Worked Examples and Solutions for the Book: Approximate Dynamic Programming: Solving the Curses of Dimensionality by Warren B. Powell

John L. Weatherwax*

October 23, 2020

*wax@alum.mit.edu

To my family.

Introduction

Here you'll find solutions to the problems that I wrote up as I worked through this excellent book. For some of the problems I used various computer languages (`MATLAB`, `python`, or `R`) to perform any needed calculations. The code snippets for various exercises can be found at the following location:

`http://www.waxworksmath.com/Authors/N_Z/Powell/powell.html`

As a warning to the reader, by working this book I'm trying to improve my understanding of approximate dynamic programming (ADP). ADP is a very powerful framework for solving all sorts of dynamic optimization problems. Unfortunately I'm not quite an expert enough "yet" to be able to answer every question in the book as well as I would have liked.

In these notes you might find full problems or parts of problems that I have not been able to do. This could be because I've not yet had time to work them or because I was not fully sure that what I would write would be correct or maybe because I didn't know how to solve them at all. In any case if you have a comment or a way to make these notes better please send it to me. I might be able to learn a bit more and expand on any comments provided. As this material stands there is a lot of material here and if you are looking to learn ADP using Powell's book I think these notes should help you considerably. One can always use these notes compare any proposed solution to what I present.

As always, please let me know of any errors that might exist. I'll gladly add to the acknowledgments in later printings the name of the first person to bring each problem to my attention.

Chapter 2 (Some Illustrative Models)

Problem 2.4 (gambling where the value of S^N is $\sqrt{S^N}$)

Following the section in the book entitled "the gambling problem" the value of ending with S^N dollars at the end of the Nth trial is $V^N(S^N) = \sqrt{S^N}$. Then stepping back to $n = N-1$ using Bellman's equation we have

$$\begin{aligned} V^{N-1}(S^{N-1}) &= \max_{0 \le a^N \le S^{N-1}} E\left\{V^N(S^{N-1} + a^N W^N - a^N(1-W^N))|S^{N-1}\right\} \\ &= \max_{0 \le a^N \le S^{N-1}} \left\{ p\sqrt{S^{N-1}+a^N} + (1-p)\sqrt{S^{N-1}-a^N} \right\} . \end{aligned}$$

We find the optimal a^N by differentiating and equating the result to zero and then solving for a^N. The equation we need to solve for a^N is then

$$\frac{p}{2\sqrt{S^{N-1}+a^N}} - \frac{(1-p)}{2\sqrt{S^{N-1}-a^N}} = 0 .$$

When we solve for a^N in the above we get

$$a^N = \left(\frac{2p-1}{2p^2-2p+1} \right) S^{N-1} .$$

Note that with this value of a^N we find that

$$\begin{aligned} S^{N-1} + a^N &= \left(\frac{2p^2-2p+1+2p-1}{2p^2-2p+1} \right) S^{N-1} = \left(\frac{2p^2}{2p^2-2p+1} \right) S^{N-1} \quad \text{and} \\ S^{N-1} - a^N &= \left(\frac{2(p-1)^2}{2p^2-2p+1} \right) S^{N-1} . \end{aligned}$$

When we use our optimal expression for a^N we find $V^{N-1}(S^{N-1})$ given by

$$\begin{aligned} V^{N-1}(S^{N-1}) &= p\left(\frac{\sqrt{2}p}{\sqrt{2p^2-2p+1}} \right)\sqrt{S^{N-1}} + (1-p)\left(\frac{\sqrt{2}(1-p)}{\sqrt{2p^2-2p+1}} \right)\sqrt{S^{N-1}} \\ &= \sqrt{2(1-2p+2p^2)}\sqrt{S^{N-1}} = K^{N-1}\sqrt{S^{N-1}} . \end{aligned}$$

Where in the above we have defined the coefficient K in the above expression. We would now want to compute the functional form of $V^{N-2}(S^{N-2})$ using the above expression for $V^{N-1}(S^{N-1})$ in the same way we computed the functional form for $V^{N-1}(S^{N-1})$. That is we would need to consider

$$\begin{aligned} V^{N-2}(S^{N-2}) &= \max_{0 \le a^{N-1} \le S^{N-2}} \left\{ pV^{N-1}(S^{N-2}+a^{N-1}) + (1-p)V^{N-1}(S^{N-2}-a^{N-1}) \right\} \\ &= \max_{0 \le a^{N-1} \le S^{N-2}} \left\{ pK\sqrt{S^{N-2}+a^{N-1}} + (1-p)K\sqrt{S^{N-2}-a^{N-1}} \right\} . \end{aligned}$$

Now the coefficient K does not affect the optimization in that it can be factored to the outside and we have the same problem that we had when we computed $V^{N-1}(S^{N-1})$. Thus we know the optimal value of a^{N-1} is given by

$$a^{N-1} = \left(\frac{2p-1}{2p^2-2p+1} \right) S^{N-2} ,$$

and the optimal functional form for $V^{N-2}(S^{N-2})$ given by

$$V^{N-2}(S^{N-2}) = K^2 \sqrt{S^{N-2}} .$$

In general for this problem the pattern then looks like

$$a^n = \left(\frac{2p-1}{2p^2-2p+1} \right) S^n \quad \text{for} \quad 1 \leq n \leq N ,$$

and given the expression for K is

$$V^n(S^n) = (2(1-2p+2p^2))^{\frac{N-n}{2}} \sqrt{S^n} .$$

Problem 2.5 (a shortest path algorithm)

An algorithm that works backwards from the destination node that is very similar to the Algorithm in Figure 2.2 is the following. We begin with our candidate list C only containing the terminal node r. Then at each step we will follow Dijkstra's trick of selecting the node from the candidate list to remove that has the smallest cost to go, v_i, and then update v_j for all nodes j that go into i i.e. $j \in I_i^-$. If a node has its cost-to-go value modified during a given step it is added to the candidate list if it is not already there. The iterations proceed until the candidate list is empty.

Problem 2.6 (an proof of the solution to the continuous budgeting problem)

To prove the given functional form

$$V_{T-t+1}(R_{T-t+1}) = t \sqrt{\frac{R_{T-t-1}}{t}} \quad \text{for} \quad 1 \leq t \leq T , \tag{1}$$

using mathematical induction for the solution to the continuous budgeting problem when the rewards are given as $C_t(x_t) = \sqrt{x_t}$ we will assume the above form is true for $1 \leq t \leq \hat{t}$ and derive that it is also true for $t = \hat{t}+1$. Note that we have shown this expression is valid when $t = 1$ and $t = 2$ so the initial conditions of our induction recursion are satisfied. From Bellman's equation we have that $V_{T-t-1}(R_{T-t+1})$ is given by

$$\begin{aligned} V_{T-t+1}(R_{T-t+1}) &= V_{T-\hat{t}}(R_{T-\hat{t}}) \\ &= \max_{x_{T-\hat{t}} \leq R_{T-\hat{t}}} \left(\sqrt{x_{T-\hat{t}}} + V_{T-\hat{t}+1}(R_{T-\hat{t}+1}(x_{T-\hat{t}})) \right) \\ &= \max_{x_{T-\hat{t}} \leq R_{T-\hat{t}}} \left(\sqrt{x_{T-\hat{t}}} + V_{T-\hat{t}+1}(R_{T-\hat{t}} - x_{T-\hat{t}}) \right) . \end{aligned}$$

We have used the fact that $R_{T-\hat{t}+1}(X_{T-\hat{t}}) = R_{T-\hat{t}} - x_{T-\hat{t}}$ since if we allocate an amount $x_{T-\hat{t}}$ to the task at stage $T-\hat{t}$ from a total of $R_{T-\hat{t}}$ we will have

$$R_{T-\hat{t}} - x_{T-\hat{t}}\,,$$

remaining for possible allocation to the later tasks. By the induction hypothesis we know the functional form of $V_{T-\hat{t}+1}(\cdot)$ we see that the above is given by

$$V_{T-\hat{t}}(R_{T-\hat{t}}) = \max_{x_{T-\hat{t}} \le R_{T-\hat{t}}} \left(\sqrt{x_{T-\hat{t}}} + \hat{t}\sqrt{\frac{R_{T-\hat{t}} - x_{T-\hat{t}}}{\hat{t}}} \right) .$$

To solve for the maximum in this expression we take the derivative of the above, set it equal to zero and solve for $x_{T-\hat{t}}$. We find for the derivative

$$\frac{1}{2\sqrt{x_{T-\hat{t}}}} - \left(\frac{\hat{t}}{\sqrt{\hat{t}}}\right)\left(\frac{1}{2}\right)\frac{1}{\sqrt{R_{T-\hat{t}} - x_{T-\hat{t}}}} = 0\,,$$

or

$$\left(\frac{\hat{t}}{\sqrt{\hat{t}}}\right)\frac{1}{\sqrt{R_{T-\hat{t}} - x_{T-\hat{t}}}} = \frac{1}{\sqrt{x_{T-\hat{t}}}}\,,$$

or by squaring both sides of the above and solving for $x_{T-\hat{t}}$ we find that

$$\hat{x}_{T-\hat{t}} = \frac{R_{T-\hat{t}}}{\hat{t}+1}\,.$$

Using this value the functional form of $V_{T-\hat{t}}(\cdot)$ then becomes

$$V_{T-\hat{t}}(R_{T-\hat{t}}) = \sqrt{\frac{R_{T-\hat{t}}}{\hat{t}+1}} + \left(\frac{\hat{t}}{\sqrt{\hat{t}}}\right)\sqrt{R_{T-\hat{t}} - \frac{R_{T-\hat{t}}}{\hat{t}+1}} = (1+\hat{t})\sqrt{\frac{R_{T-\hat{t}}}{1+\hat{t}}}\,.$$

This is the claimed functional form given in Equation 1 when $t = \hat{t}+1$. Thus the given expression is correct.

Problem 2.7 (what if $C_t(x_t) = c_t\sqrt{x_t}$)

We are told now to assume that the reward for allocating x_t during the tth stage is given by $C_t(x_t) = c_t\sqrt{x_t}$, or that we have a possibly time dependent multiplier c_t of $\sqrt{x_t}$. To find the cost-to-go, V, following the discussion on the continuous budgeting problem we start by computing $V_T(R_T)$

$$V_T(R_T) = \max_{x_T \le R_T} c_T\sqrt{x_T}\,,$$

which has a solution $x_T = R_T$ giving $V_T(R_T) = c_T\sqrt{R_T}$. Then the value of having an amount R_{T-1} to allocate at stage $T-1$, $V_{T-1}(R_{T-1})$, is obtained from Bellman's equation

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \max_{x_{T-1} \le R_{T-1}} \left(c_{T-1}\sqrt{x_{T-1}} + V_T(R_T(x_{T-1}))\right) \\ &= \max_{x_{T-1} \le R_{T-1}} \left(c_{T-1}\sqrt{x_{T-1}} + c_T\sqrt{R_{T-1} - x_{T-1}}\right) . \end{aligned}$$

To maximize this requires setting its derivative to zero. We find this given by

$$\frac{c_{T-1}}{2\sqrt{x_{T-1}}} - \frac{c_T}{2\sqrt{R_{T-1} - x_{T-1}}} = 0\,.$$

When we solve for x_{T-1} in this equation we find $x_{T-1} = \frac{c_{T-1}^2 R_{T-1}}{c_{T-1}^2 + c_T^2}$. Thus $V_{T-1}(\cdot)$ looks like

$$\begin{aligned} V_{T-1}(R_{T-1}) &= c_{T-1}\sqrt{\frac{c_{T-1}^2 R_{T-1}}{c_{T-1}^2 + c_T^2}} + c_T\sqrt{\frac{c_T^2 R_{T-1}}{c_{T-1}^2 + c_T^2}} \\ &= (c_{T-1}^2 + c_T^2)\sqrt{\frac{R_{T-1}}{c_{T-1}^2 + c_T^2}}\,. \end{aligned}$$

Based on this expression we hypothesis that at the stage $T-i+1$, the value of having R_{T-i+1} to allocate to this stage and all subsequent stages is given by

$$V_{T-i+1}(R_{T-i+1}) = S_{T-i+1}\sqrt{\frac{R_{T-i+1}}{S_{T-i+1}}}\,, \tag{2}$$

with S_{T-i+1} defined as the sum of squared rewards up from stage T down to stage $T-i+1$ or

$$S_{T-i+1} = c_{T-i+1}^2 + c_{T-i+2}^2 + \cdots + c_{T-1}^2 + c_T^2\,.$$

We have shown that Equation 2 is valid for $i = 1$ and $i = 2$. To show it is more general than that consider the Bellman equation for the previous stage. That is the value function $V_{T-i}(R_{T-i})$. We have

$$\begin{aligned} V_{T-i}(R_{T-i}) &= \max_{x_{T-i} \le R_{T-i}} \left(c_{T-i}\sqrt{x_{T-i}} + V_{T-i+1}(R_{T-i} - x_{T-i})\right) \\ &= \max_{x_{T-i} \le R_{T-i}} \left(c_{T-i}\sqrt{x_{T-i}} + S_{T-i+1}\sqrt{\frac{R_{T-i} - x_{T-i}}{S_{T-i+1}}}\right) \\ &= \max_{x_{T-i} \le R_{T-i}} \left(c_{T-i}\sqrt{x_{T-i}} + \sqrt{S_{T-i+1}}\sqrt{R_{T-i} - x_{T-i}}\right)\,. \end{aligned}$$

To maximize this expression we take the derivative and set it equal to zero. Doing so gives

$$\frac{c_{T-i}}{2\sqrt{x_{T-i}}} - \frac{\sqrt{S_{T-i+1}}}{2\sqrt{R_{T-i} - x_{T-i}}} = 0\,.$$

Solving for x_{T-i} we find

$$x_{T-i} = \frac{c_{T-i}^2 R_{T-i}}{c_{T-i}^2 + S_{T-i+1}} = \frac{c_{T-i}^2 R_{T-i}}{S_{T-i}}\,.$$

When we put this expression back into Bellman's equation to compute $V_{T-i}(\cdot)$ we find

$$V_{T-i}(R_{T-i}) = c_{T-i}\sqrt{\frac{c_{T-i}^2 R_{T-i}}{S_{T-i}}} + \sqrt{S_{T-i+1}}\sqrt{R_{T-i} - \frac{c_{T-i}^2 R_{T-i}}{S_{T-i}}}\,.$$

Since

$$1 - \frac{c_{T-i}^2}{S_{T-i}} = \frac{S_{T-i+1}}{S_{T-i}}\,,$$

the above expression for V_{T-i} becomes

$$\begin{aligned} V_{T-i}(R_{T-i}) &= c_{T-i}\sqrt{\frac{c_{T-i}^2 R_{T-i}}{S_{T-i}}} + \sqrt{S_{T-i+1}}\sqrt{\frac{S_{T-i+1}R_{T-i}}{S_{T-i}}} \\ &= (c_{T-i}^2 + S_{T-i+1})\sqrt{\frac{R_{T-i}}{S_{T-i}}} = S_{T-i}\sqrt{\frac{R_{T-i}}{S_{T-i}}}\,, \end{aligned}$$

which is the claimed functional form for V_{T-i}.

Problem 2.8 (what if $C_t(x_t) = \ln(x_t)$)

We now assume that our reward at stage t is given by $C_t(x_t) = \ln(x_t)$. Working backwards from the terminal decision at stage T we

$$V_T(R_T) = \max_{x_T \le R_T} \ln(x_T)\,,$$

which again has a solution $x_T = R_T$ so that $V_T(R_T) = \ln(R_T)$. Using Bellman's equation the cost-to-go, $V_{T-1}(R_{T-1})$, at the previous stage $T-1$ must satisfy

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \max_{x_{T-1} \le R_{T-1}} (\ln(x_{T-1}) + V_T(R_T(x_{T-1}))) \\ &= \max_{x_{T-1} \le R_{T-1}} (\ln(x_{T-1}) + \ln(R_{T-1} - x_{T-1}))\,. \end{aligned}$$

Taking the derivative we see that to maximize this we need to solve

$$\frac{1}{x_{T-1}} - \frac{1}{R_{T-1} - x_{T-1}} = 0\,,$$

for x_{T-1}. We find $x_{T-1} = \frac{R_{T-1}}{2}$ so that

$$V_{T-1}(R_{T-1}) = \ln(\frac{R_{T-1}}{2}) + \ln(\frac{R_{T-1}}{2}) = 2\ln(\frac{R_{T-1}}{2})\,.$$

If we try to jump to the general case we might hypothesis that the functional form of $V_{T-t+1}(\cdot)$ is given by

$$V_{T-t+1}(R_{T-t+1}) = t \ln\left(\frac{R_{T-t+1}}{t}\right)\,. \tag{3}$$

We have in fact shown that this expression is true when $t = 1$ and $t = 2$. To show that it is true in general we use Bellman's equation for the stage prior to this one given by

$$\begin{aligned} V_{T-t}(R_{T-t}) &= \max_{x_{T-t} \le R_{T-t}} (\ln(x_{T-t}) + V_{T-t+1}(R_{T-t} - x_{T-t})) \\ &= \max_{x_{T-t} \le R_{T-t}} \left(\ln(x_{T-t}) + t\ln\left(\frac{R_{T-t} - x_{T-t}}{t}\right)\right)\,. \end{aligned}$$

Taking the derivative of this and setting it equal to zero gives

$$\frac{1}{x_{T-t}} - \frac{t}{R_{T-t} - x_{T-t}} = 0\,.$$

Solving for x_{T-t} we find $x_{T-t} = \frac{R_{T-t}}{t+1}$, so that

$$\begin{aligned} V_{T-t}(R_{T-t}) &= \ln\left(\frac{R_{T-t}}{t+1}\right) + t\ln\left(\frac{1}{t}\left(R_{T-t} - \frac{R_{T-t}}{t+1}\right)\right) \\ &= (1+t)\ln(\frac{R_{T-t}}{t+1})\,, \end{aligned}$$

proving the stated functional form.

Problem 2.9 (a general reward function)

In this problem we are told only that our reward function $C_t(x_t)$ is continuously differential, monotonically increasing, and convex (like $\sqrt{x_t}$). Following the general dynamic programming prescription, the value of having R_T left at the final stage T is given by

$$V_T(R_T) = \max_{x_T \le R_T} C_T(x_T)\,.$$

Since C_T is monotonically increasing, the solution to this maximum is given by $x_T = R_T$ which results in $V_T(R_T) = C_T(R_T)$. Then at the previous stage

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \max_{x_{T-1} \le R_{T-1}} (C_{T-1}(x_{T-1}) + V_T(R_{T-1} - x_{T-1})) \\ &= \max_{x_{T-1} \le R_{T-1}} (C_{T-1}(x_{T-1}) + C_T(R_{T-1} - x_{T-1}))\,. \end{aligned}$$

If we assume that the reward function C_t does not depend on the stage i.e. has no actual t dependence, the above becomes

$$V_{T-1}(R_{T-1}) = \max_{x_{T-1} \le R_{T-1}} (C(x_{T-1}) + C(R_{T-1} - x_{T-1}))\,.$$

The maximum of the above expression should occur where the derivative vanishes. Since C is differentiable we can take the derivative of the argument of the maximization above to obtain

$$C'(x_{T-1}) - C'(R_{T-1} - x_{T-1}) = 0 \quad \text{or} \quad C'(x_{T-1}) = C'(R_{T-1} - x_{T-1})\,.$$

This would need to be solved for x_{T-1}. Since C is convex we have $C' > 0$ and thus C' has an inverse. Thus the solution to the above equation is

$$x_{T-1} = R_{T-1} - x_{T-1} \quad \text{so} \quad x_{T-1} = \frac{R_{T-1}}{2}\,.$$

Using this we find that

$$V_{T-1}(R_{T-1}) = C\left(\frac{R_{T-1}}{2}\right) + C\left(\frac{R_{T-1}}{2}\right) = 2C\left(\frac{R_{T-1}}{2}\right)\,.$$

At the next stage we need to solve

$$\begin{aligned} V_{T-2}(R_{T-2}) &= \max_{x_{T-2} \le R_{T-2}} \left(C(x_{T-2}) + V_{T-1}(R_{T-2} - x_{T-2}) \right) \\ &= \max_{x_{T-2} \le R_{T-2}} \left(C(x_{T-2}) + 2C\left(\frac{R_{T-2} - x_{T-2}}{2} \right) \right) . \end{aligned}$$

Taking the derivative and setting the result equal to zero gives

$$C'(x_{T-2}) + 2C'\left(\frac{R_{T-2} - x_{T-2}}{2} \right) \left(-\frac{1}{2} \right) = 0 .$$

Using the inverse of C' we can write this as

$$x_{T-2} = \frac{R_{T-2} - x_{T-2}}{2} \quad \text{so} \quad x_{T-2} = \frac{R_{T-2}}{3} .$$

With this value of x_{T-2} we get

$$V_{T-2}(R_{T-2}) = C\left(\frac{R_{T-2}}{3} \right) + 2C\left(\frac{(2/3)R_{T-2}}{2} \right) = 3C\left(\frac{R_{T-2}}{3} \right) .$$

At this point we are starting to see a pattern and it looks like equation 2.13 holds

$$V_{T-t+1}(R_{T-t+1}) = tC\left(\frac{R_{t-T+1}}{t} \right) \quad \text{for} \quad 1 \le t \le T . \tag{4}$$

We could prove this is the correct form by induction.

Problem 2.10 (budget allocation with $C_t(x_t) = x_t^2$)

Following the book (namely Section 2.1.2 on Page 29) when $C_t(x_t) = x_t^2$ we have

$$V_T(R_T) = \max_{0 \le a_T \le R_T} a_T^2 \quad \text{so} \quad V_T(R_T) = R_T^2 .$$

Working backwards using the Bellman equation we would have

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \max_{0 \le a_{T-1} \le R_{T-1}} \left(a_{T-1}^2 + V_T(R_{T-1} - a_{T-1}) \right) \\ &= \max_{0 \le a_{T-1} \le R_{T-1}} \left(a_{T-1}^2 + (R_{T-1} - a_{T-1})^2 \right) . \end{aligned}$$

To find this maximum we take the derivative of the above expression with respect to a_{T-1}, set the result equal to zero, and solve that equation for for a_{T-1}. The derivative is given by

$$2a_{T-1} + 2(R_{T-1} - a_{T-1})(-1) = 0 .$$

Solving for a_{T-1} we get $a_{T-1} = \frac{R_{T-1}}{2}$. Note that the second derivative of the expression we seek to maximize is given by

$$2 + 2(-1)(-1) = 4 > 0 ,$$

showing that the solution $\frac{R_{T-1}}{2}$ found is in fact a minimum and not a maximum. Thus the maximum must occur at the end points of the domain i.e. at either $a_{T-1} = 0$ or $a_{T-1} = R_{T-1}$. Note that if we take $a_{T-1} = 0$ the value of the expression R_{T-1}^2. If we take $a_{T-1} = R_{T-1}$ we also get R_{T-1}^2. Notice that in both cases the functional form for $V_{T-1}(R_{T-1})$ is a quadratic. Thus to achieve a maximum we should take $a_{T-1} = 0$ or $a_{T-1} = R_{T-1}$. If we take $a_{T-1} = 0$ we are saying that to make no allocations now and make all allocations at $t = T$. If we take $a_{T-1} = R_{T-1}$ we are saying to make all allocations now and none later. Either of these two solutions is optimal. What we see here is that *both* V_t and C_t are of the same functional form (a quadratic) and thus there really is no difference in cost between allocations now or allocations latter.

Problem 2.12 (allocating among products)

Following the general budgeting problem presented in the section of the book entitled "The Discrete Budgeting Problem" we seek to find allocation amounts a_t that at each stage satisfy

$$V_t(R_t) = \max_{0 \le a_t \le R_t} \left(C_t(a_t) + V_{t+1}(R_t - a_t) \right) . \tag{5}$$

In this problem we have only five stages so $T = 5$. We can assume that there is no value to having additional dollars once all stages have passed. Thus we assume that $V_{T+1}(R) = V_6(R) = 0$ for all R. We can solve this problem by working backwards starting at $T = 5$. This procedure is equivalent to using the above equation to express the value of $V_t(R_t)$ in a recursive formulation. In the `python` code `chap_2_prob_12.py` we implement a recursive algorithm to solve this problem. Namely we start with $t = 1$ and recursively iterate the above. When we run that code we get the output given by

```
allocation a_t for stage t=  1 is a_t=  9
allocation a_t for stage t=  2 is a_t=  0
allocation a_t for stage t=  3 is a_t=  9
allocation a_t for stage t=  4 is a_t=  0
allocation a_t for stage t=  5 is a_t=  9
```

We see that the above algorithm uses $9 + 9 + 9 = 27$ total dollars and gets a value of

$$3(12) + 1(0) + 4(12) + 2(0) + 5(12) = 144 .$$

We are left with 3 dollars unspent. Note that this is less than the minimal amount under which $f(x_t)$ gives a nonzero value (which is 5).

Problem 2.13 (how much time to spend on each course)

Let our state, R_t, be the amount of time remaining to complete any remaining tasks. Let a_t be the amount of time allocated to task t, then from the problem statement, our rewards as

a function of time are given by

$$C_1(a_1) = 100\sqrt{\frac{a_1}{20}} \quad \text{for} \quad 0 \le a_1 \le 20$$
$$C_2(a_2) = 100\sqrt{\frac{a_2}{15}} \quad \text{for} \quad 0 \le a_2 \le 15$$
$$C_3(a_3) = 100\sqrt{\frac{a_3}{10}} \quad \text{for} \quad 0 \le a_3 \le 10\,.$$

Lets denote the amount of time needed for each task to get a 100 on the project as B_t, thus $B_1 = 20$, $B_2 = 15$, and $B_3 = 10$. Let $V_t(R_t)$ be the value of having R_t time remaining to spend on the tasks $t, t+1, t+2, \ldots, T-1, T$ remaining projects. Given the above definition of our state our transition function is given by

$$R_{t+1} = R_t - a_t \quad \text{for} \quad 1 \le t \le 3\,.$$

We expect that there is no value in having extra time to allocate to nonexistent tasks and thus we will take $V_4(R_4) = 0$ for all values of R_4. We step backwards in time using Bellman's equation given by

$$V_t(R_t) = \max_{5 \le a_t \le \min(B_t, R_t)} \left(C_t(a_t) + V_{t+1}(R_t - a_t)\right)\,.$$

Notice that the domain of our optimization problem is multiples of five (starting at five) and up to the smaller of B_t (the bound needed to get 100 on the project) and R_t the maximum allowed time we have to allocate. In addition, while we don't explicitly state this in our mathematical notation, the problem requires that the time allocation a_t be a multiple of five. We will however enforce this criterion as we find the optimal values for a_t. Given the above set up, the goal of this problem is to evaluate $V_1(30)$ and the optimal policy that results in determining this value. Using Bellman's equation with $t = T - 1 = 4 - 1 = 3$ we find

$$\begin{aligned} V_3(R_3) &= \max_{5 \le a_3 \le \min(10, R_3)} \left(100\sqrt{\frac{a_3}{10}} + V_4(R_3 - a_3)\right) \\ &= \max_{5 \le a_3 \le \min(10, R_3)} \left(100\sqrt{\frac{a_3}{10}}\right) = 100\sqrt{\frac{\min(10, R_3)}{10}}\,. \end{aligned}$$

This results states that whatever time remains for the third task you should use all of it on the last task (which makes sense). For $V_2(R_2)$ we compute

$$\begin{aligned} V_2(R_2) &= \max_{5 \le a_2 \le \min(15, R_2)} \left(100\sqrt{\frac{a_2}{15}} + V_3(R_2 - a_2)\right) \\ &= \max_{5 \le a_2 \le \min(15, R_2)} \left(100\sqrt{\frac{a_2}{15}} + 100\sqrt{\frac{\min(10, R_2 - a_2)}{10}}\right)\,. \end{aligned}$$

If we didn't have the $\min(10, R_2 - a_2)$ in the above expression to find the above maximum of the above we could take the derivative of the above expression, set the result equal to

zero, and solve for a_2. Rather than solve analytically we will write down the expression for $V_1(R_1)$ and then solve this problem numerically. For $V_1(R_1)$ we have

$$V_1(R_1) = \max_{5 \leq a_1 \leq \min(20, R_1)\}} \left(100\sqrt{\frac{a_1}{20}} + V_2(R_1 - a_1) \right) ,$$

with the function V_2 given by its own expression above.

In the `python` code `chap_2_prob_13.py` we implement a recursive algorithm to solve this problem. When we run this code we get the solution

```
allocation a_t for stage t=  1 is a_t=  5
allocation a_t for stage t=  2 is a_t= 15
allocation a_t for stage t=  3 is a_t= 10
```

What is interesting about this solution is that rather than spend much time on project one (which has the longest completion requirements of 20 hours) we do a relatively poor job on it so that we have time to work on the other two projects. On these two we do the full amount of work so that we get 100% on them.

Problem 2.16 (questionnaires to N population segments)

We can consider this a N stage decision problem where the action at each stage to decide the number x_i of questionnaires to allocate and the cost of this allocation is given by $\frac{w_i}{\sqrt{x_i}}$. The state R_i is the amount of budget remaining to satisfy subsequent demand. Our dynamic programming problem them becomes to find the value function $V_i(R_i)$ given by

$$V_i(R_i) = \min_{x_i}(C_i(x_i) + V_{i+1}(R_i - x_i)) = \min_{0 \leq x_i \leq R_i} \left(\frac{w_i}{\sqrt{x_i}} + V_{i+1}(R_i - x_i) \right) ,$$

ending with the value function expressing the fact that having unused questionnaires is of no value or $V_{N+1}(R) = 0$.

Chapter 3 (Introduction to Markov Decision Processes)

Problem 3.1 (a classical inventory problem)

Part (a): At the end of period t we have R_t inventory and can place an order to get an additional amount a_t. This total $R_t + a_t$ will then be reduced by the random demand $\hat{D}_{t+1}$ that happens over the range $(t, t+1)$. Thus our state update equation is given by

$$R_{t+1} = \max(0, R_t + a_t - \hat{D}_{t+1}) .$$

Part (b): We would have

$$P(R_{t+1} = j | R_t = i, A^\pi = a_t) = \begin{cases} \sum_{d=R_t+a_t}^{\infty} P(D = d) & j = 0 \\ P(D = R_t + a_t - j) & 1 \leq j \leq R_t + a_t \\ 0 & j > R_t + a_t \end{cases} .$$

The first statement (where $j = 0$) is due to all demands between $(t, t+1)$ that are greater than the current inventory $R_t + a_t$. The second statements are for demands that are not as large as the inventory $R_t + a_t$. For example if the demand is one less than the inventory i.e. $\hat{D}_{t+1} = R_t + a_t - 1$ then we end with an inventory of 1. This happens with a probability $P(D = R_t + a_t - 1)$. The final statement is the statement that there is no probability of getting a value for R_{t+1} greater than the inventory $R_t + a_t$ since the demand is always nonnegative.

Problem 3.2 (inventory with a policy)

The idea of this problem is to introduce the idea that if our inventory is above the value of q we don't order any additional inventory. If the inventory falls below q we order $Q - R_t$ so that we have Q inventory on hand. Thus our state transition function is

$$R_t = \begin{cases} \max(0, Q - \hat{D}_t) & R_{t-1} < q \text{ we order } Q - R_{t-1} \\ \max(0, R_{t-1} - \hat{D}_t) & R_{t-1} \geq q \text{ we order nothing} . \end{cases}$$

We next want to compute $P(R_t = j | R_{t-1} = i, A^\pi = a_t)$. We first assume that $R_{t-1} = i \geq q$ and thus according to our policy we will not order any additional inventory. In this case our state $R_t = j$ can take on different values given by

- $R_t = 0$ if $\hat{D}_t \geq i$ which happens with a probability $\sum_{d=R_{t-1}}^{\infty} P(D = d)$.
- $R_t = 1$ if $\hat{D}_t = i - 1$ which happens with a probability $P(D = i - 1)$.
- $R_t = 2$ if $\hat{D}_t = i - 2$ which happens with a probability $P(D = i - 2)$.
- pattern continues

- $R_t = j$ if $\hat{D}_t = 0$ which happens with a probability $P(D = 0)$.
- $R_t > j$ which happens with a probability 0.

The general rule in the case where $R_{t-1} \geq q$ can be written as

$$P(R_t = j | R_{t-1} = i \geq q, A^\pi = a_t = 0) = \begin{cases} \sum_{d=R_{t-1}}^{\infty} & j = 0 \\ P(D = R_{t-1} - j) & 1 \leq j \leq R_{t-1} \\ 0 & j > R_{t-1} \end{cases} .$$

Next we assume that $R_{t-1} = i < q$ so that we order $a_t = Q - R_{t-1}$ to bring our inventory up to Q. In this case our state $R_t = j$ can take on different values given by

- $R_t = 0$ if $\hat{D}_t \geq Q$ which happens with a probability $\sum_{d=Q}^{\infty} P(D = d)$.
- $R_t = 1$ if $\hat{D}_t = Q - 1$ which happens with a probability $P(D = Q - 1)$.
- $R_t = 2$ if $\hat{D}_t = Q - 2$ which happens with a probability $P(D = Q - 2)$.
- pattern continues
- $R_t = Q$ if $\hat{D}_t = 0$ which happens with a probability $P(D = 0)$.
- $R_t > Q$ which happens with a probability 0.

The general rule in the case where $R_{t-1} < q$ can be written as

$$P(R_t = j | R_{t-1} = i < q, A^\pi = a_t = Q - R_{t-1}) = \begin{cases} \sum_{d=Q}^{\infty} & j = 0 \\ P(D = Q - j) & 1 \leq j \leq Q \\ 0 & j > Q \end{cases} .$$

Note that our transition probability p_{ij}^π explicitly depends on the state we are in (i.e. R_{t-1}) since that defines the action we will then take.

Problem 3.3 (value iteration)

Note that in this problem the contribution C is now of the form $C(s, a, s')$ and depends on the state s' we end up at. Thus we need to move the expectation over s' so that it includes this contribution term. Dropping the action variable a since this problem does not have an action variable. We then get for the value in each state s the following

$$\begin{aligned} v^n(s) &= \sum_{s' \in S} \mathbb{P}(s'|s) C(s, s') + \gamma \sum_{s' \in S} \mathbb{P}(s'|s) v^{n-1}(s') \\ &= \sum_{s' \in S} \mathbb{P}(s'|s) (C(s, s') + \gamma v^{n-1}(s')) . \end{aligned}$$

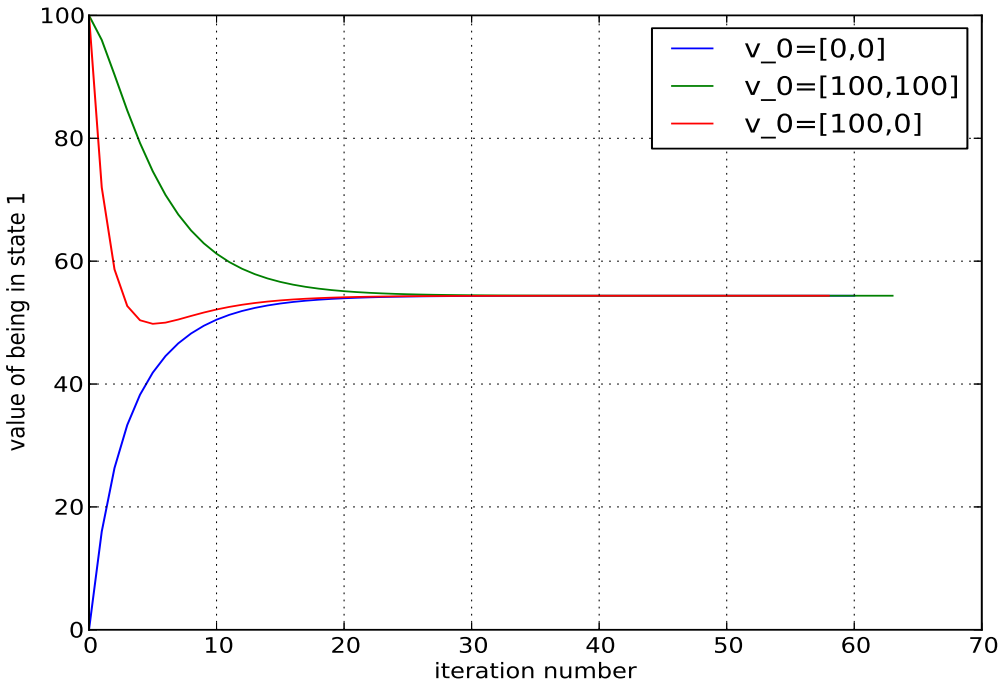

Figure 1: Convergence plots for the three state value function initializations of in Problem 3.3.

This is the expression we will do value iteration on. For value iteration we can use either version given in the book (regular or Gauss-Seidel). See the python code `chap_3_prob_3.py` where this problem is worked. When we initialize our value function estimate with the three suggested initial values and then run the regular value iteration algorithm we get the plot shown in Figure 1. In that plot we see two types of behaviors. Monotonic convergence to the final limit or oscillation around the final limit of the state value function.

Problem 3.7 (shipping oil)

Interpretation 1: Let our state S_t be the amount of full oil tanks at time t. Let a_t be the amount of oil to order $a_t \in \{0, 1, 2\}$. Our state transition model for this problem is then given by

$$S_{t+1} = \max(0, S_t - \hat{D}_{t+1}) + a_t \,,$$

where $\hat{D}_{t+1} \in \{0, 1, 2\}$ is the random demand for oil that happens in the time between t and $t+1$. Notice that the inventory does not arrive until time $t+1$ so we can not use it to satisfy any demand $\hat{D}_{t+1}$ for times between t and $t+1$. This is why the value of a_t is outside of the maximization above.

Lets now describe the "contribution" that we receive when in state S_t and choosing action a_t. We first note that the statement about the sales cost if we end up with more than two

tanks of oil is a bit confusing. I took this statement to mean that if we have more than two tanks of oil we will have to store that remaining oil on ships and satisfy all demand for oil first from these ships (at a reduced profit) before we can satisfy demand from the tanks of oil. This then becomes a constraint that prevents us from ordering "too much" oil as our profits greatly decrease when we have to sell the oil from the ship (a profit of 0.7 vs. 2).

To implement our contribution function $\hat{C}_{t+1}(S_t, a_t, \hat{D}_{t+1})$ we need to break our current state S_t down into the amount that is in tanks and the amount that is on ships as follows

$$S_t = \min(2, S_t) + \max(0, S_t - 2)\,.$$

By considering $S_t \in \{0, 1, 2, 3, \cdots\}$ and plotting each of the functions on the right-hand-side we see that the above expression is an equality.

Now from the problem statement (and considering the section from the book entitled "Random Contributions") we would have several terms in our expression of $\hat{C}_{t+1}$. We have

$$\begin{aligned}\hat{C}_{t+1}(S_t, a_t, \hat{D}_{t+1}) = &-1.6a_t \\ &- 0.02\max(0, S_t - \hat{D}_{t+1}) \\ &+ 0.7\min(\max(0, S_t - 2), \hat{D}_{t+1}) \\ &+ 2\min(\min(2, S_t), \max(0, \hat{D}_{t+1} - \max(0, S_t - 2)))\,.\end{aligned}$$

The first term is the cost to order oil that is delivered in the next time period. The second term is the cost that we have to pay to store any oil not sold in this time period. The third term represents the profit we make when we sell oil from the ships. We have $\max(0, S_t - 2)$ units of oil available on the ships that can be used to satisfy demand $\hat{D}_{t+1}$. The amount we sell is then the smaller of these two numbers. After this amount of oil is sold, the amount of demand still remaining is then

$$\max(0, \hat{D}_{t+1} - \max(0, S_t - 2))\,.$$

To meet this additional demand we can sell from the tanks of which we have $\min(2, S_t)$ units and a demand given by the previous expression. Thus the profit from selling from the tanks is given by the last line in the expression for $\hat{C}_{t+1}$. Since the above is a rather complicated expression to check that it is correct I evaluate it in the `python` code `oil_profit.py`. Next, to compute C_t we need to take the expectation of the above with respect to the given distribution of demand $\hat{D}_{t+1}$. We would then write a function that would evaluate the expectation of the above function $\hat{C}_{t+1}$ given the known distribution of demand.

Interpretation 2: Again let our state S_t be the amount of full oil tanks at time t. Let a_t be the amount of oil to order $a_t \in \{0, 1, 2, \cdots\}$. Our state transition model for this problem is then given by

$$\begin{aligned}\tilde{S}_{t+1} &= \max(0, S_t - \hat{D}_{t+1}) + a_t \\ S_{t+1} &= \min(\tilde{S}_{t+1}, 2)\,.\end{aligned}$$

Note that now our final state S_{t+1} is an element from $\{0, 1, 2\}$ which seems to be a condition of the problem. Here again $\hat{D}_{t+1} \in \{0, 1, 2\}$ is the random demand for oil that happens in

the time between t and $t+1$. If we end up with more than 2 tanks of oil we will have to sell it at a cost of 0.7 per ship of oil before we can meet demand in the next time period. This gives rise to the following contribution $\hat{C}_{t+1}$ function

$$\begin{aligned}\hat{C}_{t+1}(S_t, a_t, \hat{D}_{t+1}) &= -1.6a_t \\ &\quad + 0.7\max(\tilde{S}_{t+1} - 2, 0) \\ &\quad + 2\min(S_{t+1}, \hat{D}_{t+1}) \\ &\quad - 0.02\max(0, S_{t+1} - \hat{D}_{t+1})\,.\end{aligned}$$

The first term is the cost to order oil that is delivered in the next time period. The second term is the profit we make when we have to sell the oil from the ship due to ordering too much. The third term represents the profit made from selling oil from our storage tanks. Finally, the last line is the cost to store any extra oil. From the given expressions for $\tilde{S}_{t+1}$ and S_{t+1} we could write the above in terms of only the variables $S_t, a_t, \hat{D}_{t+1}$ but it seems wise to use intermediate variables. The parts of this problem are worked in the `python` code `chap_3_prob_7.py`.

Part (a): We are asked to evaluate the expectation of $\hat{C}_{t+1}(S_t, a_t, \hat{D}_{t+1})$ and we would have

$$C_t(S_t, a_t) = 0.4C_{t+1}(S_t, a_t, 0) + 0.4C_{t+1}(S_t, a_t, 1) + 0.2C_{t+1}(S_t, a_t, 2)\,.$$

We can evaluate this in `python` for all of the suggested values of S_t and a_t and get

```
s_t a_t C_t(s_t,a_t)
  0   0      0.000
  0   1     -0.408
  0   2     -1.624
  1   0     -0.008
  1   1     -0.416
  1   2     -1.344
  2   0      0.784
  2   1     -0.144
  2   2     -0.784
```

Part (b): We are asked to evaluate the expression P^π for various policies π i.e. ordering one or two tank of oil. We find

```
action=0                action=1                action=2
P=[[ 1.   0.   0. ]     P=[[ 0.   1.   0. ]     P=[[ 0.  0.  1.]
   [ 0.6  0.4  0. ]        [ 0.   0.6  0.4]        [ 0.  0.  1.]
   [ 0.2  0.4  0.4]]       [ 0.   0.2  0.8]]       [ 0.  0.  1.]]
```

Here we present the action and the corresponding transition probability matrix below the action.

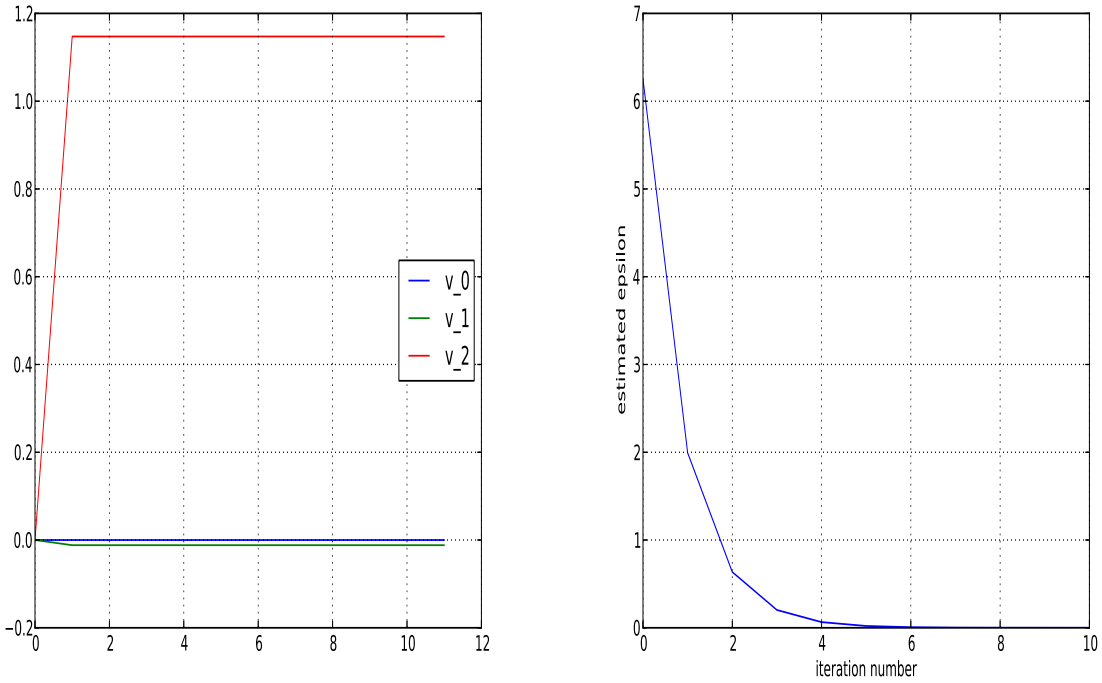

Figure 2: **Left:** Convergence plots for the three states in Problem 3.7 Part (d). **Right:** Convergence plots for $||v^{\pi^\epsilon} - v^*||$ for Problem 3.7 Part (e).

Part (c): When we implement Gauss-Seidel value iteration on this problem we get for the value in each state given by

```
[ 0.          -0.01176466   1.14740086]
```

Following the prescription to move to the state with the largest value we see that if we are in state 0 (have no full tanks) we should order 2 to try to get to the state 2. If we are in state 1 we should order one tank (to again try to get to state 2) and if we are in state 2 we should do nothing.

Part (d): Using the same `python` code we can plot $V^n(s)$ for each of the three states as a function of iteration number n. When we do this we get Figure 2 (left). At each iteration given the estimate of $V^n(s)$ our optimal action would be to move to the state with the largest value of $V^n(s)$.

Part (e): In the Gauss-Seidel value iterations loop we compute $||v^n - v^{n-1}||$ using a maximum norm for $||\cdot||$. Then following the book if we set

$$||v^n - v^{n-1}|| = \frac{\epsilon(1-\gamma)}{2\gamma} ,$$

we can solve for ϵ, since we know γ and the value of the left-hand-side. Once we have this

we can use Theorem 3.4.2 to argue that

$$||v^{\pi^\epsilon} - v^*|| \leq \epsilon \,.$$

Here the notation v^{π^ϵ} means we run value iteration with a stopping parameter ϵ and return the estimate of the state value function when it finishes. Then π^ϵ is the policy where we follow the prescription given by moving the state in v^{π^ϵ} that is maximal. When we estimate ϵ as describe above and plot it as a function of iteration we get Figure 2 (right).

Problem 3.9 (when to buy a new car)

Let S_t be the age of the car we are driving at time step t for $0 \leq t \leq T$. Then if our car does not break down during our state transition function is a function of our action a_t only and looks like

$$S_{t+1}(S_t, a_t) = \begin{cases} S_t + 1 & \text{when } a_t = 0 \text{ i.e. don't buy a new car} \\ 0 & \text{when } a_t = 1 \text{ i.e. do buy a new car} \end{cases} .$$

For this problem during the interval $t+1$ our car can break down and if this happens we must purchase another one. Let $\hat{W}_{t+1} = 0$ if there is no break down and $\hat{W}_{t+1} = 1$ if there is in the interval $t+1$. Adding this random component means that our state transition function now looks like

$$S_{t+1}(S_t, a_t, \hat{W}_{t+1}) = \begin{cases} S_t + 1 & a_t = 0 \quad \text{and} \quad \hat{W}_{t+1} = 0 \\ 0 & a_t = 0 \quad \text{and} \quad \hat{W}_{t+1} = 1 \\ 1 & a_t = 1 \quad \text{and} \quad \hat{W}_{t+1} = 0 \\ 0 & a_t = 1 \quad \text{and} \quad \hat{W}_{t+1} = 1 \end{cases} .$$

Notice in the above we include the unfortunate event where we decide to replace our car ($a_t = 1$) and also have a break down during the first timestep we drive our new car.

We now need to specify the cost that we must pay over the range of times t to $t+1$. Note that this cost is *random* i.e. it depends on whether or not the car breaks down during that interval. We denote this random cost as $\hat{C}_t(S_t, a_t, \hat{W}_{t+1})$. If we don't have a break down our cost $\hat{C}$ looks like

$$\hat{C}_{t+1}(S_t, a_t, \hat{W}_{t+1} = 0) = \begin{cases} P - r(0) + c(0) & \text{when } a_t = 1 \\ c(S_t) - r(S_t) & \text{when } a_t = 0 \end{cases} .$$

If we do have a break down our cost $\hat{C}$ will be given by

$$\hat{C}_{t+1}(S_t, a_t, \hat{W}_{t+1} = 1) = \begin{cases} P - r(0) + c(0) + K + P & \text{when } a_t = 1 \\ c(S_t) - r(S_t) + K + P & \text{when } a_t = 0 \end{cases} .$$

Lets now evaluate our expected costs $C_t(S_t, a_t) = E\{\hat{C}_{t+1}\}$. We find for each action that

$$\begin{aligned} C_t(S_t, a_t = 0) &= (c(S_t) - r(S_t))(1 - b(S_t)) + (K + P + c(S_t) - r(S_t))b(S_t) \\ &= c(S_t) - r(S_t) + (K + P)b(S_t) \\ C_t(S_t, a_t = 1) &= (P - r(0) + c(0))(1 - b(S_t)) + (2P + K - r(0) + c(0))b(S_t) \\ &= P - r(0) + c(0) + (K + P)b(S_t) \,. \end{aligned}$$

The equation we want to solve is then given by

$$V_t(S_t) = \max_{a_t} \left(C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|S_t, a_t) V_{t+1}(s') \right) .$$

In terms of this problem this breaks down into $V_t(S_t)$ being equal to the maximum of

$$c(S_t) - r(S_t) + (K+P)b(S_t) + \gamma[(1 - b(S_t))V_{t+1}(S_t + 1) + b(S_t)V_{t+1}(0)] ,$$

when $a_t = 0$ or

$$P - r(0) - c(0) + (K+P)b(S_t) + \gamma[(1 - b(S_t))V_{t+1}(1) + b(S_t)V_{t+1}(0)] ,$$

when $a_t = 1$. We could then use policy iteration to evaluate find the unknown function $V_t(S_t)$.

Problem 3.11 (a four-state process)

From the given diagram the transition probabilities (among different states) when we take the "wait" action is given by

$$P^{\text{wait}} = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0.9 & 0.1 \\ 1 & 0 & 0 & 0 \end{bmatrix} .$$

Note that the probability we stay in the same state increases as we go from $1, 2, 3, 4$ under the wait action. If we collect the reward our transition probabilities are given by

$$P^{\text{collect}} = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Note if we collect then in states two and three we go directly to state one. The contributions we receive under each of these actions are given by

$$C_t(S_t, \text{wait}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 20 \end{bmatrix} \quad \text{and} \quad C_t(S_t, \text{collect}) = \begin{bmatrix} 0 \\ 10 \\ 15 \\ 20 \end{bmatrix} .$$

We can find the optimal action to perform in each state $\{1, 2, 3, 4\}$ by running policy iteration where our actions are $a_t \in \{\text{wait}, \text{collect}\}$. Our initial policy $\pi^{(0)}$ will be taken to be

$$\pi^{(0)} = \begin{bmatrix} \text{wait} \\ \text{wait} \\ \text{wait} \\ \text{wait} \end{bmatrix} ,$$

so that we have initially

$$P^{\pi^{(0)}} = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad C^{\pi^{(0)}} = \begin{bmatrix} 0 \\ 0 \\ 15 \\ 20 \end{bmatrix} .$$

We will now implement policy iteration for this problem to find the steady state solution. This basically involves solving the linear system $(I - \gamma P^{\pi})v = c^{\pi^{(n-1)}}$ multiple times for the vector v . We implement this in the `python` code `chap_3_prob_11.py`. When we run that code we compute the long run policy vector π given by

```
[ 0.  1.  1.  0.]
```

This solution indicates that if we are in states 2 or three we should select the action to "collect" while if we are states 1 or 4 we select the action to "wait".

Problem 3.12 (the distance to the value function)

Notice that the value of each component in the value function v is increasing with each iteration. Now given the values of the vector v^n for $n = 8, 9, 10, 11, 12$ we will compute

$$||v^{12} - v^{11}|| ,$$

and set the given expression equal to $\frac{\epsilon(1-\gamma)}{2\gamma}$ to solve for ϵ. Then from Theorem 3.4.2 we know that

$$||v^{12} - v^*|| \leq \frac{\epsilon}{2} ,$$

which gives us a bound on how close to the optimal v^* we are. In the `python` code `chap_3_prob_12.py` we perform the above calculations and compute that

$$||v^{12} - v^*|| = \frac{10.8}{2} = 5.4 .$$

The norm used here is the maximum norm $||v|| = \max_{i=1,\cdots,n} |v_i|$.

Chapter 4 (Introduction to ADP)

Problem 4.1 (from uniform to exponential)

The cumulative distribution function for the random variable R can be computed by integrating over the set of uniform random variables U's that satisfy $R \leq x$. This set can be computed by considering the random variables U that satisfy

$$-\frac{1}{\lambda}\ln(U) \leq x \quad \text{or} \quad U \geq e^{-\lambda x}\,.$$

Thus the cumulative distribution function $\mathbb{P}$ is given by

$$\mathbb{P}[R \leq x] = \int_{U \geq e^{-\lambda x}} dU = \int_{e^{-\lambda x}}^{1} du = 1 - e^{-\lambda x}\,,$$

as requested.

Problem 4.2 (the sum of two uniform random variables)

For the random variable R defined by $R = U_1 + U_2$ we desire to compute its cumulative distribution function. This is defined as

$$\mathbb{P}[R \leq x] = \int_{\Omega} du_1 du_2\,,$$

with Ω the set of points (u_1, u_2) given by $\{(u_1, u_2)|u_1+u_2 \leq x\}$ i.e. the set of pairs of uniform random variables who's sum is less than x. This domain is represented by a triangle in the u_1-u_2 plane. Visualizing this domain we compute

$$\begin{aligned}
\mathbb{P}[R \leq x] &= \int_{u_1=0}^{x} \int_{u_2=0}^{x-u_1} du_2 du_1 \\
&= \int_{u_1=0}^{x} (x-u_1) du_1 = -\frac{(x-u_1)^2}{2}\Big|_0^x \\
&= -\frac{1}{2}\left(0 - \frac{x^2}{2}\right) = \frac{x^2}{2}\,.
\end{aligned}$$

This implies that the probability density function for R is given by

$$f_R(x) = x\,.$$

Problem 4.4

Lets compute the distribution function for the random variable $Z \equiv F^{-1}(U)$ where U is the uniform random variable on $[0, 1]$. Lets call this distribution function $G(z)$. Thus we have

$$G(z) = P[Z \leq z] = P[F^{-1}(U) \leq z] = P[U \leq F(z)]\,.$$

Since U is a uniform random variable we know what its distribution function is and thus have that

$$P[U \leq F(z)] = \int_0^{F(z)} 1 d\xi = F(z) \,.$$

Thus we see that $G(z) = F(z)$ thus the distribution function for $F^{-1}(U)$ is equal to $F(z)$ as we were to show.

Problem 4.5

Part (a): Following the example on page 132 "Selling an Asset" we let the pre-decision state is $S_t = R_t$ (which is one or zero depending on whether we have the asset at time t) and the post decision state is $S_t^a = R_t^a$ which is defined as

$$R_t^a = \begin{cases} 1 & \text{if } R_t = 1 \text{ and } a_t = 0 \\ 0 & \text{otherwise} \end{cases} .$$

Part (b-c): Let V_t be the value of holding the asset (and not selling) at the time t. Thus for this problem the value function will only depend on time t (and not the price p_t). We will use backwards dynamic programming to solve for V_t exactly. Starting at $t = T$ which is special since if we have the asset then we must sell it. Thus we have that

$$V_T = E[\hat{p}_T] = \bar{p} \,.$$

Here $\bar{p}$ is the average of the possible prices that could be offered for the asset on the last time step. From the problem description the prices are uniformly distributed between $[0, 10]$ and thus $\bar{p} = 5$ and so $V_T = 5$ also. Thus the value of having the asset at $t = T$ is the expected value obtained when one sell it (which seems to be a reasonable statement).

To move to the previous time step $T - 1$ (for the moment assuming discounting) we have

$$\begin{aligned} V_{T-1} &= \max_{a_{T-1}} (C_{T-1}(S_{T-1}, a_{T-1}) + \gamma V_T) = \max \begin{cases} E[\hat{p}_{T-1}] & \text{if we sell} \\ \gamma \bar{p} & \text{if we hold} \end{cases} \\ &= \max \begin{cases} \bar{p} & \text{if we sell} \\ \gamma \bar{p} & \text{if we hold} \end{cases} = \bar{p} \,, \end{aligned}$$

assuming that $\gamma \leq 1$. Working backwards once more we have

$$\begin{aligned} V_{T-2} &= \max_{a_{T-2}} (C_{T-2}(S_{T-2}, a_{T-2}) + \gamma V_{T-1}) \\ &= \max \begin{cases} E[\hat{p}_{T-2}] & \text{if we sell} \\ \gamma \bar{p} & \text{if we hold} \end{cases} = \max \begin{cases} \bar{p} & \text{if we sell} \\ \gamma \bar{p} & \text{if we hold} \end{cases} \\ &= \bar{p} \,, \end{aligned}$$

by the same logic as above. The pattern for V_{T-k} now seems clear and we have

$$V_{T-k} = \bar{p} \,.$$

for $0 \leq k \leq T - 1$.

If we know the value of V_t at each time t then the optimal selling algorithm is then the following. If at time t the price $\hat{p}_t$ is offered then we should sell the asset if $\hat{p}_t > V_t$ and hold (not sell) otherwise. Note that the threshold where we sell i.e. the value of V_t is a decreasing function of t as shown above.

Note that the threshold where we sell i.e. the value of V_t is a *constant* function of t. This result seems suspect in that it means that $V_1 = V_{10}$ while I would have thought that having more time to see what bids were presented would make the value of earlier times larger and would have expected to have $V_1 > V_{10}$ with strict inequality. In general, I would expect $V_k > V_{k+1}$ for all k. We in fact see that result numerically in the Monte-Carlo simulations done in the next part of this problem. The calculation done above seems to contradict this intuition somewhat. If anyone sees anything wrong with my logic please contact me.

Part (d): The approximate dynamic programming algorithm we use for this problem is as follows (this is derived by following the algorithm in Figure 4.2)

- Step 0: Initialization
  - Step 0a: Initialize $\bar{V}_t^0$ for all times t (we start with the value of zero).
  - Step 0b: Take the initial value of time to be $t = 0$.
  - Step 0c: Set $n = 1$.
- Step 1: Choose a sample path ω^n i.e. a set of offers $\hat{p}_1, \hat{p}_2, \cdots \hat{p}_9, \hat{p}_{10}$.
- Step 2: For $t = 1, 2, \cdots T-1, T$ do
  - Step 2a: Solve
    $$\hat{v}_t^n = \max \begin{cases} \hat{p}_t & a_t = 1 \\ \gamma \bar{V}_{t+1}^{n-1} & a_t = 0 \end{cases}$$
    Here $a_t = 1$ if we sell the asset and thus get a contribution of value $\hat{p}_t$. If we sell the asset we don't have it any longer and in a state with zero value. If $a_t = 0$ we don't sell the asset and move to the next time step.
  - Step 2b: Update $\bar{V}_t^n$ using
    $$\bar{V}_t^n = (1 - \alpha_{n-1})\bar{V}_t^{n-1} + \alpha_{n-1}\hat{v}_t^n \, .$$
- Step 3: Set $n = n + 1$ if $n < N$ and go to Step 1.

See the `python` code `chap_4_prob_5.py` where we implement the above approximate dynamic programming algorithm.

When we run that code we get the plot given in Figure 3. In that plot we see that the two different values for α converge to about the same estimate of $\bar{V}_t$. As noted above the true value of V_t (in green) is below both curves.

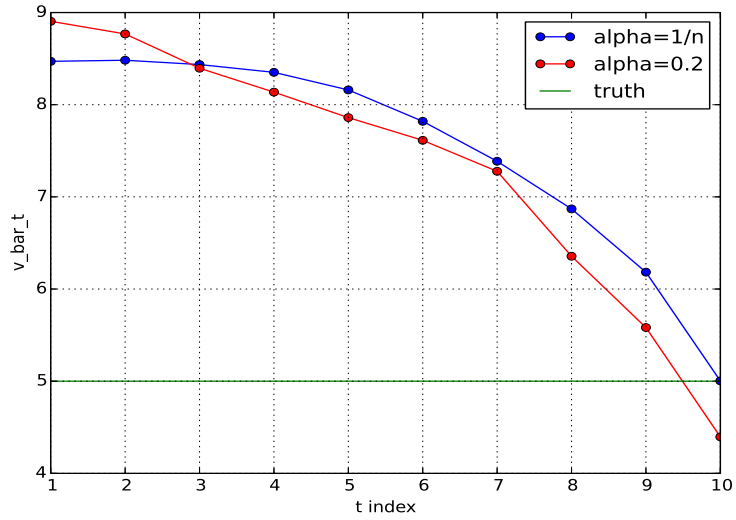

Figure 3: Value functions computed using approximate dynamic programming for Problem 5. The approximate value function obtained with $\alpha = \frac{1}{n}$ seems less noisy that the one computed when $\alpha = 0.2$.

Problem 4.6

To start this problem we might take the given asset price dynamics and simulate some trajectories to get a feel for how the price offered p_t depends on each time step. As the noise term $\hat{p}_{t+1}$ in the dynamics has a zero mean we can "drop" it from the given equation to get a feel for how the average offer price evolves. We can plot this average using the following R code

```
# Plot the price dynamics (the price of the asset increases as time does)
#
ps = c( 100 ) # store p0

# iterate the dynamics of price:
#
for( ii in 1:10 ){
    p_tp1 = ps[ii] + 0.5 * ( 120 - ps[ii] )
    ps = c( ps, p_tp1 )
}
plot( 0:10, ps, type='l', col='black', xlab='time', ylab='mean price' )
grid()
```

When we plot the above output we get the result given in Figure 4. Notice that the price offered increases as time goes on from the initial value of 100 to a value of around 120. In fact that the mean value of p_1 obtained from the recurrence relationship is $p_1 = 100 + 0.5(120 - 100) = 110$ means that even if we add the most negative amount of uniform noise possible

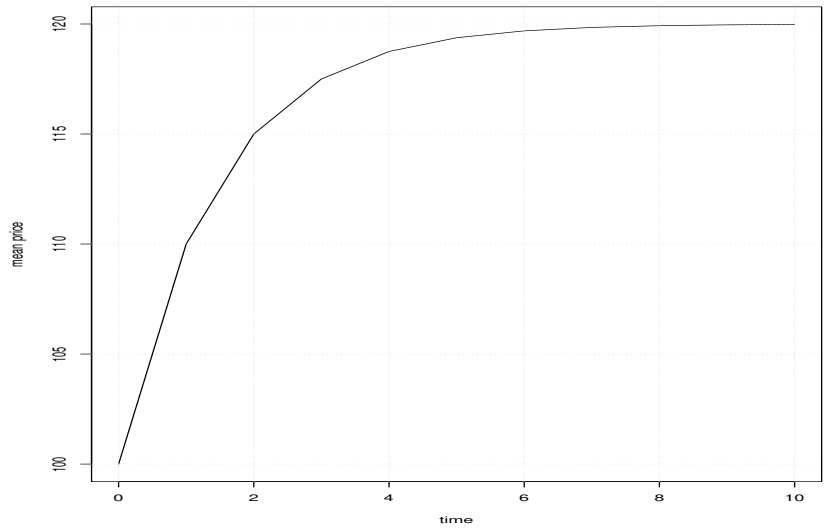

Figure 4: The mean evolution of p_t with time under the model $p_{t+1} = p_t + 0.5(120 - p_t)$.

we will have $p_1 \geq 100$. Thus we can take the value of 100 as our lower limit on the possible value for prices. In fact the value of 100 is also a good lower bound on the value function for all iterations (since the value of selling the option after $t = 0$ will always be larger than 100). Thus when we do our binning we will use the values of $[100, 130]$ as the lower and upper range of prices that p_t can evaluate to.

As mentioned in the problem statement the value function is now a function of p_t i.e. takes the form $V_t(p_t)$. While this functional form suggests that we should retain a time dependence for this problem I'll assume that the time dependence in V is really captured by the *price* dependence. This simplifies things in that V is only a one-dimensional function.

The approximate dynamic programming algorithm (forward dynamic programming) used to solve this problem is as follows (loosely following Figure 4.4 from the book)

- Step 0: Initialization
  - Step 0a: Initialize $\bar{V}_t^0(p)$ for all times t and discretized states p (taken to be zero).
  - Step 0b: The initial state at time $t = 0$ has the value of $p_0^1 = 100$.
  - Step 0c: Set $n = 1$
- Step 1: For $t = 0, 1, 2, \cdots T-2, T-1$ do
  - Step 1a: Choose a random sample of outcomes $\hat{\Omega}^n$ representing a sample realization of possible realizations (i.e. values of $\hat{p}_{t+1}$) between the times t and $t+1$.
  - Step 1b: Solve

$$\hat{v}_t^n = \max \begin{cases} p_t & a_t = 1 \\ \gamma \sum_{\hat{\omega} \in \hat{\Omega}^n} p^n(\hat{\omega}) \bar{V}_{t+1}^{n-1}(p_t + 0.5(120 - p_t) + \hat{\omega}) & a_t = 0 \end{cases}$$

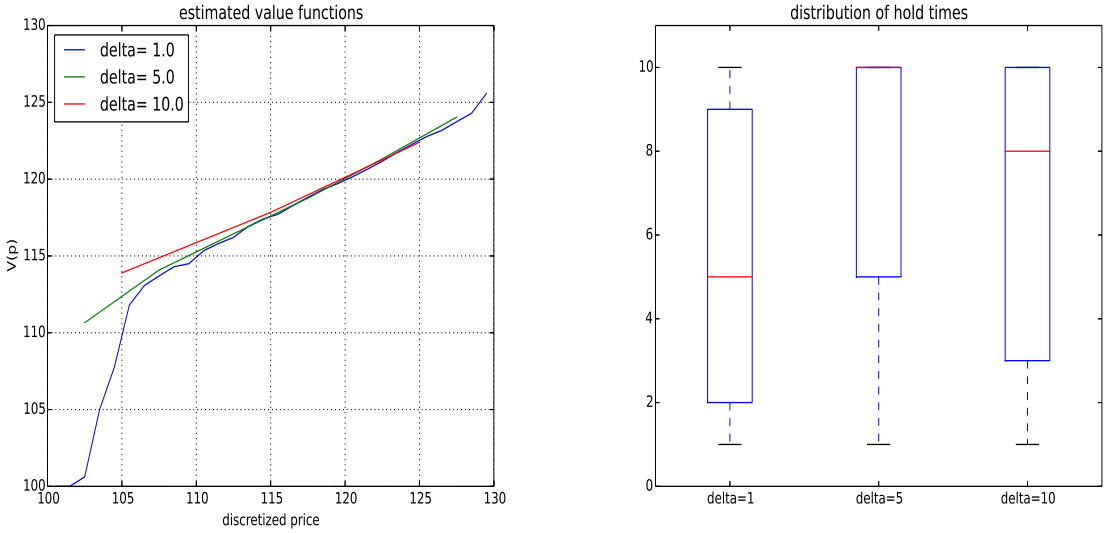

Figure 5: **Left:** Plots of the numerically estimated value function $V_t(p)$ for the three discretization parameters $\delta \in \{1, 5, 10\}$. **Right:** A box plot of the hold time of the asset when using a value function based on the three discretization parameters.

The reasoning for this expression is as follows. If if we decide to sell at time t then the contribution received is that due to the sale price p_t and we no longer have the asset which caries a value of zero. If we don't sell then we still have the asset and will move to the next time step with new prices. Here I'm denoting $\hat{\omega}$ as one of the sample increment in prices that we have from our realizations and the fact that our system model gives for the value of the price offered at the next time step

$$p_{t+1} = p_t + 0.5(120 - p_t) + \hat{\omega}\,.$$

- Step 1c: Update $\bar{V}_t^{n-1}(p)$ using

$$\bar{V}_t^n(p) = \begin{cases} (1-\alpha_{n-1})\bar{V}_t^{n-1}(p^n) + \alpha_{n-1}\hat{v}_t^n & p = p^n \\ \bar{V}_t^{n-1}(p) & \text{otherwise} \end{cases}$$

- Compute a single new random update in price $\hat{p}_{t+1}$ and determine the next price using

$$p_{t+1} = 0.5(120 - p_t) + \hat{p}_{t+1}\,.$$

- Step 2: Set $n = n + 1$ if $n < N$ and go to Step 1.

See the `python` code `chap_4_prob_6.py` where we implement the above approximate dynamic programming algorithm. When we run that code we first build approximations to $V_t(p)$ for the three suggested values of δ.

These three approximations for $V(p)$ are shown in Figure 5 (left). There we see that the finest descretization (the one with $\delta = 1$) has more structure to the curve than the other two for small values of p. This is just another way of saying that the estimate value of $V(p)$

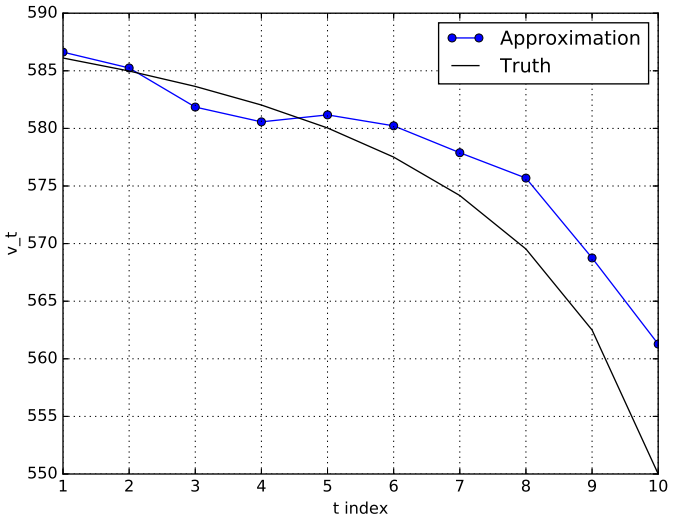

Figure 6: Plots of the true value function (in black) and the approximate value function (in blue) for this version of the asset selling problem.

(when p is small) is smaller than the others. Thus we expect the hold time in using this model will be less than that when we use the other approximations. This is because there is more chance that for these early times a random offer will come in that is larger than $V(p)$ and we will accept that offer and sell.

In Figure 5 (right) we present a boxplots of the hold time following the policy provided by the value function under each of the three approximations. We do indeed see that the hold time as far as the median value and perhaps the shape of the distribution for the $\delta = 1$ model is smaller than the other two.

Problem 4.7

Part (a): Let $\bar{V}_t$ be the approximate value of holding the asset at time t for $t \in \{1, 2, \ldots, 10\}$. Let $\hat{p}_t$ be the random offer received at time t. Then if the offer at time t is larger than the discounted future value of the upcoming offers we should sell. Mathematically the decision is made to sell if

$$\hat{p}_t > \gamma \bar{V}_{t+1} \, .$$

We would hold (and not sell) otherwise.

Part (b): A Monte Carlo recursive formulation for the exact value function is then

$$V_t = E[\max(\hat{p}_t, \gamma V_{t+1})] \quad \text{for} \quad t = 1, 2, \ldots, 9 \tag{6}$$

$$V_{10} = E[\hat{p}_{10}] = \frac{1}{2}(500 + 600) = 550 \, , \tag{7}$$

since on the last time step we must sell.

Given we know the distribution of $\hat{p}_t$ we can compute the *exact* value of V_t recursively as

$$\begin{aligned} V_t &= \frac{1}{100}\int_{500}^{600} \max(\hat{p}_t, \gamma V_{t+1}) dp \\ &= \frac{1}{100}\int_{500}^{\gamma V_{t+1}} \gamma V_{t+1} dp + \frac{1}{100}\int_{\gamma V_{t+1}}^{600} \hat{p}_t dp \\ &= \frac{\gamma V_{t+1}}{100}(\gamma V_{t+1} - 500) + \frac{\hat{p}^2}{200}\Big|_{\gamma V_{t+1}}^{600} \\ &= \frac{(\gamma V_{t+1})^2}{100} - 5\gamma V_{t+1} + \frac{1}{200}(600^2 - (\gamma V_{t+1})^2) \\ &= 1800 - 5\gamma V_{t+1} + \frac{1}{200}(\gamma V_{t+1})^2 \,. \end{aligned}$$

We can now start with $V_{10} = 550$ and iterate the above difference equation. Taking $\gamma = 1$ and doing this in the `python` code `chap_4_prob_7.py` we get the plot given in Figure 6 (black). Notice in looking at that plot we see that the asset is more valuable the longer we have before we have to sell, that is $V_t > V_{t+1}$.

Part (c): A Monte Carlo sampling approach to solving this problem would be to take Monte Carlo values for $\hat{p}_t$ and then update the current estimate of $\bar{V}_t^n$ using an equation similar to Equation 6 namely

$$\hat{v}_t^n = \max(\hat{p}_t, \gamma \bar{V}_{t+1}^{n-1}) \,,$$

as this is a sample based estimate we need to smooth it to get our approximation to V_t to use at the next episode

$$\bar{V}_t^n = (1 - \alpha_{n-1})\bar{V}_t^{n-1} + \alpha_{n-1}\hat{v}_t^n \,.$$

Part (d): The above approximate dynamic programming algorithm is implemented in the second part of `chap_4_prob_7.py`. When we run that code we get the plot given in Figure 6 (blue). Notice how that curve is noisier but follows the same pattern as the black (true) curve.

Part (e): From the above discussion the "price" at which we decide to sell is given by $\bar{p}_t = \gamma \bar{V}_{t+1}$.

Problem 4.8

Part (a): Taking the expectation inside the summation we have

$$F^T = \sum_{t=0}^{T} \gamma^t (50) = 50\frac{1 - \gamma^{T+1}}{1 - \gamma} \,.$$

If $\gamma = 0.7$ and $T = 20$ the above equals 166.6667.

Part (b): As suggested in the book we define

$$\hat{v}_t^n \equiv \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}(\omega^n) \,. \tag{8}$$

This is the discounted reward we will obtain for all samples from $t' = t$ to $t' = T$ along the n sample path denoted by ω^n. We can write the above in a recursive form as

$$\begin{aligned} \hat{v}_t^n &= R_t(\omega^n) + \sum_{t'=t+1}^{T} \gamma^{t'-t} R_{t'}(\omega^n) = R_t(\omega^n) + \sum_{t'=t+1}^{T} \gamma^{t'+1-(t+1)} R_{t'}(\omega^n) \\ &= R_t(\omega^n) + \gamma \sum_{t'=t+1}^{T} \gamma^{t'-(t+1)} R_{t'}(\omega^n) = R_t(\omega^n) + \gamma \hat{v}_{t+1}^n \,. \end{aligned} \tag{9}$$

Note that evaluating the definition of $\hat{v}_t^n$ in Equation 8 at $t = 0$ gives us a sample realization of $\hat{v}_0^n \approx F^T$.

Part (c): Starting with the ADP algorithm in Figure 4.4 Page 128 our algorithm will be

- Start by initializing $\bar{v}_t^0$ for $t = 0, 1, \ldots, T-1, T$. Here $\bar{v}_t^n$ will be the approximate value of the value function at time t and our right-most boundary condition is $\bar{v}_{T+1}^n = 0$ since there are no subsequent rewards for times later than T. Intuitively, we know that there is more "value" for earlier times t and thus we may want to initialize $\bar{v}_t^0$ in such a way that $\bar{v}_t^0 \geq \bar{v}_{t+1}^0$ represents this.
- Set $n = 1$
- Choose a sample path ω^n over time t (a set of $T+1$ random numbers) and compute for $t = T, T-1, \ldots, 1, 0$ a proposed *sample based* current state estimate following Equation 9 above i.e.

  $$\hat{v}_t^n = R_t^n + \gamma \hat{v}_{t+1}^n \,,$$

  and taking $\hat{v}_{T+1}^n = 0$ for all n.
- Update our value function approximation $\bar{v}_t^{n-1}$ using the previous sample based estimate $\hat{v}_t^n$ as

  $$\bar{v}_t^n = (1 - \alpha_{n-1}) \bar{v}_t^{n-1} + \alpha_{n-1} \hat{v}_t^n \,.$$

  Different expressions can be taken for α_{n-1} for example $\alpha_{n-1} = \frac{1}{n}$.
- Let $n = n + 1$ and go-to the first step.

This algorithm is implemented in the `python` code `chap_4_prob_8.py`. In that code we return the full value function approximation $\bar{v}_t^n$ but really only $\bar{v}_0^n$ is of interest as that is a sample realization of F^T.

Part (d): Running the given `python` code we get the plot given in Figure 7. In that plot we see the true value of F^T along with the values of $\bar{v}_t^n$ produced using $\alpha_{n-1} = \frac{1}{n}$ and $\alpha_{n-1} = 0.9$.

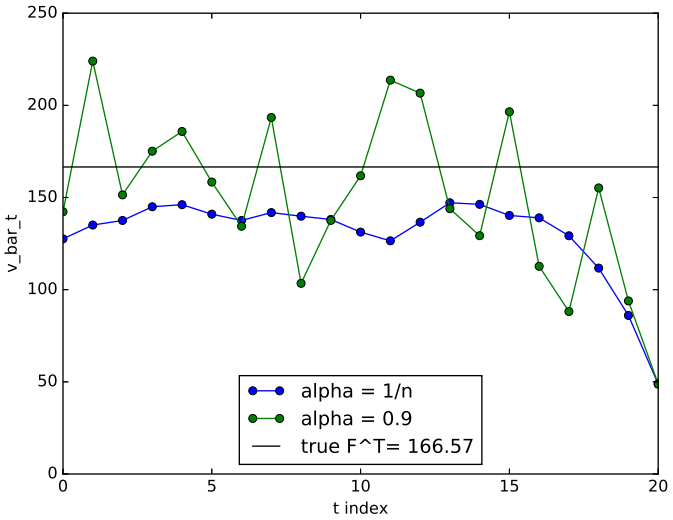

Figure 7: Plots of the ADP output for Problem 8.

Notice that the estimate of $\bar{v}_t^n$ for a fixed value of α_{n-1} is much more noisy. As can be see from the plot both choices for α_{n-1} give reasonable estimates of F^T.

Part (e-f): The confidence interval for the mean μ is given by

$$\left(\mu - z_{\alpha/2}\frac{\sigma}{\sqrt{n}}, \mu + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\right) .$$

Then to satisfy the required condition we want to have

$$\frac{1}{\mu}\left(\frac{2z_{\alpha/2}\sigma}{\sqrt{n}}\right) < 0.02 .$$

If we use the sample based estimates for μ and σ i.e. $\bar{X} \approx \mu$ and $s^2 \approx \sigma^2$ then we can use data (and the above) to compute the desired number of iterations n. We do that in the `python` code and get the following output

```
ADP estimates for F^T: mean: 147.940; std= 28.399; std err (10)= 2.840
percent error: 0.020 needs n > 1415.6
```

Here we have used the mean and the standard deviation of the 10 estimates of F^T to compute the number of iterations of our ADP algorithm we would have to run to get a percent error of less than 2%.

Problem 4.9

I was not sure how to work this problem.

Problem 4.10

Part (a-b): This problem is worked in the `python` code `chap_4_prob_10.py`. In that code we run both of the given policies according to the rules given. Doing this we get

```
Transfer 500K; n_mc= 1000; average cost: 93724.63; std cost= 681.72
Transfer 650K; n_mc= 1000; average cost= 96019.07; std cost= 1486.18
```

Thus it looks like the average cost of the new strategy is *more* and has a larger standard deviation.

Part (c): We could compute the number of Monte-Carlo observations that are needed using ideas like discussed in the book in the section on "Evaluating Policies". There we determine a confidence interval for the mean difference and require the number of Monte-Carlo iterations such that we can be confident that the difference is significant.

Part (d): If we used the same random demands to evaluate both strategies the results would be correlated and we would need to use a correlation to adjust the variance of the mean difference between the two polices. This is discussed in the book near the end of the section entitled "Evaluating Policies".

Problem 4.11

I was not sure how to work this problem.

Problem 4.12

I was not sure how to work this problem.

Problem 4.13

To start this problem lets compute the expected costs in moving from node i to node j (denoted as c_{ij}). Using the numbers given in the problem we have

$$
\begin{aligned}
c_{12} &= 0.4(2) + 0.6(10) = 6.8 \\
c_{24} &= 0.5(3) + 0.5(9) = 6 \\
c_{13} &= 0.2(2) + 0.8(8) = 6.8 \\
c_{34} &= 0.5(4) + 0.5(8) = 6\,.
\end{aligned}
$$

Notice that the expected cost in going on the top-half of the graph is *equal to* that if we go on the bottom half of the graph. In both cases this cost is $6.8 + 6 = 12.8$. We will determine (for the different methods below) an assessment of how important the information about the exact link lengths is when making decisions. Chance may give us very short lengths for some paths and if we have the information about this, our expected total path should be smaller.

Part (c): This part has the simplest formulation in terms of dynamic programming. Let $V(i)$ be the cost-to-go to node 4 from node i for $i \in \{1, 2, 3, 4\}$. We then have $V(4) = 0$. Dynamic programming would then say that that since there is no decision to make at nodes $i = 2$ and $i = 3$ that

$$V(2) = E[\text{cost to go to node 4 from 2}] = c_{24} = 6\,,$$

and

$$V(3) = E[\text{cost to go to node 4 from 3}] = c_{34} = 6\,.$$

The one node that has a decision to make is $i = 1$ where we have

$$V(1) = \min_{a \in \{2,3\}} (c_{1a} + \gamma V(a))\,.$$

As $c_{12} = c_{13} = 6.8$ and taking $\gamma = 1$ we have that $V(1) = 6.8 + 6 = 12.8$ for the expected cost in going from node 1 to 4.

Part (d): I would expect these three results to be ordered

$$V(b) < V(a) < V(c)\,.$$

Here $V(a)$ is the "cost-to-go" starting at node one for the formulation in Part (a) and corresponding meanings for $V(b)$ and $V(c)$. The motivation for the above inequalities are that in moving from left-to-right we have less and less information about the link edges in the graph. For example, we expect the cost-to-go from node 1 to node 4 to be the smallest in Part (b) because in that case we have knowledge of the entire realized network before we start traveling.

We now set up a simple Monte-Carlo simulation to evaluate if this idea is correct. In the `R` code `chap_4_prob_13.R` we estimate the cost-to-go for each of the parts using the information assumed available under each of the parts. Running that code we get

```
[1] "V_a= 10.886 (0.04); V_b= 10.424 (0.04); V_c= 12.805 (0.04)"
```

The values in parenthesis is an estimate of the standard error of the mean value for the cost-to-go. Note that the value estimated above *do* have the ordering mentioned above. In addition, the estimated value for $V(c)$ is quite close (with-in standard error) of the theoretical value of $V(c) = 12.8$

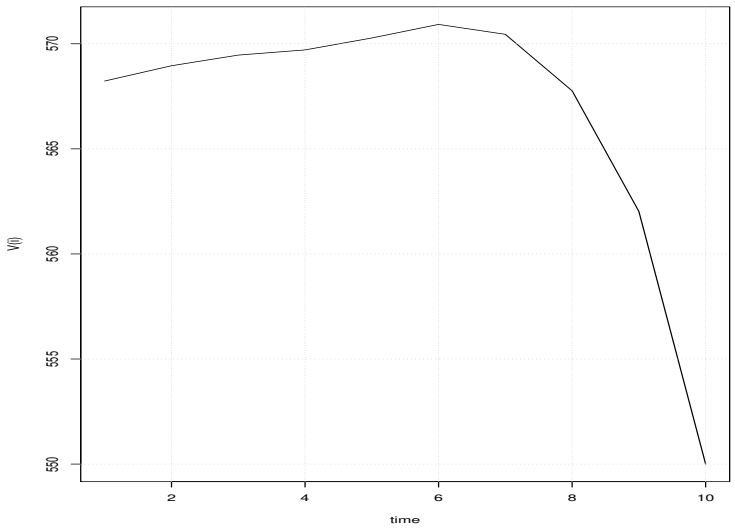

Figure 8: Plots of the numerically estimated value function $V(i)$ for the value of $i \in \{1, 2, \ldots, 9, 10\}$.

Problem 4.14

For this problem let $V(i)$ be the value of holding the asset at the time $i \in \{1, 2, \ldots, 9, 10\}$. We expect this function to satisfy $V(i) > V(i+1)$ as when i is small there is more time to consider more offers and "luck" might give us a very good price that we could sell at. For example, if we know that $\hat{p}_i \in [500, 600]$ and we see that $\hat{p}_1 = 599$ we have obtained a very good offer and should probably sell at that price. As we must sell at the tenth time step we have that

$$V(10) = E[\hat{p}_{10}] = \frac{1}{2}(500 + 600) = 550\,.$$

Then we assume that we have an estimate of V to use on iteration n given by $\bar{V}^{n-1}$ and then we compute

$$\hat{v}_i^n = \max(\hat{p}_i, \gamma \bar{V}^{n-1}(i+1))\,.$$

Here $\hat{p}_i$ is a sample of the offer at time i. Then we update our estimate of V using

$$\bar{V}^n(i) = (1 - \alpha_{n-1})\bar{V}^{n-1}(i) + \alpha_{n-1}\hat{v}^n\,.$$

As suggested by the problem we start with $\bar{V}^0(i) = 0$. As this is asynchronous state sampling we will randomly sample the states $i \in \{1, 2, \ldots, 8, 9\}$ many times applying the above two procedures. That is we don't necessarily sample them in any certain order.

We implement this algorithm in the `R` code `chap_4_prob_14.R` we implement the above procedure. When we run that code we get the estimate of $\bar{V}(i)$ given in Figure 8. Notice that the value of $V(i)$ increases for a while and then decreases quickly as we get closer to $t = 10$.

Problem 4.15 (the taxicab problem)

The grid given is 10×10 but with two columns partially removed. The state will be the location in the grid and the actions we can take at each state depend on where in the grid we are located. For ease of use we can either refer to a location in this grid (i.e. a state) using "matrix" notation or using a "linear indexing" notation. In code, this numerical value depends on if the programming language uses zero or one based indexing and whether the programming language represents matrices in FORTRAN or C ordering.

We will use `python` to implement value iteration starting with an initial random policy and then observe the final value function in each state. We do that in the code `chap_4_prob_15.py`. When we run that code we get the following output representing the value (the negative of) the expected number of steps needed to reach the goal) at each state

```
[[-18. -17. -16. -15. -14. -13. -12.  nan -10.  -9.]
 [-17. -16. -15. -14. -13. -12. -11.  nan  -9.  -8.]
 [-16. -15. -14. -13. -12. -11. -10.  nan  -8.  -7.]
 [-15. -14. -13. -12. -11. -10.  -9.  nan  -7.  -6.]
 [-14. -13. -12. -11. -10.  -9.  -8.  nan  -6.  -5.]
 [-15. -14. -13.  nan  -9.  -8.  -7.  nan  -5.  -4.]
 [-16. -15. -14.  nan  -8.  -7.  -6.  nan  -4.  -3.]
 [-17. -16. -15.  nan  -7.  -6.  -5.  -4.  -3.  -2.]
 [-18. -17. -16.  nan  -6.  -5.  -4.  -3.  -2.  -1.]
 [-19. -18. -17.  nan  -5.  -4.  -3.  -2.  -1.   0.]]
```

The walls are denoted by nan (as they are not valid states). Here I'm assuming a reward of -1 for every time step in the grid where we are not at the final state.

Problem 4.16 (value function initialization for the taxicab problem)

In working this problem I've assumed that we will run episodes that start at the upper left corner (the "in" arrow) and end at the lower right corner (the "out" arrow) moving greedily according to the current estimate of the state value function. We will sometimes "explore" by taking a random action in a given state rather than following the greedy policy. Thus with this problem implementation we are updating the value of each state as we move through the grid. This is different than the previous problem where we are allowed to update all states in the grid on each iteration.

In working this problem I found that the "initial condition" of the state value function was very important to determining how fast value iteration converged or even if it converged at all. See the notes below on each part for more information.

Part (a): With all estimates of the value function taken to be zero we end up with the problem where every action from the start state (and many states around this starting

location) look to have the same value. Thus initially the algorithm cannot select a reasonable direction to step and spends a huge number of iterations undergoing a random walk in regions of the state space where the value function is estimated the same. Because of this in the `python` code for the first 10 iterations I force a more aggressive exploring algorithm where we randomly choose actions at each step. By forcing random steps each time we are more likely to randomly reach the goal state and coarsely learn the directions to step. I should note that if I *don't* allow for these initial random explorations I don't seem to ever get an algorithm that converges.

This random exploration is needed until we get a few paths that reach the goal and from that point on using greedy actions performs better. Running the code in this way I'm getting

```
55
[[-18. -17. -16. -30. -16. -28. -27.  nan  -6. -14.]
 [-17. -16. -15. -16. -15. -29. -29.  nan -17. -12.]
 [-28. -15. -14. -13. -14. -29. -12.  nan -15. -17.]
 [-26. -27. -13. -12. -11. -28. -25.  nan -17. -15.]
 [-25. -26. -26. -11. -10.  -9.  -8.  nan -15. -17.]
 [-25. -25. -27.  nan -11. -10.  -7.  nan -17. -18.]
 [-25. -25. -27.  nan  -8. -13.  -6.  nan -19. -17.]
 [-24. -25. -25.  nan -23.  -6.  -5.  -4.  -3.  -2.]
 [-24. -25. -25.  nan  -8.  -7.  -4.  -3.  -2.  -1.]
 [-23. -24. -25.  nan  -7. -20.  -5.  -4. -19.   0.]]
```

Indicating that this run took 55 episodes of value iteration to converge to the given value function.

Part (b): In this case I still seem to need a decent amount of exploring in the beginning to get our algorithm to converge. With enough exploring the algorithm converges to a similar grid as shown above.

Part (c): In this case the suggested initial condition for $v_k(s)$ is basically the true solution. Thus my code for value iteration converges very quickly (in one or two iterations).

Problem 4.17 (the nomadic trucker project)

For this problem wee the `python` codes

- `chap_4_prob_17_utils.py`
- `chap_4_prob_17_part_a.py`
- `chap_4_prob_17_part_b.py`

where the parts of this problem are worked. When we run these codes we are attempting to estimate $\bar{V}^{n-1}(j)$ for each of the cities j using different approximate dynamic programming algorithm. Notice that while the *values* of the value function $\bar{V}^{n-1}(j)$ are different between the two approaches the *action* taken in each state might be the same due to the fact that it is the relative differences between values that matter to the decision we would take.

When we run the code for Part (a) for the rewards obtained during the simulation (after learning values for $\bar{V}^{n-1}$) are given by

```
n_times_visited=  [ 423.    0.    0.    0.    0.  271.    0.    0.    0.    0.
                      0.    0.  306.    0.    0.    0.    0.    0.    0.    0.]
Profit: mean: 1649.232; std: 2721.640
```

The same thing for Part (b) are given by

```
n_times_visited=  [  13.    2.    0.    5.   93.   66.   92.  159.    0.    1.
                      0.    0.   87.   40.  147.   61.    0.    0.  218.   16.]
Profit: mean: 2358.216; std: 2347.124
```

The second algorithm requires more computations to run but produces a larger profit mean and a smaller profit standard deviation. This is to be expected because the second algorithm spends more time exploring and should produce better estimates of the true state value function. Notice that the second algorithm visits more cities than the first one does.

Problem 4.18 (the Connect-4 project)

For this problem wee the `python` code `chap_4_prob_18.py`. Running that code on the command line gives a simple text based view of the board where the first players chip is the symbol "+1" and the second players chip is the symbol "-1". On each iteration you select a column to play (from the integers $0, 1, \ldots, 6$) and then the learned policy plays against you. I think if you try to play this game you will be surprised at how hard it is to win!

Chapter 5 (Modeling Dynamic Programs)

Problem 5.1

Part (a): We will formulate this as an "information" process. Let t be the time index for $0 \leq t \leq 7$ for the "moment" before the start of semester $t+1$. At the point in time before the semester starts she will have completed some number of each of the classes in the required departments. Thus we can let our state x_t be represented as

$$x_t = \begin{bmatrix} L_t \\ S_t \\ D_t \\ M_t \end{bmatrix} .$$

Here

$$\begin{aligned} L_t &= \text{be the number of language classes taken and passed} \\ S_t &= \text{be the number of science classes taken and passed} \\ D_t &= \text{be the number of departmental classes taken and passed} \\ M_t &= \text{be the number of math classes taken and passed} , \end{aligned}$$

before semester $t+1$ starts. When the student starts we have $t = 0$ and have

$$\begin{aligned} L_0 &= 0 \\ S_0 &= 0 \\ D_0 &= 0 \\ M_0 &= 0 . \end{aligned}$$

The constraints of the problem require that when $t = 8$ (after finishing her eighth semester) we require that we have taken a minimum number of classes of each type given by

$$\begin{aligned} L_8 &\geq 2 \\ S_8 &\geq 1 \\ D_8 &\geq 8 \\ M_8 &\geq 2 . \end{aligned}$$

We must also have that the total number of courses exceed 34 or the constraint

$$L_8 + S_8 + D_8 + M_8 \geq 34 .$$

Part (b): As actions, the student then decides how many of each type of class she will take during the $t+1$th semester for $0 \leq t \leq 7$. Thus we expect our "action" a_t is to have

$$a_t = \begin{bmatrix} a_t^L \\ a_t^S \\ a_t^D \\ a_t^M \end{bmatrix} .$$

Here $a_t^?$ are the number of course of the given type that the student attempt to take. There are constraints on the above in that $a_t^? \geq 0$ and

$$3 \leq a_t^L + a_t^S + a_t^D + a_t^M \leq 5 \,.$$

Assuming every class taken is passed we have

$$x_{t+1} = S^M(x_t, a_t) = x_t + a_t \,.$$

Part (c): If the passing of a class is random then the state transition function would take the form of something like

$$x_{t+1} = S^M(x_t, a_t) = x_t + \hat{a}_t \,,$$

where

$$\hat{a}_{ti} \sim \mathcal{U}[0, a_{ti}] \,.$$

The above statement indicates that once the number of classes of a given type has been decided (i.e. the number a_{ti}) the *actual* number passed (i.e. $\hat{a}_{ti}$) is to be a uniform random variable drawn between zero (passing none of the classes of this type taken) and a_{ti} (for passing all of the classes of this type taken). Obviously different probability models for $\hat{a}_{ti}$ are possible.

Problem 5.3

In this case he is using the informational representation. At the end of the tth day one would have observed $\hat{v}_t$ and the only information needed to compute the forecast at time t is f_{t-1}. Thus the state variable is f_t.

Problem 5.4

In this case he has to have all of the nine previous volumes $\{\hat{v}_j\}_{t=t-9}^{j=t-1}$ around after he observed $\hat{v}_t$ to compute his forecast f_t. Thus the state variable is ten dimensional i.e. has ten elements in it.

Problem 5.5

Part (a): I think the idea of this problem is to assume that we are sampling at integer times from the start to the end of the day. Then as it takes only one timestep to get from location to location. Then at each sampling we can only be located at his current location and not traveling between the two locations. Thus we need no additional information in determining where salesman is located.

Part (b): In this case as it may take several timesteps to move from i to j the state of the system would need to include his past location (say i) his future location (say j) and how

long it might take him to get there (which is stored in τ_{ij}). We could then determine if he is at his origin i, destination j or between cities when we receive a query as to his location.

Part (c): I think this is like Part (b) but in making decisions we would need to know the value of the random draw τ_{ij} before we make our decision as to which city j to travel to. Once we decide on a city j we would need to store that random travel time.

Problem 5.6

Part (a): We are using the informational representation for this problem. I also believe that the variables D_t are *known* at each timestep i.e. not random. The transition function would be

$$R_{t+1} = S^M(R_t, x_t) = \max(R_t + x_t - D_{t+1}, 0)\,.$$

Part (b): The post-decision state variable would be given by

$$R_t^x = R_t + x_t\,.$$

Part (c): The post-to-pre transitions would look like

$$\begin{aligned} R_t^x &= R_t + x_t \\ R_{t+1} &= \max(R_t^x - D_{t+1}, 0)\,. \end{aligned}$$

Problem 5.7

Part (a): This is like the section in this chapter entitled *Resource Acquisition II – Purchasing Futures*. Thus we need our state to be $R_{tt'}$ to be the number of futures held now (time period t) that can be exercised in time period t' for $t' > t$.

Part (b): Then assuming we cannot purchase on the spot market, our transition function would depend on the random demand in time period $t+1$ i.e. $\hat{D}_{t+1}$

$$R_{t+1,t+1} = \max(R_{t,t} + R_{t,t+1} + x_{t,t+1} - \hat{D}_{t+1}, 0)\,,$$

which is the $t' = t+1$ case and

$$R_{t+1,t'} = R_{t,t'} + x_{t,t'}\,,$$

for $t' > t$.

Problem 5.8

I think there is a typo with the notation in this problem. If we use the informational representation and take D_t to be the random demand in the time period between $(t, t+1)$ then the transition function would look like

$$R_{t+1} = \max(R_t + x_t - \hat{D}_{t+1}, 0)\,.$$

Here the distribution of $\hat{D}_{t+1}$ would be given by the p_d numbers. This is the same equation we obtained in the section of the book *Resource Acquisition I – Purchasing Resources for Immediate Use*.

Problem 5.9

Part (a): The state here would be the location of the resource. I'm assuming that the costs to travel each link are known and accessible to the algorithm and thus are not necessarily stored in the state variable.

Part (b): Here you have to estimate c_{ij} as you move in the network and thus estimates of all of these costs would have to be held in memory somewhere. In this problem formulation we can only update the single estimate of c_{ij} for the link $i \to j$ we choose to take.

Part (c): I think this is like Part (b) in that estimate for each of these costs would have to be stored and computed. Once difference is that because we see all of the costs (not just the cost of the link we take) upon entering a node we can update our estimates for c_{ij} for all j.

Part (d): I think in this case as the distribution of travel times is *known* i.e. we are told what it is in the problem statement we don't have to know anything other than our current location i.

Part (e): This is like Part (d) but in this case we don't know θ and thus it would need to be estimated as we transition in our set of cities.

Problem 5.10

Part (a): For the state in this problem we would need to keep track of the (x, y) location of the boat. As observations we would need to know the wind speed as a vector or (w_x, w_y) . We would then adjust the boats velocity (v_x, v_y) to make sure that we generally maintain an approximate 45° angle with the wind. Thus the vector (v_x, v_y) is the "action space". The exogenous information is the wind speed as that is random and changes. As a contribution function we could simplify take a contribution of -1 for every timestep we are not at the location B and stop the episode when we reach B.

If each step is done for a length of time Δt then after taking an action by specifying (v_x, v_y) we should end up at the new location (x', y') given by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \Delta t \begin{bmatrix} v_x - w_x \\ v_y - w_y \end{bmatrix}.$$

Part (b): Let the domain the boat is in be of size $L \times L$ then the number of possible values

for (x, y) if we discretize the state to a size of δ we would have a total state dimension of

$$\left(\frac{L}{\delta}\right)^2 .$$

Assuming that the velocity can only be positive (or one signed) and bounded by V to discretize either our action space (v_x, v_y) or our observation space (w_x, w_y) would require

$$\left(\frac{V}{\delta}\right)^2 ,$$

elements.

Part (c): From the above we would have

$$\frac{V}{\delta} = \frac{30}{0.1} = 300 .$$

Squaring this we get 90000 which is quite a large space to search.

Problem 5.12

One can think of the history of the process as the sequence of state, action, and outcome that our agent observes at each timestep as they attempt to optimally perform their required task.

Problem 5.13

To make this problem a bit simpler I'll assume that there is *one* spot price $\hat{p}_t^{\text{spot}}$ for each supplier during week t (I don't think it makes much sense for each supplier to have their own spot price).

Part (a): This would be the tuples $(\hat{p}_{ti}, \hat{q}_{ti})$ for each of the suppliers $i \in \{1, 2, \ldots, |\mathcal{I}|\}$ and the single weekly spot price $\hat{p}_t^{\text{spot}}$.

Part (b): These would be the ordering amounts x_{ti} for each of the $|\mathcal{I}|$ suppliers.

Part (c): The current inventory during week t i.e. s_t is the state variable.

Part (d): The transition function for s_t is

$$s_{t+1} = S^M(s_t, x_t) = \max(s_t - D_t, 0) + \sum_{i=1}^{|\mathcal{I}|} x_{ti} . \tag{10}$$

Part (e): The one-period contribution/cost function for C_t is

$$C_t(s_t, x_t) = -\sum_{i=1}^{|\mathcal{I}|} x_{ti}\hat{p}_{ti} - \max(D_t - s_t, 0)\hat{p}_t^{\text{spot}} .$$

This is the contribution in week t of making decisions x_{ti}.

Part (f): Lets consider an "order-up-to" policy. If

$$s_t - D_t < q \,,$$

meaning that after satisfying our demands in week t we have a small amount of product left then we specify x_{ti} such that we "order-up-to" some larger amount i.e.

$$\sum_{i=1}^{|\mathcal{I}|} x_{ti} = Q \,.$$

Part (g): We want to maximize

$$C_t(s_t, x_t) + \gamma V_{t+1}(S^M(s_t, x_t)) \,,$$

over the vector x_t subject to the constraint that $x_t < \hat{q}_t$.

Part (h): The space of policies in this case are just all possible policies parameterized by the two constants (q, Q).

Problem 5.14

Part (a): An instance of ω is a specific sequence of random calls at the times t_e with random service duration τ_e and reward r_e.

Part (b): A decision epoch will be one run where all of the calls are either rejected for having r_e/τ_e too small or placed in a queue and then serviced. The epoch will end when the queue is empty. An artificial epoch end could be enforced say at the end of a work day or after so much time had elapsed.

Part (c): The state variable of the system would be the list of requests that must be serviced. After the first item in the list is serviced the list would shrink by one and grow by the number of events that were received in the time it took to process this last event.

Part (d): The action space is to accept or reject an incoming request depending on the ratio of r_e to τ_e.

Part (e): The one period reward function would be

$$r_e \,,$$

the reward for the most recently processed request.

Problem 5.15

Part (a): This would be the tuples $(\hat{p}_{ti}, \hat{q}_{ti})$ for each of the sources.

Part (b): The state would be the current inventory (amount of oil in millions of barrels) during week t i.e. s_t is the state variable. The transition function for s_t is

$$s_{t+1} = S^M(s_t, x_t) = s_t + \sum_{j=1}^{|\mathcal{J}|} x_{tj} \,, \tag{11}$$

with $s_0 = 0$.

The weekly cost C_t is then given by

$$C_t = \sum_{j=1}^{|\mathcal{J}|} x_{tj} \hat{p}_{tj}$$

If there is no discounting then our global goal is to minimize the total cost

$$\sum_{t=1}^{10} C_t \,,$$

subject to the constraint that we obtain the final 20 million barrels or

$$s_{10} = 20 \,.$$

Part (c): A very simple strategy might be to buy all oil below a price $\bar{p}$ and stop when we have enough oil.

Part (d): A policy in the above formulation is parameterized by the value of $\bar{p}$.

Part (e): We seek to minimize the total cost paid or

$$\mathbb{E}\left[\sum_{t=1}^{10}\left(\sum_{j=1}^{|\mathcal{J}|} x_{tj}\hat{p}_{tj}\right)\right] \,,$$

subject to the constraint that we have enough oil or

$$s_{10} = \sum_{t=1}^{10}\left(\sum_{j=1}^{|\mathcal{J}|} x_{tj}\right) = 20 \,,$$

and

$$x_{tj} < \hat{q}_{tj} \,,$$

for $1 \leq t \leq 10$ and $1 \leq j \leq |\mathcal{J}|$.

Problem 5.16

Part (a): The state space of this problem is $s_t = 1$ if we own the asset during week t or $s_t = 0$ if not.

Part (b): The action space is the decision to sell the asset.

Part (c): The exogenous process are the asset returns $\hat{r}_t$. In a five period example we would have an initial price p_0 then the price at the end of the first week is

$$\hat{r}_1 p_0 \,,$$

the price at the end of the second week is

$$\hat{r}_2 p_1 \,,$$

and so on. The price at the end of the fifth week is

$$\hat{r}_5 p_4 \,.$$

Part (d): The objective function we seek to maximize is

$$\bar{p} + q\bar{p}(100 - T) \,,$$

where T is the first time that the price p_T is less than $\bar{p}$. In the above the first term is the profit we make from selling our asset one share sold at a price of $\bar{p}$ and the second term is the profit we make once we invest $\bar{p}$ dollars in the bank. If the price never falls below $\bar{p}$ then the final value of the objective function would be

$$p_{10} \,.$$

Chapter 6 (Policies)

Problem 6.1

For this problem we will need to recall the four broad categories of policies discussed in this chapter (see the book for more detail).

- Myopic policies
- Lookahead policies
- Policy function approximations
- Value function approximations

For the given actions taken I would think

- This is a myopic policy as it only considers the temperature at the beginning of the day and not any future information like where you might be at the end of the day and if having a jacket there is helpful or not.
- This is a lookahead policy in that the computer algorithm needs to consider the consequences of taking different paths as it seeks its way to your destination.
- I think this is a lookahead policy in that we are planning for one day forward which actions we are going to take. Thus we have a suboptimization problem that we have to solve to move forward one timestep.
- The chess player is deciding how likely a win might be by looking at the position of the board. This is an example of a value function policy.
- I think this is a lookahead policy as the stock broker is planning based on using information about the future outcome of events.
- This is again a lookahead policy as the utility company is planning for 12 months (the lookahead) of water flow and then acting now based on those predictions.
- This seems like a myopic policy as it is only using very recent data and no planning of the future expected events is included in the policy.
- As we have a projection for the next season it look like this is a lookahead policy? The information about us adding a function indicates that we have an underlying state (the number of TVs) and we might be acting based on that which would indicate that this policy is similar to a value function?
- As there is no real "planning" in this procedure I'm guessing that this is a myopic policy as the adjustments made today are really about optimizing very recent (tomorrows) rewards.

Problem 6.2

When this example was discussed in the book it is for a *myopic* (i.e. short term) policy and we are assuming that $t < \tau_j$ when we do our planning. In that case for $0 \leq t \leq \tau_j$ we have

$$e^{-\theta_1 \tau_j} < e^{-\theta_1(\tau_j - t)} < 1 \,.$$

Thus the contribution $c^{\pi}_{tij}(\theta)$ is only "slightly" increased over the baseline c_{ij}. If we plan much further into the future say for $t \sim T$ then we might end up considering $t > \tau_j$ and our planning policy would estimate an expected contribution where

$$c^{\pi}_{tij}(\theta) = c_{ij} + \theta_0 e^{-\theta_1(\tau_j - t)} \rightarrow \infty \,,$$

when $t \gg \tau_j$. This may mean that our system might "wait" i.e. not process any jobs now to be sure that it is able to perform these estimated value future jobs. This is not the intended modification (to better select jobs about to expire) and would most likely not give the behavior desired (a modification where we stop the exponential increase might however).

Chapter 7 (Policy Search)

Problem 7.1

Rather than use a spreadsheet I did this simulation in the `python` code `chap_7_prob_1.py`.

Part (a): This is done in the given python code and we find the following "returns"

```
[9.329 8.46  4.961 9.189 7.95  9.764 9.663 9.757 5.052 9.292]
```

This gives

```
beta= 1.0; mean= 8.3; std= 1.8; std_mean= 0.55
```

Part (b): Doing the same thing for $\beta = 2$ we get

```
beta= 2.0; mean= 6.9; std= 4.1; std_mean= 1.3
```

Running a t-test to compare the means between these two samples using the `ttest_ind` function in `scipy.stats` I get

```
Ttest_indResult(statistic=0.9482543241498, pvalue=0.35555822243230706)
```

This result indicates (with its P-value larger than 0.05) that these two sets are *not* different at the 95% significance level. As an aside I didn't find significant differences between the the two results even when I ran 1000 simulations each.

Part (c): When I implement this stochastic gradient ascent algorithm (but with 100 gradient ascent steps and 100 simulations each) I find

```
... more output here ...
ii=   90; beta=      0.543; g_n=      0.267; F_bar=       8.46
ii=   91; beta=      0.556; g_n=      0.209; F_bar=       8.68
ii=   92; beta=      0.566; g_n=      0.417; F_bar=        8.1
ii=   93; beta=      0.587; g_n=     -0.704; F_bar=       8.35
ii=   94; beta=      0.553; g_n=        0.4; F_bar=       8.43
ii=   95; beta=      0.572; g_n=       0.12; F_bar=       9.06
ii=   96; beta=      0.578; g_n=     -0.119; F_bar=       8.41
ii=   97; beta=      0.572; g_n=     0.0157; F_bar=       6.62
ii=   98; beta=      0.573; g_n=      0.134; F_bar=       8.79
ii=   99; beta=      0.579; g_n=    -0.0278; F_bar=       7.78
```

In this part by only looking at the first few iterations of stochastic gradient ascent it was not clear to me that this algorithm was converging. But by running more steps in the gradient ascent algorithm and using a larger number of trajectories to get a better estimate the values of $\bar{F}(\beta)$ and g^n (as was done in generating the above) we see that it does look to be converging. The optimal value of β appears to be $\beta^* \approx 0.57$.

Part (d): I think this part means to run just three trajectories and then with them estimate $\bar{F}(\beta)$ for each value of β. Unfortunately when you run the `python` code for this part of the problem you see that the means of $F(\beta)$ for each of the β values do not seem differ very much. With only $n = 3$ samples of F for each β value the results are very noisy and its not clear that a definitive pattern exists. Using $n = 100$ samples to estimate $F(\beta)$ for each β still does yield a clear picture of the optimal value for β. This seems in direct contrast to the stochastic gradient ascent results in the previous part where the optimal value of β should be around 0.57. If anyone sees anything I have done wrong please contact me.

Problem 7.2

Rather than use a spreadsheet I did this simulation in the `python` code `chap_7_prob_2.py`.

I didn't see a value given for the parameter μ but it seemed like a problem could be worked if we took $\mu = \frac{1}{2}(45 + 25) = 35$.

WWX: working here.

Chapter 8 (Approximating Value Functions)

Problem 8.1

WWX: do this problem.

Problem 8.2

WWX: do this problem.

Problem 8.3

Recall that $\overline{\theta}^{n-1}$ is the *random* average of $n-1$ random observations $\hat{\theta}^k$ about a non-stationary (and nonrandom) mean θ^k for $1 \leq k \leq n-1$ by expanding the quadratic in the expression given we have

$$\begin{aligned} E\left[(\overline{\theta}^{n-1} - \theta^n)^2\right] &= E\left[(\overline{\theta}^{n-1})^2 - 2\overline{\theta}^{n-1}\theta^n + (\theta^n)^2\right] \\ &= E\left[(\overline{\theta}^{n-1})^2\right] - 2E\left[\overline{\theta}^{n-1}\right]\theta^n + (\theta^n)^2 \,. \end{aligned}$$

Next from the definition of the bias β^n we have

$$E\left[\overline{\theta}^{n-1}\right] = \beta^n + \theta^n \,,$$

and using the fact that

$$E(X^2) = \mathrm{Var}(X) + E(X)^2 \,,$$

we can write the above as

$$\begin{aligned} E\left[(\overline{\theta}^{n-1} - \theta^n)^2\right] &= \mathrm{Var}\left(\overline{\theta}^{n-1}\right) + E\left[\overline{\theta}^{n-1}\right]^2 - 2(\beta^n + \theta^n)\theta^n + (\theta^n)^2 \\ &= \mathrm{Var}\left(\overline{\theta}^{n-1}\right) + (\beta^n + \theta^n)^2 - 2\beta^n\theta^n - 2(\theta^n)^2 + (\theta^n)^2 \\ &= \lambda^{n-1}\sigma^2 + (\beta^n)^2 + 2\beta^n\theta^n + (\theta^n)^2 - 2\beta^n\theta^n - (\theta^n)^2 \\ &= \lambda^{n-1}\sigma^2 + (\beta^n)^2 \,, \end{aligned}$$

as we were to show.

Problem 8.4

Recall that the nth sample $\hat{\theta}^n$ is assumed to be normally distributed as $\hat{\theta}^n \sim \mathcal{N}(\theta^n, \sigma^2)$ so that

$$\hat{\theta}^n = \theta^n + \varepsilon^n \,,$$

where θ^n is a constant and ε^n is random. Next expanding the given expectation we have

$$\begin{aligned}\nu^n &= E\left[(\overline{\theta}^{n-1} - \hat{\theta}^n)^2\right] = E\left[(\overline{\theta}^{n-1} - \theta^n - \varepsilon^n)^2\right] \\ &= E\left[(\overline{\theta}^{n-1} - \theta^n)^2 - 2\varepsilon^n(\overline{\theta}^{n-1} - \theta^n) + (\varepsilon^n)^2\right] \\ &= \lambda^{n-1}\sigma^2 + (\beta^n)^2 + 0 + E\left[(\varepsilon^n)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2 + \sigma^2 \\ &= (1+\lambda^{n-1})\sigma^2 + (\beta^n)^2\,.\end{aligned}$$

In the above derivation we have used the result from the previous exercise and the fact that

$$E[\varepsilon^n(\overline{\theta}^{n-1} - \theta^n)] = E[\varepsilon^n]E[\overline{\theta}^{n-1} - \theta^n] = 0 \times E[\overline{\theta}^{n-1} - \theta^n] = 0\,,$$

by the independence of the ε^n.

Problem 8.5

We start with the definition of variance, expand, and evaluate. We have

$$\begin{aligned}\mathrm{Var}\left(\hat{\theta}^n\right) &= \frac{1}{n-1}\sum_{m=1}^{n}(\hat{\theta}^m - \overline{\theta}^n)^2 \\ &= \frac{1}{n-1}\sum_{m=1}^{n}(\hat{\theta}^m - \overline{\theta}^{n-1} + \overline{\theta}^{n-1} - \overline{\theta}^n)^2 \\ &= \frac{1}{n-1}\sum_{m=1}^{n}\left[(\hat{\theta}^m - \overline{\theta}^{n-1})^2 + 2(\hat{\theta}^m - \overline{\theta}^{n-1})(\overline{\theta}^{n-1} - \overline{\theta}^n) + (\overline{\theta}^{n-1} - \overline{\theta}^n)^2\right] \\ &= \frac{1}{n-1}\sum_{m=1}^{n}(\hat{\theta}^m - \overline{\theta}^{n-1})^2 + \frac{2(\overline{\theta}^{n-1} - \overline{\theta}^n)}{n-1}\sum_{m=1}^{n}(\hat{\theta}^m - \overline{\theta}^{n-1}) + \left(\frac{n}{n-1}\right)(\overline{\theta}^{n-1} - \overline{\theta}^n)^2\,.\end{aligned}$$

Now in the above expression the limit on the sum in the second term is n but we can evaluate it using the fact that

$$\sum_{m=1}^{n-1}(\hat{\theta}^m - \overline{\theta}^{n-1}) = 0\,.$$

This is true because $\overline{\theta}^{n-1}$ is the mean of the first $n-1$ samples $\hat{\theta}^m$. Note the upper limit on that sum is $n-1$. Using the general idea of summing to $n-1$ rather than n the expression for $\mathrm{Var}(\hat{\theta}^n)$ becomes

$$\begin{aligned}\mathrm{Var}\left(\hat{\theta}^n\right) &= \frac{1}{n-1}\left[\sum_{m=1}^{n-1}(\hat{\theta}^m - \overline{\theta}^{n-1})^2 + (\hat{\theta}^n - \overline{\theta}^{n-1})^2\right] + \frac{2(\overline{\theta}^{n-1} - \overline{\theta}^n)}{n-1}\left[0 + (\hat{\theta}^n - \overline{\theta}^{n-1})\right] + \left(\frac{n}{n-1}\right)(\overline{\theta}^{n-1} - \overline{\theta}^n) \\ &= \frac{1}{n-1}\left[(n-2)\mathrm{Var}(\hat{\theta}^{n-1}) + (\hat{\theta}^n - \overline{\theta}^{n-1})^2\right] + \frac{2(\overline{\theta}^{n-1} - \overline{\theta}^n)(\hat{\theta}^n - \overline{\theta}^{n-1})}{n-1} + \left(\frac{n}{n-1}\right)(\overline{\theta}^{n-1} - \overline{\theta}^n)^2 \\ &= \frac{n-2}{n-1}\mathrm{Var}\left(\hat{\theta}^{n-1}\right) + \frac{(\overline{\theta}^{n-1} - \hat{\theta}^n)^2}{n-1} + \frac{2(\overline{\theta}^{n-1} - \overline{\theta}^n)(\hat{\theta}^n - \overline{\theta}^{n-1})}{n-1} + \left(\frac{n}{n-1}\right)(\overline{\theta}^{n-1} - \overline{\theta}^n)^2\,.\end{aligned}$$

Now to continue the evaluation of the above we will derive the recursive formulation of the mean. We have

$$\begin{aligned}\overline{\theta}^n &= \frac{1}{n}\sum_{m=1}^{n}\hat{\theta}^n = \frac{1}{n}\left(\sum_{m=1}^{n-1}\hat{\theta}^m + \hat{\theta}^n\right) = \frac{1}{n}\left(\frac{n-1}{n-1}\sum_{m=1}^{n-1}\hat{\theta}^m + \hat{\theta}^n\right) \\ &= \frac{n-1}{n}\overline{\theta}^{n-1} + \frac{1}{n}\hat{\theta}^n \\ &= \overline{\theta}^{n-1} + \frac{1}{n}(\hat{\theta}^n - \overline{\theta}^{n-1})\,.\end{aligned}$$

From this expression we see that

$$\overline{\theta}^{n-1} - \overline{\theta}^n = \frac{1}{n}(\overline{\theta}^{n-1} - \hat{\theta}^n)\,.$$

This can be used in the third term in the expression thus derived for $\mathrm{Var}(\hat{\theta}^n)$ where we find

$$\begin{aligned}\mathrm{Var}\left(\hat{\theta}^n\right) &= \frac{n-2}{n-1}\mathrm{Var}\left(\hat{\theta}^{n-1}\right) + \frac{(\overline{\theta}^{n-1} - \hat{\theta}^n)^2}{n-1} + \frac{2}{(n-1)n}(\overline{\theta}^{n-1} - \hat{\theta}^n)(\hat{\theta}^n - \overline{\theta}^{n-1}) + \left(\frac{n}{n-1}\right)\frac{1}{n^2}(\overline{\theta}^{n-1} - \hat{\theta}^n)^2 \\ &= \left(\frac{n-2}{n-1}\right)\mathrm{Var}(\hat{\theta}^{n-1}) + \left(\frac{1}{n-1}\right)(\overline{\theta}^{n-1} - \hat{\theta}^n)^2 + \frac{1}{n(n-1)}\left[-(\overline{\theta}^{n-1} - \hat{\theta}^n)^2\right] \\ &= \left(\frac{n-2}{n-1}\right)\mathrm{Var}(\hat{\theta}^{n-1}) + \frac{(\overline{\theta}^{n-1} - \hat{\theta}^n)^2}{n-1}\left(1 - \frac{1}{n}\right) \\ &= \left(\frac{n-2}{n-1}\right)\mathrm{Var}(\hat{\theta}^{n-1}) + \frac{1}{n}(\overline{\theta}^{n-1} - \hat{\theta}^n)^2\,.\end{aligned}$$

as we were to show when we recall that $\mathrm{Var}(\hat{\theta}^n) \equiv (\hat{\sigma}^2)^n$.

Problem 8.6

WWX: do this problem.

Problem 8.7

WWX: do this problem.