17 Dynamic Programming

Lab Objective: *Sequential decision making problems are a class of problems in which the current choice depends on future choices. They are a subset of Markov decision processes, an important class of problems with applications in business, robotics, and economics. Dynamic programming is a method of solving these problems that optimizes the solution by breaking the problem down into steps and optimizing the decision at each time period. In this lab we use dynamic programming to solve two classic dynamic optimization problems.*

The Marriage Problem

Many dynamic optimization problems can be classified as *optimal stopping* problems, where the goal is to determine at what time to take an action to maximize the expected reward. For example, when hiring a secretary, how many people should you interview before hiring the current interviewer? Or how many people should you date before you get married? These problems try to determine at what person t to stop in order to maximize the chance of getting the best candidate.

For instance, let N be the number of people you could date. After dating each person, you can either marry them or move on; you can't resume a relationship once it ends. In addition, you can rank your current relationship to all of the previous options, but not to future ones. The goal is to find the policy that maximizes the probability of choosing the best marriage partner. That policy may not always choose the best candidate, but it should get an almost-best candidate most of the time.

Let $V(t-1)$ be the probability that we choose the best partner when we have passed over the first $t-1$ candidates with an optimal policy. In other words, we have dated $t-1$ people and want to know the probability that the t^{th} person is the one we should marry. Note that the probability that the t^{th} person is not the best candidate is $\frac{t-1}{t}$ and the probability that they are is $\frac{1}{t}$. If the t^{th} person is not the best out of the first t, then probability they are the best overall is 0 and the probability they are not is $V(t)$. If the t^{th} person is the best out of the first t, then the probability they are the best overall is $\frac{t}{N}$ and the probability they are not is $V(t)$.

By Bellman's optimality equations,

$$V(t-1) = \frac{t-1}{t}\max\{0, V(t)\} + \frac{1}{t}\max\left\{\frac{t}{N}, V(t)\right\} = \max\left\{\frac{t-1}{t}V(t) + \frac{1}{N}, V(t)\right\}. \tag{17.1}$$

Notice that (17.1) implies that $V(t-1) \geq V(t)$ for all $t \leq N$. Hence, the probability of selecting the best match $V(t)$ is non-increasing. Conversely, $P(\text{t is best overall}|t \text{ is best out of the first } t) =$

$\frac{t}{N}$ is strictly increasing. Therefore, there is some t_0, called the *optimal stopping point*, such that $V(t) \leq \frac{t}{N}$ for all $t \geq t_0$. After t_0 relationships, we choose the next partner who is better than all of the previous ones. We can write (17.1) as

$$V(t-1) = \begin{cases} V(t_0) & t < t_0, \\ \frac{t-1}{t}V(t) + \frac{1}{N} & t \geq t_0. \end{cases}$$

The goal of an optimal stopping problem is to find t_0, which we can do by backwards induction. We start at the final candidate, who always has probability 0 of being the best overall if they are not the best so far, and work our way backwards, computing the expected value $V(t)$, for $t = N, N-1, \ldots, 1$.

If $N = 4$, we have

$$\begin{aligned} V(4) &= 0, \\ V(3) &= \max\left\{\frac{3}{4}V(4) + \frac{1}{4}, 0\right\} = .25, \\ V(2) &= \max\left\{\frac{2}{3}V(3) + \frac{1}{4}, .25\right\} = .4166, \\ V(1) &= \max\left\{\frac{1}{4}, .4166\right\} = .4166. \end{aligned}$$

In this case, the maximum expected value is .4166 and the stopping point is $t = 2$. It is also useful to look at the optimal stopping percentage of people to date before getting married. In this case, it is $2/4 = .5$.

Problem 1. Write a function that accepts a number of candidates N. Calculate the expected values of choosing candidate t for $t = 0, 1, \ldots, N-1$.

Return the highest expected value $V(t_0)$ and the optimal stopping point t_0.
(Hint: Since Python starts indices at 0, the first candidate is $t = 0$.)

Check your answer for $N = 4$ with the example detailed above.

Problem 2. Write a function that takes in an integer M and runs your function from Problem 1 for each $N = 3, 4, \ldots, M$. Graph the optimal stopping percentage of candidates (t_0/N) to interview and the maximum probability $V(t_0)$ against N. Return the optimal stopping percentage for M.

The optimal stopping percentage for $M = 1000$ is .367.

Both the stopping time and the probability of choosing the best person converge to $\frac{1}{e} \approx .36788$. Then to maximize the chance of having the best marriage, you should date at least $\frac{N}{e}$ people before choosing the next best person. This famous problem is also known as the *secretary problem*, the *sultan's dowry problem*, and the *best choice problem*. For more information, see `https://en.wikipedia.org/wiki/Secretary_problem`.

The Cake Eating Problem

Imagine you are given a cake. How do you eat it to maximize your enjoyment? Some people may prefer to eat all of their cake at once and not save any for later. Others may prefer to eat a little bit

at a time. If we are to consume a cake of size W over $T+1$ time periods, then our consumption at each step is represented as a vector

$$\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \cdots & c_T \end{bmatrix}^\mathsf{T},$$

where

$$\sum_{i=0}^{T} c_i = W.$$

This vector is called a *policy vector* and describes how much cake is eaten at each time period. The enjoyment of eating a slice of cake is represented by a utility function. For some amount of consumption $c_i \in [0, W]$, the utility gained is given by $u(c_i)$.

For this lab, we assume the utility function satisfies $u(0) = 0$, that $W = 1$, and that W is cut into N equally-sized pieces so that each c_i must be of the form $\frac{i}{N}$ for some integer $0 \leq i \leq N$.

Discount Factors

A person or firm typically has a time preference for saving or consuming. For example, a dollar today can be invested and yield interest, whereas a dollar received next year does not include the accrued interest. Since cake gets stale as it gets older, we assume that cake in the present yields more utility than cake in the future. We can model this by multiplying future utility by a discount factor $\beta \in (0, 1)$. For example, if we were to consume c_0 cake at time 0 and c_1 cake at time 1, with $c_0 = c_1$ then the utility gained at time 0 is larger than the utility at time 1:

$$u(c_0) > \beta u(c_1).$$

The total utility for eating the cake is

$$\sum_{t=0}^{T} \beta^t u(c_t).$$

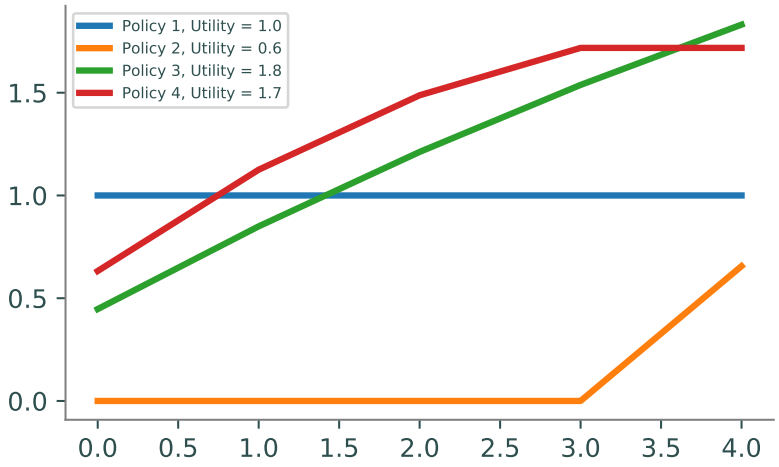

Figure 17.1: Plots for various policies with $u(x) = \sqrt{x}$ and $\beta = 0.9$. Policy 1 eats all of the cake in the first step while policy 2 eats all of the cake in the last step. Their difference in utility demonstrate the effect of the discount factor on waiting to eat. Policy 3 eats the same amount of cake at each step, while policy 4 begins by eating .4 of the cake, then .3, .2, and .1.

The Value Function

The cake eating problem is an optimization problem where we maximize utility.

$$\begin{aligned} \max_{\mathbf{c}} \sum_{t=0}^{T} \beta^t u(c_t) \\ \text{subject to } \sum_{t=0}^{T} c_t = W \\ c_t \geq 0. \end{aligned} \tag{17.2}$$

One way to solve it is with the value function. The value function $V(a, b, W)$ gives the utility gained from following an optimal policy from time a to time b.

$$\begin{aligned} V(a, b, W) = \max_{\mathbf{c}} \sum_{t=a}^{b} \beta^t u(c_t) \\ \text{subject to } \sum_{t=a}^{b} c_t = W \\ c_t \geq 0. \end{aligned}$$

$V(0, T, W)$ gives how much utility we gain in T days and is the same as Equation 17.2.

Let W_t represent the total amount of cake left at time t. Observe that $W_{t+1} \leq W_t$ for all t, because our problem does not allow for the creation of more cake. Notice that $V(t+1, T, W_{t+1})$ can be represented by $\beta V(t, T-1, W_{t+1})$, which is the value of eating W_{t+1} cake later. Then we can express the value function as the sum of the utility of eating $W_t - W_{t+1}$ cake now and W_{t+1} cake later.

$$V(t, T, W_t) = \max_{W_{t+1}} \left(u(W_t - W_{t+1}) + \beta V(t, T-1, W_{t+1}) \right) \tag{17.3}$$

where $u(W_t - W_{t+1})$ is the value gained from eating $W_t - W_{t+1}$ cake at time t.

Let $\mathbf{w} = \begin{bmatrix} 0 & \frac{1}{N} & \cdots & \frac{N-1}{N} & 1 \end{bmatrix}^\mathsf{T}$. We define the *consumption matrix* C by $C_{ij} = u(w_i - w_j)$. Note that C is an $(N+1) \times (N+1)$ lower triangular matrix since we assume $j \leq i$; we can't consume more cake than we have. The consumption matrix will help solve the value function by calculating all possible value of $u(W_t - W_{t+1})$ at once. At each time t, W_t can only have $N+1$ values, which will be represented as $w_i = \frac{i}{N}$, which is i pieces of cake remaining. For example, if $N = 4$, then $w = [0, .25, .5, .75, 1]^\mathsf{T}$, and $w_3 = 0.75$ represents having three pieces of cake left. In this case, we get the following consumption matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ u(0.25) & 0 & 0 & 0 & 0 \\ u(0.5) & u(0.25) & 0 & 0 & 0 \\ u(0.75) & u(0.5) & u(0.25) & 0 & 0 \\ u(1) & u(0.75) & u(0.5) & u(0.25) & 0 \end{bmatrix}.$$

Problem 3. Write a function that accepts the number of equal sized pieces N that divides the cake and a utility function $u(x)$. Assume $W = 1$. Create a partition vector $\mathbf{w}$ whose entries correspond to possible amounts of cake. Return the consumption matrix.

Solving the Optimization Problem

Initially we do not know how much cake to eat at $t = 0$: should we eat one piece of cake (w_1), or perhaps all of the cake (w_N)? It may not be obvious which option is best and that option may change depending on the discount factor β. Instead of asking how much cake to eat at some time t, we ask how valuable w_i cake is at time t. As mentioned above, $V(t, T-1, W_{t+1})$ in 17.3 is a new value function problem with $a = t, b = T-1$, and $W = W_{t+1}$, making 17.3 a recursion formula. By using the optimal value of the value function in the future, $V(t, T-1, W_{t+1})$, we can determine the optimal value for the present, $V(t, T, W_t)$. $V(t, T, W_t)$ can be solved by trying each possible W_{t+1} and choosing the one that gives the highest utility.

The $(N+1)\times(T+1)$ matrix A that solves the value function is called the *value function matrix*. A_{ij} is the value of having w_i cake at time j. $A_{0j} = 0$ because there is never any value in having w_0 cake, i.e. $u(w_0) = u(0) = 0$.

We start at the last time period. Since there is no value in having any cake left over when time runs out, the decision at time T is obvious: eat the rest of the cake. The amount of utility gained from having w_i cake at time T is given by $u(w_i)$. So $A_{iT} = u(w_i)$. Written in the form of (17.3),

$$A_{iT} = V(0, 0, w_i) = \max_{w_j} \left(u(w_i - w_j) + \beta V(0, -1, w_j)\right) = u(w_i). \tag{17.4}$$

This happens because $V(0, -1, w_j) = 0$. As mentioned, there is no value in saving cake so this equation is maximized when $w_j = 0$. All possible values of w_i are calculated so that the value of having w_i cake at time T is known.

Achtung!

Given a time interval from $t = 0$ to $t = T$ the utility of waiting until time T to eat w_i cake is actually $\beta^T u(W_i)$. However, through backwards induction, the problem is solved backwards by beginning with $t = T$ as an isolated state and calculating its value. This is why the value function above is $V(0, 0, W_i)$ and not $V(T, T, W_i)$.

For example, the following matrix results with $T = 3$, $N = 4$, and $\beta = 0.9$.

$$\begin{bmatrix} 0 & 0 & 0 & u(0) \\ 0 & 0 & 0 & u(0.25) \\ 0 & 0 & 0 & u(0.5) \\ 0 & 0 & 0 & u(0.75) \\ 0 & 0 & 0 & u(1) \end{bmatrix}.$$

Problem 4. Write a function that accepts a stopping time T, a number of equal sized pieces N that divides the cake, a discount factor β, and a utility function $u(x)$. Return the value function matrix A for $t = T$ (the matrix should have zeros everywhere except the last column). Return a matrix of zeros for the policy matrix P.

Next, we use the fact that $A_{jT} = V(0, 0, w_j)$ to evaluate the $T-1$ column of the value function matrix, $A_{i(T-1)}$, by modifying (17.4) as follows,

$$A_{i(T-1)} = V(0, 1, w_i) = \max_{w_j} \left(u(w_i - w_j) + \beta V(0, 0, w_j)\right) = \max_{w_j} \left(u(w_i - w_j) + \beta A_{jT}\right). \tag{17.5}$$

Remember that there is a limited set of possibilities for w_j, and we only need to consider options such that $w_j \leq w_i$. Instead of doing these one by one for each w_i, we can compute the options for each w_i simultaneously by creating a matrix. This information is stored in an $(N+1) \times (N+1)$ matrix known as the *current value matrix*, or CV^t, where the (ij)th entry is the value of eating $w_i - w_j$ pieces of cake at time t and saving j pieces of cake until the next period. For $t = T-1$,

$$CV_{ij}^{T-1} = u(w_i - w_j) + \beta A_{jT}. \tag{17.6}$$

The largest entry in the ith row of CV^{T-1} is the optimal value that the value function can attain at $T-1$, given that we start with w_i cake. The maximal values of each row of CV^{T-1} become the column of the value function matrix, A, at time $T-1$.

Achtung!

The notation CV^t does not mean raising the matrix to the tth power; rather, it indicates what time period we are in. All of the CV^t could be grouped together into a three-dimensional matrix, CV, that has dimensions $(N+1) \times (N+1) \times (T+1)$. Although this is possible, we will not use CV in this lab, and will instead only consider CV^t for any given time t.

The following matrix is CV^2 where $T = 3$, $\beta = .9$, $N = 4$, and $u(x) = \sqrt{x}$. The maximum value of each row, circled in red, is used in the 3^{rd} column of A. Remember that A's column index begins at 0, so the 3^{rd} column represents $j = 2$.

$$CV^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.45 & 0 & 0 & 0 \\ 0.707 & 0.95 & 0.636 & 0 & 0 \\ 0.866 & 1.157 & 1.136 & 0.779 & 0 \\ 1 & 1.316 & 1.343 & 1.279 & 0.9 \end{bmatrix}$$

Now that the column of A corresponding to $t = T-1$ has been calculated, we repeat the process for $T-2$ and so on until we have calculated each column of A. In summary, at each time step t, find CV^t and then set A_{it} as the maximum value of the ith row of CV^t. Generalizing (17.5) and (17.6) shows

$$CV_{ij}^t = u(w_i - w_j) + \beta A_{j(t+1)}. \qquad A_{it} = \max_j \left(CV_{ij}^t\right). \tag{17.7}$$

The full value function matrix corresponding to the example is below. The maximum value in the value function matrix is the maximum possible utility to be gained.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 0.95 & 0.95 & 0.95 & 0.707 \\ 1.355 & 1.355 & 1.157 & 0.866 \\ 1.7195 & 1.562 & 1.343 & 1 \end{bmatrix}.$$

Figure 17.2: The value function matrix where $T = 3$, $\beta = .9$, $N = 4$, and $u(x) = \sqrt{x}$. The bottom left entry indicates the highest utility that can be achieved is 1.7195.

Problem 5. Complete your function from Problem 4 so it returns the entire value function matrix. Starting from the next to last column, iterate backwards by

- calculating the current value matrix for time t using (17.7),
- finding the largest value in each row of the current value matrix, and
- filling in the corresponding column of A with these values.

(Hint: Use `axis` arguments.)

Solving for the Optimal Policy

With the value function matrix constructed, the optimization problem is solved in some sense. The value function matrix contains the maximum possible utility to be gained. However, it is not immediately apparent what policy should be followed by only inspecting the value function matrix A. The $(N+1) \times (T+1)$ policy matrix, P, is used to find the optimal policy. The (ij)th entry of the policy matrix indicates how much cake to eat at time j if we have i pieces of cake. Like A and CV, i and j begin at 0.

The last column of P is calculated similarly to last column of A. $P_{iT} = w_i$, because at time T we know that the remainder of the cake should be eaten. Recall that the column of A corresponding to t was calculated by the maximum values of CV^t. The column of P for time t is calculated by taking $w_i - w_j$, where j is the smallest index corresponding to the maximum value of CV^t,

$$P_{it} = w_i - w_j.$$

$$\text{where } j = \{\, min\{j\} \mid CV^t_{ij} \geq CV^t_{ik} \; \forall \, k \in [0, 1, \ldots, N] \,\}$$

Recall CV^2 in our example with $T = 3$, $\beta = .9$, $N = 4$, and $u(x) = \sqrt{x}$ above.

$$CV^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.45 & 0 & 0 & 0 \\ 0.707 & 0.95 & 0.636 & 0 & 0 \\ 0.866 & 1.157 & 1.136 & 0.779 & 0 \\ 1 & 1.316 & 1.343 & 1.279 & 0.9 \end{bmatrix}$$

To calculate P_{12}, we look at the second row ($i = 1$) in CV^2. The maximum, .5, occurs at CV^2_{10}, so $j = 0$ and $P_{12} = w_1 - w_0 = .25 - 0 = .25$. Similarly, $P_{42} = w_4 - w_2 = 1 - .5 = .5$. Continuing in this manner,

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.5; \\ 0.25 & 0.25 & 0.5 & 0.75 \\ 0.25 & 0.5 & 0.5; & 1. \end{bmatrix}$$

Given that the rows of P are the slices of cake available and the columns are the time intervals, we find the policy by starting in the bottom left corner, P_{N0}, where there are N slices of cake available and $t = 0$. This entries tells us what percentage of the N slices of cake we should eat. In the example, this entry is .25, telling us we should eat 1 slice of cake at $t = 0$. Thus, when $t = 1$ we have $N - 1$ slices of cake available, since we ate 1 slice of cake. We look at the entry at $P_{(N-1)1}$, which has value .25. So we eat 1 slice of cake at $t = 1$. We continue this pattern to find the optimal policy $\mathbf{c} = \begin{bmatrix} .25 & .25 & .25 & .25 \end{bmatrix}$.

Achtung!

The optimal policy will not always be a straight diagonal in the example above. For example, if the bottom left corner had value .5, then we should eat 2 pieces of cake instead of 1. Then the next entry we should evaluate would be $P_{(N-2)1}$ in order to determine the optimal policy.

To verify the optimal policy found with P, we can use the value function matrix A. By expanding the entires of A, we can see that the optimal policy does give the maximum value.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} \\ \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.5} \\ \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.25} & \sqrt{0.75} \\ \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} + \beta^3\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.5} & \sqrt{1} \end{bmatrix}$$

Problem 6. Modify your function from Problem 4 to determine the policy matrix. Initialize the matrix as zeros and fill it in starting from the last column at the same time that you calculate the value function matrix.
(Hint: You may find `np.argmax()` useful.)

Problem 7. Write a function `find_policy()` that will find the optimal policy for the stopping time T, a cake of size 1 split into N pieces, a discount factor β, and the utility function u.