of nodes
    - vector of weights

    '''

    # Lower and upper bounds (i.e. furthest nodes)
    lb, ub = mu - sig*k, mu + sig*k

    weights = []

    nodes = np.linspace(lb, ub, N)
    case_1 = nr.cdf((nodes[0] + nodes[1]) / 2, mu, sig)
    case_N = 1 - nr.cdf((nodes[-1] + nodes[-2]) / 2, mu, sig)

    for i in range(1, N+1):
        if i == 1:
            weights.append(case_1)
        elif i == N:
            weights.append(case_N)
        else:
            now = nodes[i-1]
            fwd, bwd = nodes[i], nodes[i-2]
            z_min = (bwd + now) / 2
            z_max = (fwd + now) / 2
            weights.append(nr.cdf(z_max, mu, sig) - nr.cdf(z_min, mu, sig))

    return np.array(nodes), np.array(weights)
```

In [528]:
```
e_vec, gamma_vec = ncNormal(mu, sigma, 7, 3)
```

In [801]:
```
print(e_vec)
print()
print(gamma_vec)
print("Checking sum to 1:", np.sum(gamma_vec)==1)
```

```
[ 0.5  1.   1.5  2.   2.5  3.   3.5]

[ 0.00620967  0.06059754  0.24173034  0.38292492  0.24173034  0.06059754
  0.00620967]
Checking sum to 1: True
```

0.0.9 Exercise 5.17

Let us first generate the 7×100 (for convenience's sake) zero matrix for $V_{T+1}(W', \epsilon')$.

```
In [926]: VeTp1 = [0] * N
          VeTp1 = [VeTp1] * M
          VeTp1 = np.array(VeTp1)
          VeTp1.shape

Out[926]: (7, 100)
```

The below section defines function eU which will return the $\epsilon u(W - W')$ values (in matrix of dimensions $7 \times 100 \times 100$, for convenience's sake), given matrix of $W - W'$ values and a vector of ϵs. Note that $u(\cdot) = \ln(\cdot)$.

```
In [919]: def eU(W_Wp, evec):

              eUmat = []

              ## eUmat should result in dimensions x: N y: M z: N
              for e in evec:
                  interm = e * ln_w_np(W_Wp)
                  eUmat.append(interm)

              return np.array(eUmat)
```

The below section defines EV which will return $\mathbb{E}_{\epsilon'}[V(W', \epsilon')]$ values (in vector of length 100), given $V(W', \epsilon')$ (denoted V below), β, and the probabilities over ϵ' which is $\Gamma(\epsilon')$.

```
In [936]: def EV(V, probs, beta=0.9):

              EVvec = []

              ## each row is across different values of epsilon
              ## for a certain value of W'
              for i in range(N):
                  column = V[:, i]
                  Expec = np.sum(column * probs)
                  EVvec.append(Expec)

              return np.array(EVvec)
```

The below section will transform the vector of $\mathbb{E}_{\epsilon'}[V(W', \epsilon')]$ values into a 3-dimensional array for the sake of calculation. Note that the upper triangle part of each "slice" across axis with ϵ (with dimensions 100×100) should be punished with a negative number with large absolute value (in our case, $-1,000,000$ again) as such values are where $W - W'$ takes a nonpositive value.

```
In [937]: def EV_to_cube(EV):
```

```
    EV_cube = []
    for i in range(M):
        EV_slice = []
        for j in range(N):
            EV_slice.append(EV)
        EV_slice = np.array(EV_slice)
        EV_slice[nonpos] = verylargeneg / 10000
        EV_cube.append(EV_slice)

    return np.array(EV_cube)
```

Now, the below function *eU_bEV* and *vp_stoch* will combine the above process and produce the matrices of value functions and policy functions.

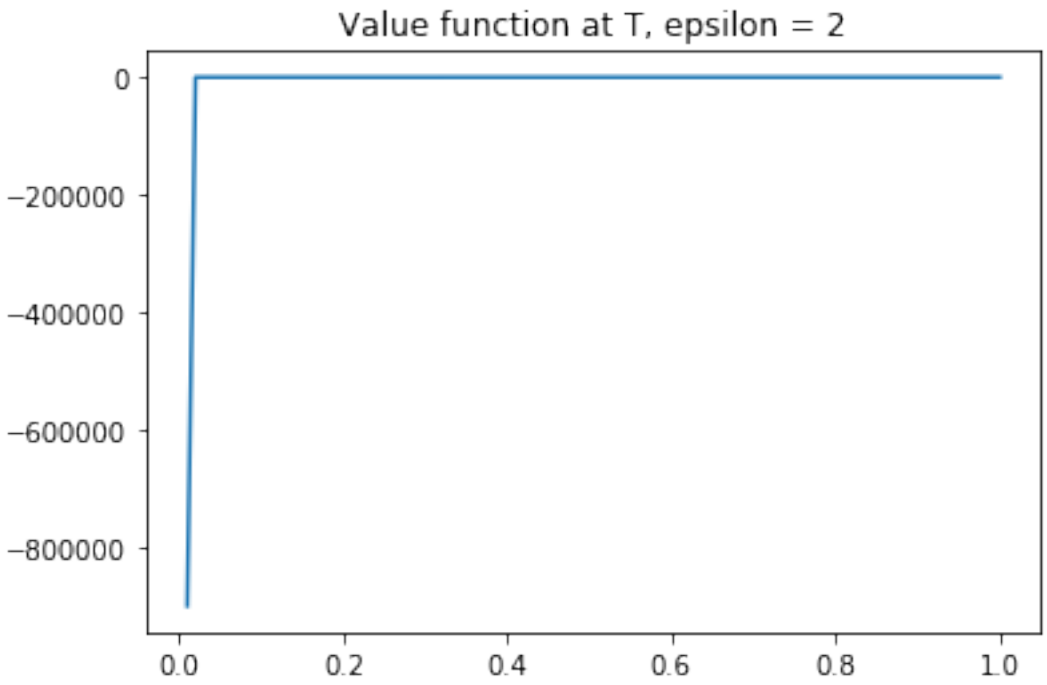
```
In [942]: def eU_bEV(eU_cube, EV_cube, beta=0.9):

              entire = eU_cube + beta * EV_cube

              Vprimes = []
              Wprimes = []
              for i in range(M):
                  contraction_e = entire[i]
                  val_iter = contraction_e.max(axis=1)
                  polidx = np.argmax(contraction_e, axis=1)
                  pol_iter = W[polidx]
                  Vprimes.append(val_iter)
                  Wprimes.append(pol_iter)

              return np.array(Vprimes), np.array(Wprimes)

In [943]: def vp_stoch(W_Wp, VWe, evec, probs, beta=0.9):

              eUcube = eU(W_Wp, evec)
              EVvec = EV(VWe, probs)
              EVcube = EV_to_cube(EVvec)

              return eU_bEV(eUcube, EVcube)
```

We can then calculate $V_T(W, \epsilon)$ and $\psi_T(W, \epsilon)$ as follows.

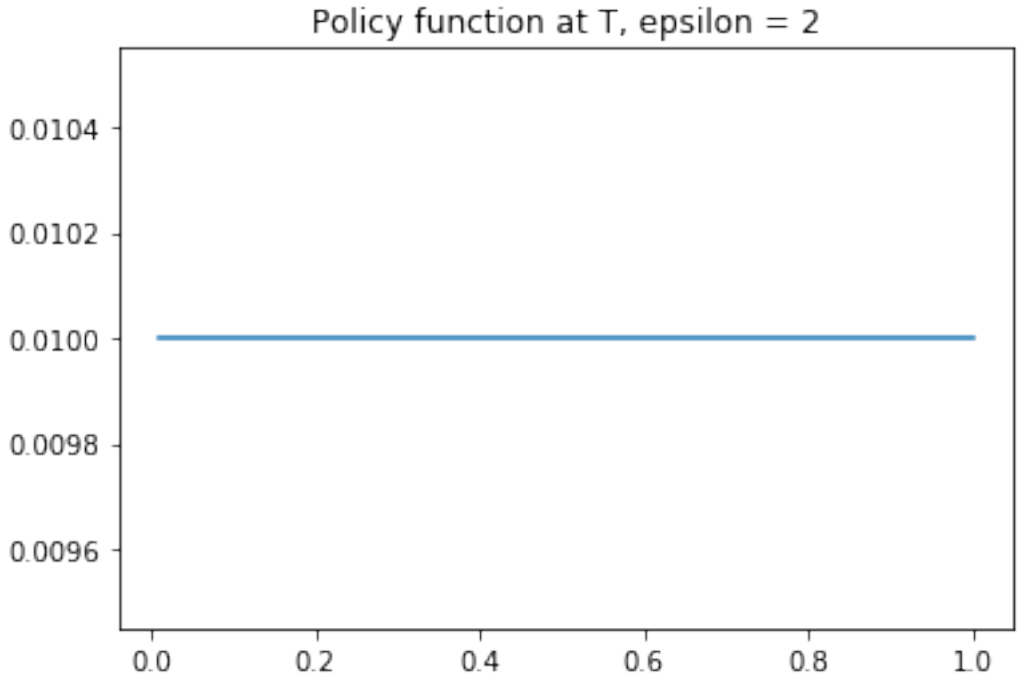
```
In [945]: VeT, PeT = vp_stoch(W_Wp, VeTp1, e_vec, gamma_vec)
```

Let us plot the value function and policy function (at $\epsilon = \mu = 2$) for period T.

```
In [946]: ## value at T, epsilon = 2
          plt.plot(W,VeT[3])
          plt.title("Value function at T, epsilon = 2")
          plt.show()
```



In [947]:
```
## policy at T, epsilon = 2
plt.plot(W,PeT[3])
plt.title("Policy function at T, epsilon = 2")
plt.show()
```



0.0.10 Exercise 5.18

First I define the function *vectorize* to vectorize a matrix as its name strongly hints.

```
In [948]: def vectorize(mat):
              mat = mat.transpose()
              vector = []
              for row in mat:
                  vector += list(row)

              return np.array(vector)
```

Next I define the function *norm_val_stoch* which vectorizes and finds the corresponding norm for two matrices.

```
In [949]: def norm_val_stoch(v_T, v_Tp1):

              err_msg = "The two inputs must have the same dimensions"
              assert v_T.shape == v_Tp1.shape, err_msg

              if type(v_T) == list:
                  v_T = np.array(v_T)

              if type(v_Tp1) == list:
                  v_Tp1 = np.array(v_Tp1)

              vecT = vectorize(v_T)
              vecTp1 = vectorize(v_Tp1)

              vec_diff = vecT - vecTp1
              mult_vec = vec_diff * vec_diff

              return np.sum(mult_vec)
```

Then, using these functions, δ_T can be found as follows; $\delta_T \approx 5.671 \times 10^{12}$.

```
In [950]: d_T = norm_val_stoch(VeT, VeTp1)
          print(d_T)
```

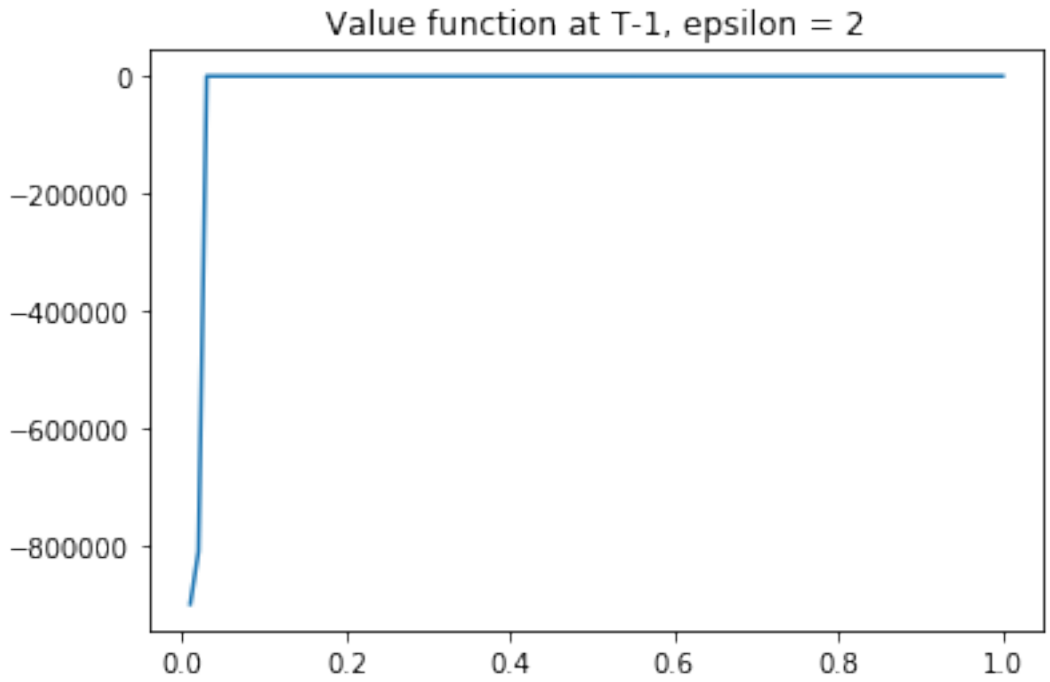
```
5.67058027626e+12
```

0.0.11 Exercise 5.19

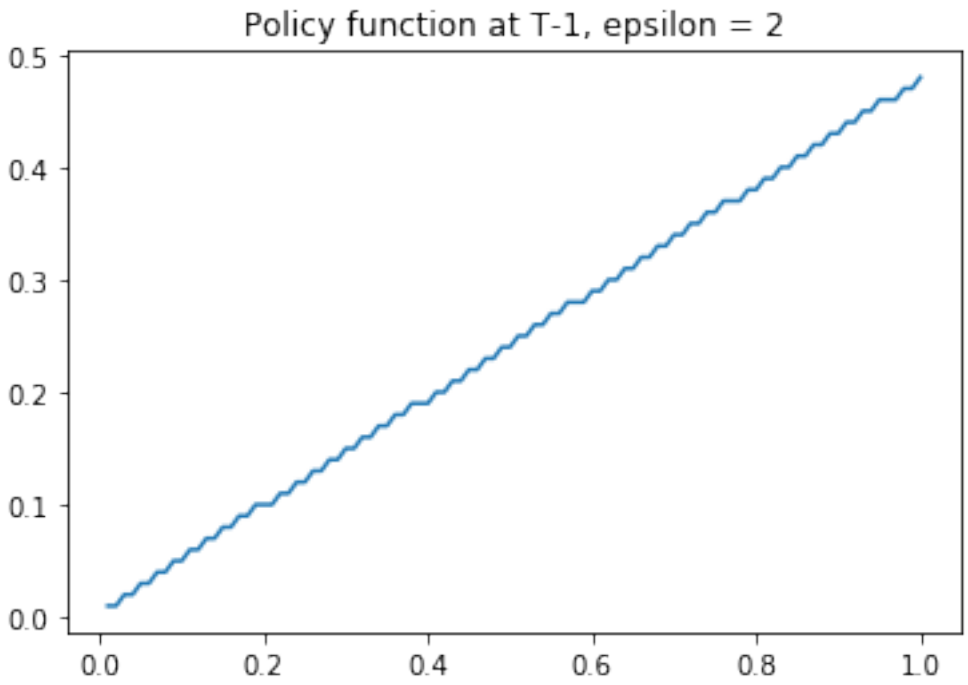
We have generated V_{T-1} and ψ_{T-1} as follows, and let us plot them for $\epsilon = \mu = 2$.

```
In [951]: VeTm1, PeTm1 = vp_stoch(W_Wp, VeT, e_vec, gamma_vec)
```

In [952]:
```
## value at T-1, epsilon = 2
plt.plot(W,VeTm1[3])
plt.title("Value function at T-1, epsilon = 2")
plt.show()
```



In [953]:
```
## policy at T-1, epsilon = 2
plt.plot(W,PeTm1[3])
plt.title("Policy function at T-1, epsilon = 2")
plt.show()
```



We then find $\delta_{T-1} \approx 5.667 \times 10^{12}$, which is smaller than $\delta_T \approx 5.671 \times 10^{12}$.

```
In [954]: d_Tm1 = norm_val_stoch(VWe_Tm1, VWe_T)
          print(d_Tm1)
```

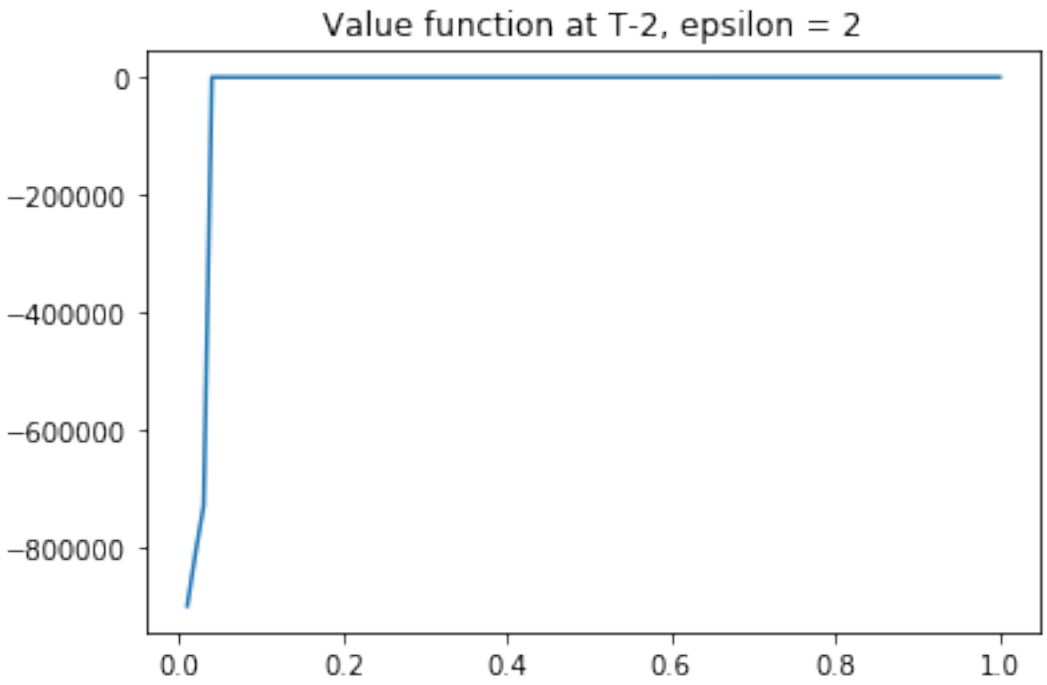
```
5.66731540908e+12
```

0.0.12 Exercise 5.20

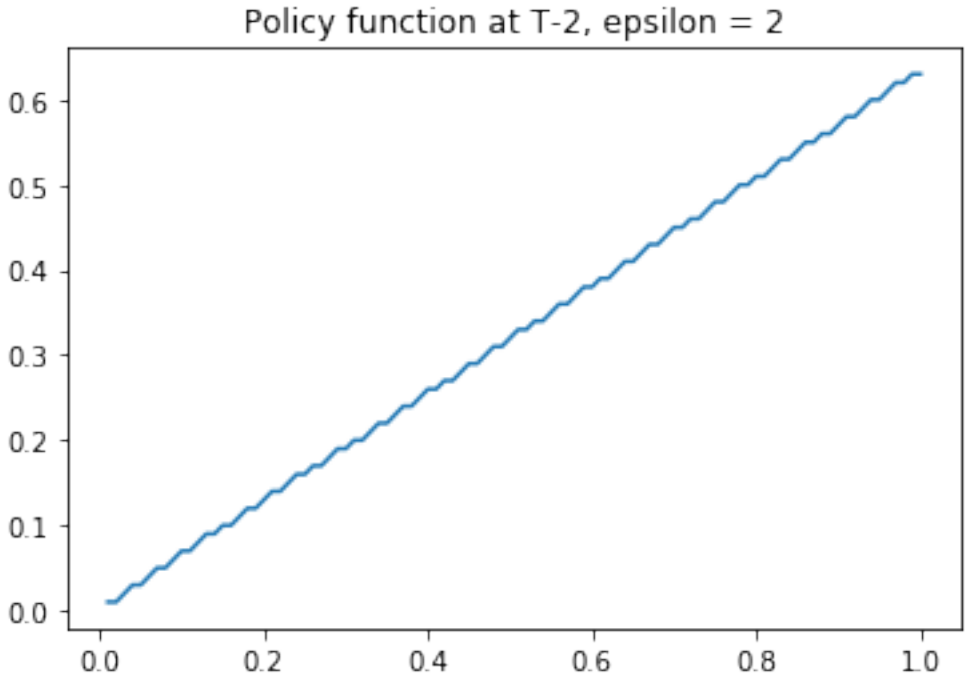
Again, we apply the functions above to find V_{T-2} and ψ_{T-2}; let us again plot them for $\epsilon = \mu = 2$.

```
In [955]: VeTm2, PeTm2 = vp_stoch(W_Wp, VeTm1, e_vec, gamma_vec)
```

```
In [958]: ## value at T-2, epsilon = 2
          plt.plot(W,VeTm2[3])
          plt.title("Value function at T-2, epsilon = 2")
          plt.show()
```



In [957]:
```
## policy at T-2, epsilon = 2
plt.plot(W,PeTm2[3])
plt.title("Policy function at T-2, epsilon = 2")
plt.show()
```



We find $\delta_{T-2} \approx 3.720 \times 10^{12}$, which is smaller than $\delta_{T-1} \approx 5.667 \times 10^{12}$ and smaller than $\delta_T \approx 5.671 \times 10^{12}$.

```
In [959]: d_Tm2 = norm_val_stoch(VeTm2, VeTm1)
          print(d_Tm2)
```

```
3.72046772907e+12
```

0.0.13 Exercise 5.21

Let us then use the function *dynprog_stoch* as described below to find whether the system converges or not, and how many iterations it requires if it does so.

```
In [963]: small = 1e-9

          def dynprog_stoch(W_Wp, VWe, evec, probs, beta=0.9):
              W_l = W_Wp.shape[0]
              Wp_l = W_Wp.shape[1]
              V_l = VWe.shape[1]

              err_msg = "Dimensions should be the same"
              assert W_l == Wp_l and Wp_l == V_l, err_msg

              s = 1
              inputVWe = VWe
              d = small + 1

              while d >= small:
                  opener = "Iteration "+str(s)+":"
                  newVWe, newPWe = vp_stoch(W_Wp, inputVWe, evec, probs)
                  d = norm_val_stoch(newVWe, inputVWe)
                  print(opener, "distance =", d)
                  inputVWe = newVWe
                  s += 1

              print("--------------------")
              print("Success at iteration "+str(s-1)+":", "distance =", d)

              return newVWe, newPWe
```

```
In [965]: val_cvg, pol_cvg = dynprog_stoch(W_Wp, VeTp1, e_vec, gamma_vec)
```

```
Iteration 1: distance = 5.67058027626e+12
Iteration 2: distance = 4.59317002971e+12
Iteration 3: distance = 3.72046772907e+12
Iteration 4: distance = 3.01357886362e+12
```

```
Iteration 5: distance = 2.44099888159e+12
Iteration 6: distance = 1.97720909554e+12
Iteration 7: distance = 1.60153936844e+12
Iteration 8: distance = 1.29724688922e+12
Iteration 9: distance = 1.05076998086e+12
Iteration 10: distance = 851123684949.0
Iteration 11: distance = 689410185163.0
Iteration 12: distance = 558422250263.0
Iteration 13: distance = 452322022938.0
Iteration 14: distance = 366380838763.0
Iteration 15: distance = 296768479550.0
Iteration 16: distance = 240382468563.0
Iteration 17: distance = 194709799644.0
Iteration 18: distance = 157714937806.0
Iteration 19: distance = 127749099705.0
Iteration 20: distance = 103476770835.0
Iteration 21: distance = 83816184443.5
Iteration 22: distance = 67891109461.0
Iteration 23: distance = 54991798721.0
Iteration 24: distance = 44543357018.2
Iteration 25: distance = 36080119235.9
Iteration 26: distance = 29224896629.6
Iteration 27: distance = 23672166316.2
Iteration 28: distance = 19174454760.1
Iteration 29: distance = 15531308397.8
Iteration 30: distance = 12580359842.5
Iteration 31: distance = 10190091511.0
Iteration 32: distance = 8253974161.05
Iteration 33: distance = 6685719106.21
Iteration 34: distance = 5415432510.55
Iteration 35: distance = 4386500366.94
Iteration 36: distance = 3553065329.59
Iteration 37: distance = 2877982948.41
Iteration 38: distance = 2331166218.81
Iteration 39: distance = 1888244667.06
Iteration 40: distance = 1529478209.44
Iteration 41: distance = 1238877378.12
Iteration 42: distance = 1003490704.18
Iteration 43: distance = 812827497.749
Iteration 44: distance = 658390300.053
Iteration 45: distance = 533296169.471
Iteration 46: distance = 431969923.291
Iteration 47: distance = 349895663.507
Iteration 48: distance = 283415512.73
Iteration 49: distance = 229566590.273
Iteration 50: distance = 185948962.778
Iteration 51: distance = 150618684.221
Iteration 52: distance = 122001158.324
```

```
Iteration 53: distance = 98820962.0927
Iteration 54: distance = 80045002.9038
Iteration 55: distance = 64836475.7324
Iteration 56: distance = 52517568.5008
Iteration 57: distance = 42539253.4301
Iteration 58: distance = 34456818.0175
Iteration 59: distance = 27910045.1313
Iteration 60: distance = 22607158.8921
Iteration 61: distance = 18311820.8398
Iteration 62: distance = 14832596.823
Iteration 63: distance = 12014425.1802
Iteration 64: distance = 9731705.95264
Iteration 65: distance = 7882703.17812
Iteration 66: distance = 6385010.72979
Iteration 67: distance = 5171879.64356
Iteration 68: distance = 4189243.25957
Iteration 69: distance = 3393307.57392
Iteration 70: distance = 2748599.44267
Iteration 71: distance = 2226385.62444
Iteration 72: distance = 1803392.20095
Iteration 73: distance = 1460767.29321
Iteration 74: distance = 1183240.86925
Iteration 75: distance = 958444.194177
Iteration 76: distance = 776358.599812
Iteration 77: distance = 628868.962839
Iteration 78: distance = 509402.046061
Iteration 79: distance = 412633.536675
Iteration 80: distance = 334250.712362
Iteration 81: distance = 270760.243009
Iteration 82: distance = 219332.553119
Iteration 83: distance = 177675.704799
Iteration 84: distance = 143933.199434
Iteration 85: distance = 116601.270732
Iteration 86: distance = 94461.8680706
Iteration 87: distance = 76528.3654502
Iteration 88: distance = 62001.5987535
Iteration 89: distance = 50234.1971895
Iteration 90: distance = 40701.8146503
Iteration 91: distance = 32979.6663073
Iteration 92: distance = 26723.7839804
Iteration 93: distance = 21655.3266153
Iteration 94: distance = 17548.5925371
Iteration 95: distance = 14220.5259173
Iteration 96: distance = 11522.8960781
Iteration 97: distance = 9335.3175162
Iteration 98: distance = 7560.27093771
Iteration 99: distance = 6118.28900608
Iteration 100: distance = 4939.0607865
```

```
Iteration 101: distance = 0.0
--------------------
Success at iteration 101: distance = 0.0
```

The above process shows that it took 101 iterations for convergence.

0.0.14 Exercise 5.22

Let us plot the 3-D graph as requested:

```
In [330]: import matplotlib.pyplot as plt
          from mpl_toolkits.mplot3d import Axes3D

In [966]: X = W
          Y = e_vec
          X, Y = np.meshgrid(X, Y)
          Z = pol_cvg

In [969]: fig = plt.figure()
          ax = fig.add_subplot(111, projection='3d')
          ax.plot_surface(Y, X, Z)
          ax.set_title("Converged policy function")
          ax.set_zlabel("Cake tomorrow")
          ax.set_ylabel("Cake today")
          ax.set_xlabel("Shock on utility")
          plt.show()
```