Problem 1

1a

Cost functional: $J(x,u) = \int_0^1 x^2u^2\,dt + [x(1)]^2$
Value function: $V(t,x) = \min_u \left(\int_t^1 x^2u^2\,dt + [x(1)]^2\right)$
HJB Equation: $-V_t(t,x) = \min_u \left\{x^2u^2 + V_x(t,x)(-xu)\right\}$
Define the Hamiltonian-like expression

$$H(t,x,u) = x^2u^2 - xuV_x$$

Derivative with respect to u, set to zero:

$$\frac{\partial H}{\partial u} = 2x^2u - xV_x = 0$$

Solve for u:

$$u^* = \frac{V_x}{2x}$$

Final Result is:

$$\boldsymbol{u^*(t) = \frac{V_x(t,x)}{2x}}$$

1b

Let's assume that the Value function takes the quadratic form of: $V(t,x) = p(t)x^2$.
The HJB equation from part (a) was: $-V_t(t,x) = \min_u \left\{x^2u^2 + V_x(t,x)(-xu)\right\}$
And, the optimal control was:

$$u^* = \frac{V_x}{2x}$$

Now, we plug in $V(t,x) = p(t)x^2$:
Compute Partial Derivatives: $V_t = \dot{p}(t)x^2$,$V_x = 2p(t)x$
Via plugging into HJB

$$-\dot{p}(t)x^2 = \min_u \left\{x^2u^2 - xu(2p(t)x)\right\}$$

$$-\dot{p}(t)x^2 = \min_u \left\{x^2u^2 - 2p(t)x^2u\right\}$$

Minimize RHS w.r.t. u:

$$x^2u^2 - 2p(t)x^2u = x^2(u^2 - 2p(t)u)$$

Minimizing $u^2 - 2p(t)u$ gives:

$$\frac{d}{du}(u^2 - 2p(t)u) = 2u - 2p(t) = 0 \Rightarrow u^* = p(t)$$

Plug $u^* = p(t)$ back into RHS:

$$-\dot{p}(t)x^2 = x^2(p(t)^2 - 2p(t)^2) = -x^2p(t)^2$$

Divide both sides by x^2:

$$-\dot{p}(t) = -p(t)^2 \Rightarrow \dot{p}(t) = p(t)^2$$

Solve the ODE for $p(t)$:

$$\frac{dp}{dt} = p^2 \Rightarrow \frac{1}{p^2}dp = dt$$

Integrate both sides:

$$-\frac{1}{p} = t + C \Rightarrow p(t) = \frac{-1}{t+C}$$

BY using the terminal condition: $V(1,x) = x^2 \Rightarrow p(1) = 1$

$$p(1) = \frac{-1}{1+C} = 1 \Rightarrow -1 = 1 + C \Rightarrow C = -2$$

Thus:

$$\boldsymbol{p(t) = \frac{-1}{t-2}}$$

Final results are:
Value function:

$$\boldsymbol{V(t,x) = \frac{-x^2}{t-2}}$$

Optimal control:

$$\boldsymbol{u^*(t) = p(t) = \frac{-1}{t-2}}$$

Optimal objective function value: At $t = 0, x(0) = 1$, so:

$$\boldsymbol{J^* = V(0,1) = \frac{-1}{-2} = \frac{1}{2}}$$

Problem 2

2a

Table 1: Enumeration Elapsed Time and Cost for Different Grid Sizes ν

ν	Elapsed Time (s)	Enumeration Cost (J)
2	0.0162	2.2707
3	0.0105	2.2707
4	0.1831	1.7937
5	0.1708	1.8575
6	1.7198	1.7457
7	1.6851	1.7331
8	9.5698	1.7225
9	9.5206	1.7045
10	39.1859	1.7097
11	39.6406	1.7065
12	128.1371	1.7001

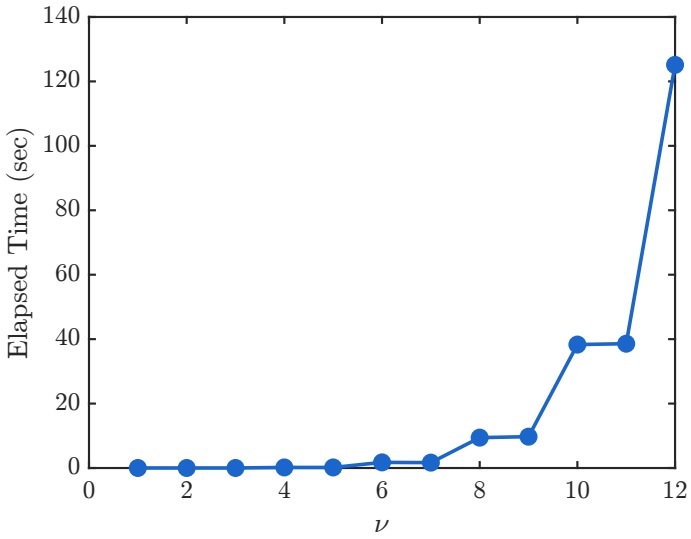

Figure 1: Computation Time for Enumeration-Based Optimal Control

- I varied ν from 2 to 12. The corresponding elapsed times and costs are summarized in Table 1 and visually presented in Figure 1. As ν increases, the computational time grows significantly.
- Based on the observed results I would say, the computational cost becomes prohibitive at around $\nu = [7, 8]$ or higher, where the computation time exceeds 10 seconds. It is clear that at $\nu = 12$, it already takes more than 2 minutes, which makes it impractical for further increases.
- In case there was more than one control input, the number of possible control sequences would grow exponentially. For instance, for m inputs:

  $$\text{Total sequences} = \nu^{m(nt-1)}$$

  So, a small increase in the number of inputs would cause an explosion in the number of simulations, quickly making the enumeration approach infeasible.

2b

Table 2: NDP Elapsed Time and Cost for Varying ν and n_x

ν	n_x	Elapsed Time (s)	Cost J
3	9	0.0157	5.5810
3	15	0.0134	2.4628
3	21	0.0156	2.4944
5	9	0.0045	5.5810
5	15	0.0175	3.4627
5	21	0.0159	2.8003
7	9	0.0037	5.5810
7	15	0.0098	3.9653
7	21	0.0160	4.3777

In this scenario I changed $\nu \in \{3, 5, 7\}$ and $n_x \in \{9, 15, 21\}$. Table 2 above summarizes the resulting computation time and objective cost for each setting. Generally, I can report that increasing n_x improves the cost function due to better resolution in the state space, while increasing ν shows diminishing or inconsistent returns. Moreover, for this low-dimensional case, runtime remained fast (under 0.02 seconds).

2c

Table 3: Comparison of Objective Cost and Runtime for Different Methods and Grid Sizes ν

ν	Enumeration-Based Approach		Numerical Dynamic Programming		Quadratic Programming	
	Cost J	Time (s)	Cost J	Time (s)	Cost J	Time (s)
1	2.5488	0.003	2.5488	0.021	1.6906	0.001
2	2.2707	0.006	2.4944	0.019	1.6906	0.001
3	2.2707	0.007	2.4944	0.017	1.6906	0.001
4	1.7937	0.180	3.2535	0.019	1.6906	0.001
5	1.8575	0.184	2.8003	0.013	1.6906	0.002
6	1.7457	1.683	1.9540	0.014	1.6906	0.000
7	1.7331	1.664	4.3777	0.023	1.6906	0.000
8	1.7225	9.681	3.9104	0.016	1.6906	0.000
9	1.7045	9.793	3.5026	0.015	1.6906	0.000

Table 3 compares 3 different methods.

- The QP method provided the lowest cost for the randomly chosen all ν values. Enumeration improved with higher ν but never matched QP (I would say it probably will get closer, but it takes a lot of time, which is not efficient). Dynamic Programming (DP) showed kind of flexible, higher and more variable costs.
- The reason why Enumeration differs from QP is that it only evaluates a finite set of control values. If the true optimal control lies between grid points, it can't be reached. However, as I mentioned, accuracy improves with larger ν, but the method becomes way more computationally expensive.
- DP Differs from QP with its usage: DP uses discretized grids and nearest-neighbor lookup, which in turn causes the approximation error. The value function is not continuous, so the solution is only an estimate.
- Instead of dynamic programming solution, I would consider finer grids, applying interpolation instead of nearest-neighbor, and employing function approximation or continuous dynamic programming techniques to better represent the value function.

2d

Control Bounds

- **Enumeration-Based Approach:** Easy to handle. Control bounds are enforced by defining the control grid `Us`.
- **Numerical Dynamic Programming:** Also easy to handle since it is straightforward. The control grid `Us` is predefined within bounds, so values outside the range are never considered.

- **Quadratic Programming:** Supports control bounds. Most solvers (e.g., `quadprog`) allow to specify lower and upper bounds directly using `lb` and `ub`.

State Bounds

- **Enumeration-Based Approach:** Difficult to enforce. Each control sequence must be simulated to check for violations and that makes the process slower and impractical .
- **Numerical Dynamic Programming:** Can be enforced by excluding state grid points outside the allowable range, but that makes the process more complex.
- **Quadratic Programming:** Supports state bounds via linear inequality constraints across the horizon. This results in a constrained QP, which remains convex. Most modern solvers can handle these constraints efficiently, although the problem size increases.