Lesson 19. Solving Stochastic Dynamic Programs with Python

Overview

- Let's solve the stochastic dynamic program we formulated for the investment problem in Lesson 18.
- In this class, we will use a package called `stochasticdp` to set up and solve stochastic dynamic programs.
  - *Warning.* This is a package that I wrote. There may be some bugs.
  - *Note.* This package is publicly available. Please feel free to use it in the future for other things. The source code is on GitHub.

Installing `stochasticdp`

- To install `stochasticdp`, open a WinPython Command Prompt and type:

```
pip install stochasticdp
```

- To use `stochasticdp`, we must first import it. In `stochasticdp`, we only need the object `StochasticDP`, so we can perform our import like this:

```
In [2]: from stochasticdp import StochasticDP
```

Setting up a stochastic dynamic program

- Recall the investment problem from Lesson 18:

Problem. Suppose you have \$5,000 to invest. Over the next 3 years, you want to double your money. At the beginning of each of the next 3 years, you have an opportunity to invest in one of two investments: A or B. Both investments have uncertain profits. For an investment of \$5,000, the profits are as follows:

Investment	Profit (\$)	Probability
A	-5,000	0.3
	5,000	0.7
B	0	0.9
	5,000	0.1

You are allowed to make at most one investment each year, and can invest only \$5,000 each time. Any additional money accumulated is left idle. Once you've accumulated \$10,000, you stop investing.

Formulate a stochastic dynamic program to find an investment policy that maximizes the probability you will have \$10,000 after 3 years.

- Let's walk through setting up the stochastic DP we formulated in the last lesson.

Initialization

- We had defined 4 stages

- To make things easier, let's renumber the stages so they start at $t = 0$:

$$\begin{aligned} \text{stage } t = 0, 1, 2 \quad &\leftrightarrow \quad \text{beginning of year } t \\ t = 3 \quad &\leftrightarrow \quad \text{end of process} \end{aligned}$$

- In each stage, we defined 3 states:

$$\text{state } n \in \{0, 5000, 10000\} \quad \leftrightarrow \quad n \text{ dollars in account}$$

- At each stage and state, we defined 3 possible decisions:

$$\text{decision } x_t \in \{\text{A}, \text{B}, \text{no investment}\}$$

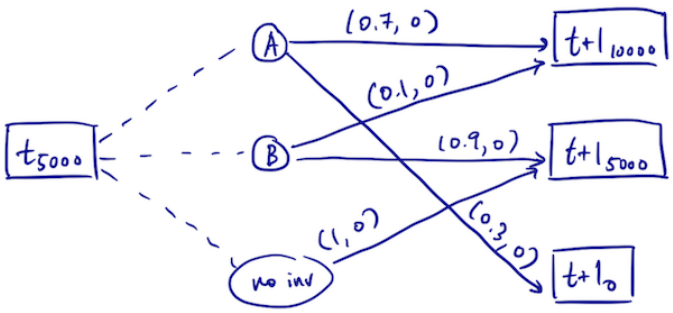
- The set of *allowable* decisions changed, depending on the stage and state. We'll address this later.
- For now, we can initialize a stochastic dynamic program with these stages, states, and decisions like this:

```
In [3]: # Number of stages
        number_of_stages = 4

        # List of states
        states = [0, 5000, 10000]

        # List of decisions
        decisions = ['A', 'B', 'no investment']

        # Initialize stochastic dynamic program - we want to maximize, so minimize = False
        dp = StochasticDP(number_of_stages, states, decisions, minimize=False)
```

- The code above initializes a stochastic dynamic program called `dp`.
- We need to add every transition that occurs with positive probability.

Transition probabilities and contributions

- First, let's tackle transitions from the state $n = 5000$:



- We can add a transition from state n to state m in stage t under decision x with probability $p(m \mid n, t, x)$ and contribution $c(m \mid n, t, x)$ as follows:

```
dp.add_transition(stage=t, from_state=n, decision=x, to_state=n, probability=p, contribution=c)
```

- So, we can input the transition probabilities and contributions from state $n = 5000$ in stages $t = 0, 1, 2$ as follows.
    - Remember that the contributions for all transitions are 0 in this stochastic DP.

```
In [4]: # Transition probabilities and contributions from state n = 5000
        for t in range(number_of_stages - 1):
            # Investment A
            dp.add_transition(stage=t, from_state=5000, decision='A', to_state=10000,
        probability=0.7, contribution=0)
            dp.add_transition(stage=t, from_state=5000, decision='A', to_state=0,
        probability=0.3, contribution=0)

            # Investment B
            dp.add_transition(stage=t, from_state=5000, decision='B', to_state=10000,
        probability=0.1, contribution=0)
            dp.add_transition(stage=t, from_state=5000, decision='B', to_state=5000,
        probability=0.9, contribution=0)

            # No investment
            dp.add_transition(stage=t, from_state=5000, decision='no investment',
        to_state=5000, probability=1, contribution = 0)
```

- *Quick check.* What can the sum of the transition probabilities from any decision node equal?

The transition probabilities from any decision node must add up to either 0 or 1. They will add up to 1 if the decision is allowable at that stage/state; otherwise they will add up to 0.

- Next, let's tackle the transitions from state $n = 0$:

- So, we can input the transition probabilities and contributions from state $n = 0$ in stages $t = 0, 1, 2$ like this:

```
In [5]: # Transition probabilities and contributions from state n = 0
        for t in range(number_of_stages - 1):
            # No investment
            dp.add_transition(stage=t, from_state=0, decision='no investment', to_state=0,
        probability=1, contribution = 0)
```

- We can tackle the transitions from state $n = 10000$ in an almost identical way:

```
In [6]: # Transition probabilities and contributions from state n = 10000
        for t in range(number_of_stages - 1):
            # No investment
            dp.add_transition(stage=t, from_state=10000, decision='no investment',
        to_state=10000, probability=1, contribution = 0)
```

Boundary conditions

- Finally, we need to define the boundary conditions.
- In particular, we need to specify the value-to-go function at the last stage (in our case, $t = 3$) for each state.
- We can add the boundary conditions for state n like this:

```
dp.add_boundary(state=0, value=0)
```

- So, let's add the boundary conditions for our problem:

```
In [7]: # Boundary conditions
        dp.add_boundary(state=10000, value=1)
        dp.add_boundary(state=5000, value=0)
        dp.add_boundary(state=0, value=0)
```

Solving the stochastic dynamic program

- Once the stochastic DP is setup, we can solve it like this:

```
In [8]: # Solve the stochastic dynamic program
        value, policy = dp.solve()
```

- Note that the method `.solve()` outputs two objects: `value` and `policy`.
- `value[t, n]` is the value-to-go function $f_t(n)$ at stage t and state n.
- `policy[t, n]` is the optimal decision x_t^* that attains the value-to-go function $f_t(n)$ at stage t and state n.
- First, let's see what the value-to-go function looks like:

```
In [9]: # Examine the value-to-go function
        value
```

```
Out[9]: {(stage: 0, state: 0): 0
         (stage: 0, state: 5000): 0.757
         (stage: 0, state: 10000): 1
         (stage: 1, state: 0): 0
         (stage: 1, state: 5000): 0.73
         (stage: 1, state: 10000): 1
         (stage: 2, state: 0): 0
         (stage: 2, state: 5000): 0.7
         (stage: 2, state: 10000): 1
         (stage: 3, state: 0): 0
         (stage: 3, state: 5000): 0
         (stage: 3, state: 10000): 1}
```

- Next, let's look at the corresponding policy:

```
In [10]: # Examine the policy
         policy
```

```
Out[10]: {(stage: 0, state: 0): {'no investment'}
          (stage: 0, state: 5000): {'B'}
          (stage: 0, state: 10000): {'no investment'}
          (stage: 1, state: 0): {'no investment'}
          (stage: 1, state: 5000): {'B'}
          (stage: 1, state: 10000): {'no investment'}
          (stage: 2, state: 0): {'no investment'}
          (stage: 2, state: 5000): {'A'}
          (stage: 2, state: 10000): {'no investment'}}
```

On your own

- Solve the stochastic DP we formulated in Lesson 17 for this problem:

Problem. The Hit-and-Miss Manufacturing Company has received an order to supply one item of a particular type. However, manufacturing this item is difficult, and the customer has specified such stringent quality requirements that the company may have to produce more than one item to obtain an item that is acceptable.

The company estimates that each item of this type will be acceptable with probability 1/2 and defective with probability 1/2. Each item costs \$100 to produce, and excess items are worthless. In addition, a setup cost of \$300 must be incurred whenever the production process is setup for this item. The company has time to make no more than 3 production runs, and at most 5 items can be produced in each run. If an acceptable item has not been obtained by the end of the third production run, the manufacturer is in breach of contract and must pay a penalty of \$1600.

The objective is to determine how many items to produce in each production run in order to minimize the total expected cost.

```
In [11]: # Number of stages
         number_of_stages = 4

         # List of states
         states = [0, 1]

         # List of decisions
         decisions = [0, 1, 2, 3, 4, 5]

         # Initialize stochastic dynamic program
         dp = StochasticDP(number_of_stages, states, decisions, minimize=True)

         # Transition probabilities and contributions from state n = 0
         for t in range(number_of_stages - 1):
             for x in decisions:
                 if x > 0:
                     K = 300
                 else:
                     K = 0

                 dp.add_transition(stage=t, from_state=0, decision=x, to_state=0,
         probability=1, contribution=K + 100*x)
                 dp.add_transition(stage=t, from_state=0, decision=x, to_state=1,
         probability=0, contribution=K + 100*x)

         # Transition probabilities and contributions from state n = 1
         for t in range(number_of_stages - 1):
             for x in decisions:
                 if x > 0:
                     K = 300
```

```
                else:
                    K = 0

                dp.add_transition(stage=t, from_state=1, decision=x, to_state=0,
        probability=1 - (1/2)**x, contribution=K + 100*x)
                dp.add_transition(stage=t, from_state=1, decision=x, to_state=1,
        probability=(1/2)**x, contribution=K + 100*x)

        # Boundary conditions
        dp.add_boundary(state=0, value=0)
        dp.add_boundary(state=1, value=1600)

        # Solve the stochastic dynamic program
        value, policy = dp.solve()
```

```
In [12]: # Examine value-to-go
         value
```

```
Out[12]: {(stage: 0, state: 0): 0
          (stage: 0, state: 1): 675.0
          (stage: 1, state: 0): 0
          (stage: 1, state: 1): 700.0
          (stage: 2, state: 0): 0
          (stage: 2, state: 1): 800.0
          (stage: 3, state: 0): 0
          (stage: 3, state: 1): 1600}
```

```
In [13]: # Examine policy
         policy
```

```
Out[13]: {(stage: 0, state: 0): {0}
          (stage: 0, state: 1): {2}
          (stage: 1, state: 0): {0}
          (stage: 1, state: 1): {2, 3}
          (stage: 2, state: 0): {0}
          (stage: 2, state: 1): {3, 4}}
```