MA5232 Assignment 2

Nguyen Ngoc Khanh - A0275047B

May 20, 2025

Contents

1 INTRODUCTION

Consider the *controlled* ODE

$$\dot{x}(t) = f(t, x(t), u(t)) = A(t)x(t) + B(t)u(t)$$

for $A(t) \in \mathbb{R}^{d\times d}$, $B(t) \in \mathbb{R}^{d\times m}$, $u(t) \in \mathbb{R}^m$ and $t \in [0, T]$ where $u \in L^\infty([0, T], \mathbb{R}^m)$ is the control signal.

1.1 COST FUNCTIONAL

Consider the terminal cost

$$\Phi(x) = x^T M x$$

for some symmetric positive definite matrix $M \in \mathbb{R}^{d\times d}$, then gradient of terminal cost is $(\nabla_x \Phi)(x) = 2Mx$. Consider the running cost

$$L(t, x, u) = x^T Q(t)x + u^T R(t)u$$

for some collection of symmetric positive definite matrices $\{Q(t), R(t) : t \in [0, T]\}$

1.2 THE LQR PROBLEM

The linear quadratic regulator problem is formulated by minimizing the cost functional subject to the dynamic of the *controlled* ODE described above. More precisely,

$$\min_{u\in L^\infty([0,T],\mathbb{R}^m)} \left\{ \Phi(x(t)) + \int_0^T (x^T Q(t)x(t) + u(t)^T R(t)u(t))dt \right\}$$

subject to $\dot{x}(t) = f(t, x(t), u(t)) = A(t)x(t) + B(t)u(t)$ and $x(0) = x_0$

2 NECCESSARY CONDITION VIA PONTRYAGIN MAXIMUM PRINCIPLE (PMP)

2.1 HAMILTONIAN

The Hamiltonian is of the problem is

$$\begin{aligned}
&H(t, x, p, u) \\
&= p^T f(t, x, u) - L(t, x, u) \\
&= p^T A(t)x + p^T B(t)u - x^T Q(t)x - u^T R(t)u
\end{aligned}$$

where $p(t) \in \mathbb{R}^d$ is the costate, then then gradients of H with respect to its variables are

$$\begin{aligned}
(\nabla_p H)(t, x, p, u) &= \nabla_p(p^T (A(t)x + B(t)u)) = A(t)x + B(t)u \\
(\nabla_x H)(t, x, p, u) &= \nabla_x(p^T A(t)x - x^T Q(t)x) = A(t)^T p - 2Q(t)x \\
(\nabla_u H)(t, x, p, u) &= \nabla_u(p^T B(t)u - u^T R(t)u) = B(t)^T p - 2R(t)u
\end{aligned}$$

2.2 STATE AND COSTATE EQUATION AND HAMILTONIAN MAXIMAL CONDITION

If $u^*(t)$ is the optimal control, $x^*(t)$ and $p^*(t)$ are the correspond state and costate trajectory, then the state equation is

$$\begin{aligned}\dot{x}^*(t) &= (\nabla_p H)(t, x^*(t), p^*(t), u^*(t)) \\ &= A(t)x^*(t) + B(t)u^*(t)\end{aligned}$$

subject to $x^*(0) = x_0$, the costate equation is

$$\begin{aligned}\dot{p}^*(t) &= -(\nabla_x H)(t, x^*(t), p^*(t), u^*(t)) \\ &= -A(t)^T p^*(t) + 2Q(t)x^*(t)\end{aligned}$$

subject to $p^*(T) = -(\nabla_x \Phi)(x^*(T)) = -2Mx^*(T)$, and Hamiltonian maximal condition is

$$H(t, x^*(t), p^*(t), u^*(t)) \geq H(t, x^*(t), p^*(t), u(t))$$

for any control $u(t)$ for almost every $t \in [0, T]$

2.3 SHOW $u^*(t) = \frac{1}{2}R^{-1}(t)B^T(t)p^*(t)$

If the optimal control exists, assume all functions are sufficiently smooth, the control set is closed, we have

$$(\nabla_u H)(t, x^*(t), p^*(t), u^*(t)) = B(t)^T p^*(t) - 2R(t)u^*(t) = 0$$

for all $t \in [0, T]$. Since $R(t)$ is symmetric positive definite, $u^*(t) = \frac{1}{2}R^{-1}(t)B^T(t)p^*(t)$.
We can assume that $x^*(t) \neq 0$ at all time $t \in [0, T]$. Then, there exists a linear map, namely $L(t) : \mathbb{R}^d \to \mathbb{R}^d$ so that $p^*(t) = L(t)x^*(t)$. Since both x^* and p^* are sufficiently smooth function, there exists a sufficiently function $L : [0, T] \to M_d[\mathbb{R}]$ where $M_d[\mathbb{R}]$ denotes the space of $d \times d$ matrices over $\mathbb{R}$

3 LINEAR RELATIONSHIP OF $p^*(t)$ ON $x^*(t)$

From now on, all functions are over variable t, we will drop [1] the notation $A(t), x^*(t)$ and simply write it as A, x
Set $p = -2Px$ for some $P : [0, T] \to M_d[\mathbb{R}]$. Then

$$\dot{p} = -2\dot{P}x - 2P\dot{x}$$

3.1 RICATTI DIFFERENTIAL EQUATION (RDE)

Recall state and costate equation

$$\begin{aligned}\dot{x} &= Ax + Bu \\ \dot{p} &= -A^T p + 2Qx\end{aligned}$$

[1] sorry, I have difficulties reading too many (t)s and $*s$

subject to $x(0) = x_0$, $p(T) = -2Mx(T)$. Substitute $u = \frac{1}{2}R^{-1}B^Tp$, we have

$$\dot{x} = Ax + \frac{1}{2}BR^{-1}B^Tp$$
$$\dot{p} = -A^Tp + 2Qx$$

Substitute $p = -2Px$, and $\dot{p} = -2\dot{P}x - 2P\dot{x}$, we have

$$\dot{x} = Ax - BR^{-1}B^TPx$$
$$-2\dot{P}x - 2P\dot{x} = 2A^TPx + 2Qx$$

That induces an equation

$$-2\dot{P}x - 2P(Ax - BR^{-1}B^TPx) = 2A^TPx + 2Qx$$

which simplifies into

$$(-PA - A^TP - Q + PBR^{-1}B^TP - \dot{P})x = 0$$

We can assume that $x(t)$ spans the whole $\mathbb{R}^d$, so

$$-PA - A^TP - Q + PBR^{-1}B^TP = \dot{P}$$

subject to $p(T) = -2Mx(T) \iff P(T) = M$

3.2 IF $P := -\frac{1}{2}YX^{-1}$ THEN P SATISFIES RDE

Let $X, Y : [0, T] \to M_d[\mathbb{R}]$ satisfying

$$\frac{d}{dt}\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} A & \frac{1}{2}BR^{-1}B^T \\ 2Q & -A^T \end{bmatrix}\begin{bmatrix} X \\ Y \end{bmatrix} \text{ and } \begin{bmatrix} X(T) \\ Y(T) \end{bmatrix} = \begin{bmatrix} I \\ -2M \end{bmatrix}$$

That is equivalent to the system of equations

$$\dot{X} = AX + \frac{1}{2}BR^{-1}B^TY$$
$$\dot{Y} = 2QX - A^TY$$

We assume that X is invertible for all $t \in [0, T]$. Let $P = -\frac{1}{2}YX^{-1}$, then $PX = -\frac{1}{2}Y$, differentiate both sides

$$\dot{P}X + P\dot{X} = -\frac{1}{2}\dot{Y}$$

So

$$\begin{aligned}
\dot{P} &= -\frac{1}{2}\dot{Y}X^{-1} - P\dot{X}X^{-1} \\
&= -\frac{1}{2}(2QX - A^TY)X^{-1} - P\left(AX + \frac{1}{2}BR^{-1}B^TY\right)X^{-1} \\
&= -Q + \frac{1}{2}A^TYX^{-1} - PA - \frac{1}{2}PBR^{-1}B^TYX^{-1} \\
&= -Q - A^TP - PA + PBR^{-1}B^TP
\end{aligned}$$

Moreover, $P(T) = -\frac{1}{2}Y(T)X(T)^{-1} = M$ which is precisely the description for RDE problem, so any X, Y satisfies the above equations also solve RDE.

4 HAMILTON-JACOBI-BELLMAN FRAMEWORK

4.1 HAMILTON-JACOBI-BELLMAN EQUATION

For any $s, \tau \in [0, T]$ with $s \leq \tau$ and some $z \in \mathbb{R}^d$, let $V(s, z)$ be the optimal cost travelling from the state $(t, x) = (s, z)$, more precisely,

$$V(s,z) = \inf_u \left\{ \int_s^\tau L(t, x(t), u(t))dt + V(\tau, x(\tau)) \right\}$$

where x solves the ODE $\dot{x}(t) = f(t, x(t), u(t))$ on $t \in [s, \tau]$ with $x(s) = z$. The corresponding Hamilton-Jacobi-Bellman (HJB) equation is

$$\nabla_s V(s,z) + \inf_u \{L(s,z,u) + [\nabla_z V(s,z)]^T f(s,z,u)\} = 0$$

subject to condition $V(T, z) = \Phi(z)$. In our problem, $V(s, z)$ is of the form

$$V(s,z) = \inf_u \left\{ \int_s^\tau (x(t)^T Q(t)x(t) + u(t)^T R(t)u(t))dt + V(\tau, x(\tau)) \right\}$$

and the HJB equation is of the form (here we change the variable names from (s, z) to (t, x) for consistency with the problem statement)

$$\nabla_t V(t,x) + \inf_u \{x^T Q(t)x + u^T R(t)u + [\nabla_x V(t,x)]^T (A(t)x + B(t)u)\} = 0$$

subject to $V(T, x) = x^T M x$.

4.2 SIMPLIFY HJB EQUATION

We can simplify HJB equation into

$$-\nabla_t V(t,x) = x^T Q(t)x + [\nabla_x V(t,x)]^T A(t)x + \inf_u \{u^T R(t)u + [\nabla_x V(t,x)]^T B(t)u\}$$

As a function of t, x, u, the minimum of $u^T R(t)u + [\nabla_x V(t,x)]^T B(t)u$ must satisfy

$$\nabla_u (u^T R(t)u + [\nabla_x V(t,x)]^T B(t)u) = 0$$

That is equivalent to $2R(t)u(t) + B(t)^T[\nabla_x V(t,x)] = 0$, hence

$$u(t) = -\frac{1}{2}R(t)^{-1}B(t)^T[\nabla_x V(t,x)]$$

Substitute $u(t)$ into $u^T R(t)u + [\nabla_x V(t,x)]^T B(t)u$, note that $R(t)$ is symmetric, so $(R(t)^{-1})^T = (R(t)^T)^{-1} = R(t)^{-1}$, we have

$$u(t)^T R(t)u(t) + [\nabla_x V(t,x)]^T B(t)u(t) = -\frac{1}{4}[\nabla_x V(t,x)]^T B(t)R(t)^{-1}B(t)^T \nabla_x V(t,x)$$

We have the equation for $V(t,x)$

$$-\nabla_t V(t,x) = x^T Q(t)x + [\nabla_x V(t,x)]^T A(t)x - \frac{1}{4}[\nabla_x V(t,x)]^T B(t)R(t)^{-1}B(t)^T \nabla_x V(t,x)$$

4.3 CHOOSE $V(t,x) = x^T P(t)x$

Let $V(t,x) = x^T P(t)x$, then gradient of V is

$$\begin{aligned}\nabla_t V(t,x) &= x^T \dot{P}(t)x \\ \nabla_x V(t,x) &= 2P(t)x\end{aligned}$$

Now if we assume that $P(t)$ solves RDE

$$-PA - A^T P - Q + PBR^{-1}B^T P = \dot{P}$$

with $P(T) = M$. Note that, if $P(t)$ solves RDE, $P(t)^T$ also solves RDE. By **uniqueness assumption** in the problem remark, $P(t)$ must be symmetric, so $x^T P(t)A(t)x = x^T A(t)^T P(t)^T x = x^T A(t)^T P(t)^T x$, we have

$$\begin{aligned}&x^T Q(t)x + [\nabla_x V(t,x)]^T A(t)x - \frac{1}{4}[\nabla_x V(t,x)]^T B(t)R(t)^{-1}B(t)^{-1}\nabla_x V(t,x) \\ &= x^T Q(t)x + [2P(t)x]^T A(t)x - \frac{1}{4}[2P(t)x)]^T B(t)R(t)^{-1}B(t)^T 2P(t)x \\ &= -x^T(-Q(t) - 2P(t)^T A(t) + P(t)^T B(t)R(t)^{-1}B(t)^T P(t))x \\ &= -x^T(-Q(t) - P(t)A(t) - A(t)P(t) + P(t)^T B(t)R(t)^{-1}B(t)^T P(t))x \\ &= x^T \dot{P}(t)x\end{aligned}$$

We can assume that x spans the whole space $\mathbb{R}^d$, so if $P(t)$ solves RDE, then $P(t)$ also solves HJB

5 NUMERICAL SOLUTION

Consider $(x(t), v(t)) \in \mathbb{R}^2$ satisfies the system of equation

$$\begin{aligned}\dot{x}(t) &= v(t) \\ \dot{v}(t) &= -\alpha(t)v(t) + u(t)\end{aligned}$$

with initial condition $x(0) = 1$ and $v(0) = 0$. We want to control $u(t) \in \mathbb{R}$ so that at time $t = 1$, $x(1)$ is close to the origin. The terminal cost is $\Phi(x) = x^2$ and the running cost is $L(t,(x,v),u) = \lambda u(t)^2$. The optimal control problem is

$$\min_{u \in L^\infty([0,1],\mathbb{R})} J[u] = \min_{u \in L^\infty([0,1],\mathbb{R})} \left\{ x(1)^2 + \lambda \int_0^1 u(t)^2 dt \right\}$$

subject to the dynamic and initial condition.

5.1 SOLVE NUMERICALLY

5.1.1 RDE METHOD

The RDE equation is of the form $P : [0, 1] \to M_d[\mathbb{R}]$

$$\dot{P} = -PA - A^T P - Q + PBR^{-1}B^T P$$

subject to $P(T) = M$. In our problems, $d = 2$ and

$$A(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha(t) \end{bmatrix}, B(t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, Q(t) = 0, R(t) = \lambda, M = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Let the symmetric matrix $P(t) = \begin{bmatrix} a(t) & b(t) \\ b(t) & c(t) \end{bmatrix}$, then RDE can be simplied into

$$\begin{bmatrix} \dot{a} & \cdot b \\ \dot{b} & \cdot c \end{bmatrix} = -\begin{bmatrix} 0 & a - b\alpha \\ 0 & b - c\alpha \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ a - b\alpha & b - c\alpha \end{bmatrix} + \lambda^{-1}\begin{bmatrix} b^2 & bc \\ bc & c^2 \end{bmatrix}$$

subject to $a(1) = 1, b(1) = 0, c(1) = 0$ which simplies to a system of three equations

$$\dot{a} = \lambda^{-1}b^2$$
$$\dot{b} = -a + b\alpha + \lambda^{-1}bc$$
$$\dot{c} = -2b + 2c\alpha + \lambda^{-1}c^2$$

subject to $a(1) = 1, b(1) = 0, c(1) = 0$. After solving for $P(t)$, we can construct

$$V(t, (x, v)) = (x, v)P(t)(x, v)^T = ax^2 + 2bxv + cv^2$$

Then get $\min_{u \in L^\infty([0,1],\mathbb{R})} J[u] = V(0, (x(0), v(0))) = a(0)$

5.1.2 SEMI-IMPLICIT SCHEME FOR RDE METHOD

We use semi-implicit scheme to solve the ODE numeriacally as follows: Let time t admit only discrete values

$$\mathcal{T} = \{t_0 = 0, t_1 = h, t_2 = 2h, ..., t_N = Nh = 1\}$$

for some $N \in \mathbb{N}$ and $h = \frac{1}{N}$. Let a_n, b_n, c_n are the corresponding values of $a(t_n), b(t_n), c(t_n)$. Let $\alpha_n = \alpha(t_n)$. We have the following semi-implicit relation

$$\frac{a_n - a_{n-1}}{h} = \lambda^{-1}b_n^2$$
$$\frac{b_n - b_{n-1}}{h} = -a_n + b_{n-1}\alpha_n + \lambda^{-1}b_{n-1}c_n$$
$$\frac{c_n - c_{n-1}}{h} = -2b_n + 2c_{n-1}\alpha_n + \lambda^{-1}c_{n-1}^2$$

for $n = 0, 1, ..., N$ and $h = 1/N$ with the initial condition $a_N = 1, b_N = 0, c_N = 0$.

5.1.3 HJB SHOOTING METHOD

Let $p(t), q(t) : [0,1] \to \mathbb{R}$ be the corresponding costates, Then, Hamiltonian is

$$H(t,x,v,p,q,u) = pv - \alpha(t)qv + qu - \lambda u^2$$

Then the optimal u for H can be obtained analytically

$$u = \frac{q}{2\lambda}$$

Let $\xi(q) : \mathbb{R} \to \mathbb{R}$ defined by $\xi(q) = q/2\lambda$. If p, q are optimal, costates must satisfy

$$\begin{bmatrix} \dot{p}(t) \\ \dot{q}(t) \end{bmatrix} = -\nabla_{x,v} H(t,x,v,p,q) = -\begin{bmatrix} 0 & 1 \\ 0 & -\alpha(t) \end{bmatrix}^T \begin{bmatrix} p(t) \\ q(t) \end{bmatrix}$$

with boundary conditions $\begin{bmatrix} p(1) \\ q(1) \end{bmatrix} = -\nabla\Phi(x(1), v(1)) = -2M \begin{bmatrix} x(1) \\ v(1) \end{bmatrix} = \begin{bmatrix} -2x(1) \\ 0 \end{bmatrix}$
together we have a system of equations

$$\begin{aligned}
\dot{x}(t) &= v(t) \\
\dot{v}(t) &= -\alpha(t)v(t) + \xi(q(t))) \\
\dot{p}(t) &= 0 \\
\dot{q}(t) &= p(t) - \alpha(t)q(t)
\end{aligned}$$

with the boundary conditions $x(0) = 1, v(0) = 0, p(1) = -2x(1), q(1) = 0$. Consider the function $g : \mathbb{R}^2 \to \mathbb{R}^2$ as follows:

> Let $(p_0, q_0) \in \mathbb{R}^2$, let the system of equation have the initial conditions $x(0) = 1, v(0) = 0, p(0) = p_0, q(0) = q_0$, then the dynamic results $x(1), v(1), p(1), q(1)$. Output $g(p_0, q_0) = (p(1) - 2x(1), q(1)) \in \mathbb{R}^2$

The root of this function is a pair (p_0, q_0) so that $(p(1) - 2x(1), q(1)) = 0$. Finding the root is equivalent to solving the optimal control problem since we can calculate $J[u]$ from $u(t) = \xi(q(t))$

$$J[u] = x(1)^2 + \lambda \int_0^1 u(t)^2 dt$$

5.2 IMPLEMENTATION AND COMPARISON

We implemented three methods

1. **RDE LSODE**: solve RDE using *LSODA* stiff ODE solver in *scipy*
2. **RDE Implicit**: solve RDE using semi-implicit method 5.1.2
3. **HJB Shooting**: solve HJB using shooting method with *LSODA* stiff ODE solver and *MINPACK*'s *hybrd* root solver in *scipy*

All three methods used $N = 1000$ evaluation steps

5.2.1 $\lambda = 1$

When $\lambda = 1$, all three methods were able to produce a result for optimal control for both $\alpha(t) = \sin(10t)$ or $\alpha(t) = t^2$ and the optimal cost are close together. **RDE LSODE** and **RDE Implicit** produced different $b(t)c(t)$ at t near 0 even though they were solving the same ODE.

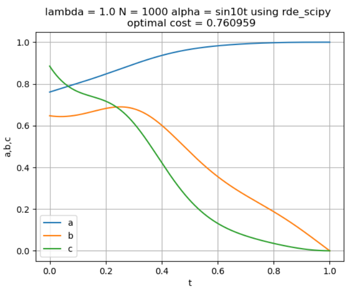

Figure 1: RDE LSODE

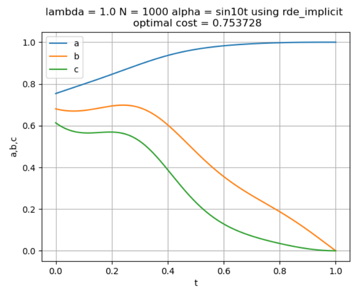

Figure 2: RDE Implicit

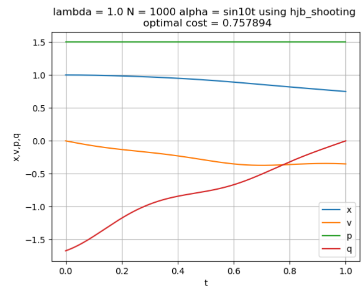

Figure 3: HJB Shooting

Figure 4: $\lambda = 1, \alpha(t) = \sin(10t)$

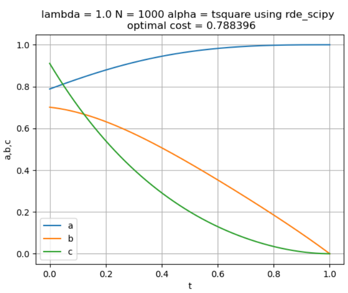

Figure 5: RDE LSODE

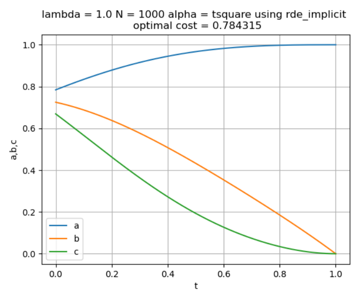

Figure 6: RDE Implicit

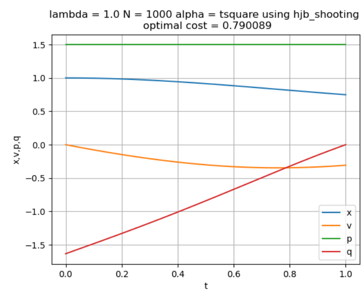

Figure 7: HJB Shooting

Figure 8: $\lambda = 1, \alpha(t) = t^2$

5.2.2 $\lambda = 0.1$

When $\lambda = 0.1$, it is *very easy* to control x by u since it does not induce too much cost, the optimal cost is lower. However, **RDE LSODE** failed to produce the result.

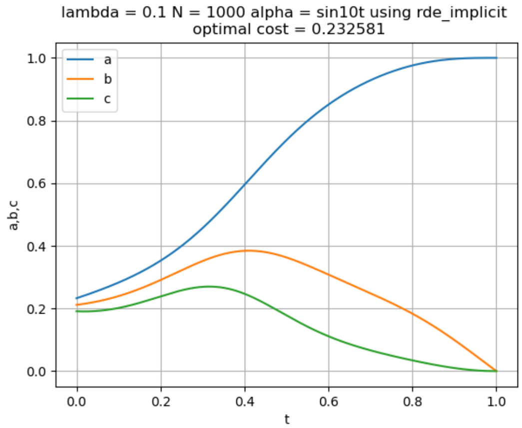

Figure 9: RDE Implicit

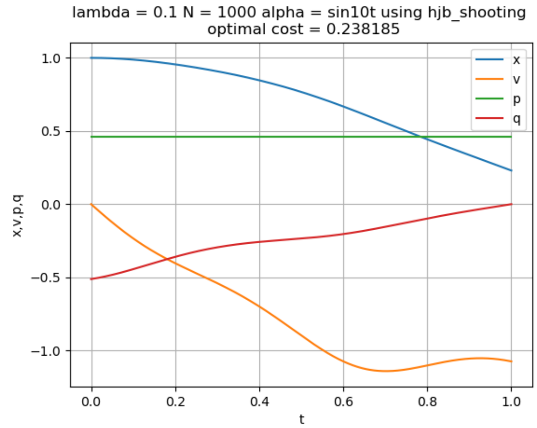

Figure 10: HJB Shooting

Figure 11: $\lambda = 0.1, \alpha(t) = \sin(10t)$

5.2.3 $\lambda = 10$

When $\lambda = 10$, it is *very hard* to control x by u since it does induce a lot of cost for a positive value of u, the optimal cost is closed to 1 which is the case when $u(t) = 0$. all three methods were able to produce a result for optimal control.

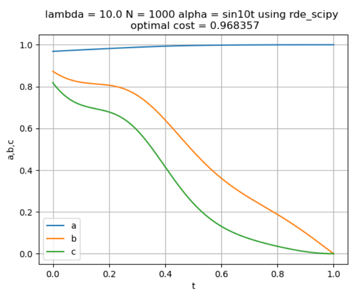

Figure 12: RDE LSODE

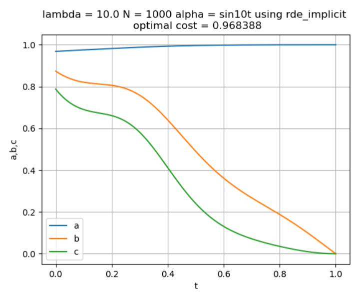

Figure 13: RDE Implicit

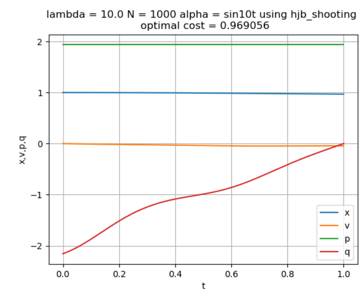

Figure 14: HJB Shooting

Figure 15: $\lambda = 10, \alpha(t) = \sin(10t)$

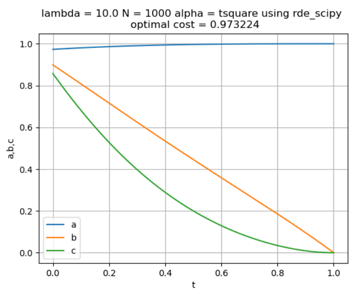

Figure 16: RDE LSODE

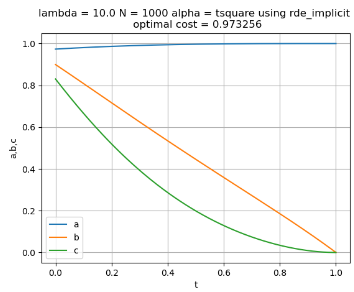

Figure 17: RDE Implicit

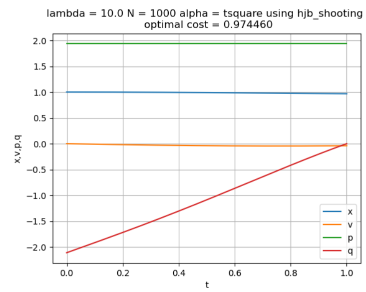

Figure 18: HJB Shooting

Figure 19: $\lambda = 10, \alpha(t) = t^2$

6 CODE

```
from typing import Callable
import numpy as np
from matplotlib import pyplot as plt

def rde_implicit_solver(N: int, l: float, alpha: Callable[[float], float]) -> list[np.ndarray]:
    from tqdm import tqdm
    """
    Implicitly solve the ODE system using the implicit Euler method.
    """
    inv_l = 1 / l
    h = 1 / N

    def get_t_n(n: int) -> float:
        return n * h

    y = np.empty((3, N+1), dtype=float)
    y[:, N] = (1, 0, 0) # initial condition
    # integrate backwards
    for n in tqdm(range(N-1, -1, -1), desc="integrating ...", unit="step"):
        # implicit Euler step
        a_n, b_n, c_n = y[:, n+1]
        t_n = get_t_n(n)

        a_n_1 = a_n - h * (inv_l * b_n**2)
        b_n_1 = (b_n + h * a_n) / (1 + h * (alpha(t_n) + inv_l * c_n))

        # solve for c_n_1
        A = h * inv_l
        B = 1 + 2 * h * alpha(t_n) # always nonnegative
        C = - c_n - 2 * h * b_n
        D = B ** 2 - 4 * A * C
        c_n_1 = (- B + np.sqrt(D)) / (2 * A) # we want the positive root

        y[:, n] = (a_n_1, b_n_1, c_n_1)

    t = np.array([get_t_n(n) for n in range(N+1)])
    a, b, c = y[0, :], y[1, :], y[2, :]
    return t, a, b, c

def rde_scipy_solver(N: int, l: float, alpha: Callable[[float], float]) -> list[np.ndarray]:
    from scipy.integrate import solve_ivp
    """
    Solve the ODE system using scipy"s solve_ivp.
    """
    def f(t: float, y: np.ndarray) -> np.ndarray:
        inv_l = 1 / l
        a, b, c = y
        a_dot = inv_l * b**2
        b_dot = -a + b * alpha(t) + inv_l * b * c
        c_dot = - 2 * b + 2 * c * alpha(t) - inv_l * c**2
        return np.array([a_dot, b_dot, c_dot])
```

```python

    sol = solve_ivp(
        fun=f,
        y0=np.array([1, 0, 0]),
        t_span=[1, 0],
        t_eval=np.linspace(1, 0, N),
        method="LSODA",  # stiff solver
    )

    if not sol.success:
        raise ValueError(sol.message)


    t, y = sol.t, sol.y
    a, b, c = y[0, :], y[1, :], y[2, :]
    return t, a, b, c

def hjb_shooting_solver(N: int, l: float, alpha: Callable[[float], float]) -> list[np.ndarray]:
    """
    Solve the HJB equation using the shooting method.
    """
    from scipy.integrate import solve_ivp
    from scipy.optimize import root
    def g(t: float, y: np.ndarray) -> np.ndarray:
        x, v, p, q = y
        u = q / (2 * l)

        x_dot = v
        v_dot = -alpha(t) * v + u
        p_dot = 0
        q_dot = p - alpha(t) * q
        return np.array([x_dot, v_dot, p_dot, q_dot])

    def f(z: np.ndarray) -> float:
        p_0, q_0 = z
        x_0, v_0 = 1, 0

        sol = solve_ivp(
            fun=g,
            y0=np.array([x_0, v_0, p_0, q_0]),
            t_span=[0, 1],
            t_eval=np.linspace(0, 1, N),
            method="LSODA", # stiff solver
        )

        if not sol.success:
            raise ValueError(sol.message)

        x_1, v_1, p_1, q_1 = sol.y[:, -1]
        return np.array([p_1 - 2 * x_1, q_1]), sol.t, sol.y

    p_0, q_0 = 0, 0 # initial guess for p, q
    sol = root(
        fun=lambda z: f(z)[0],
```

```python
        x0=np.array([p_0, q_0]),
        method="hybr", # default solver in scipy
    )
    if not sol.success:
        raise ValueError(sol.message)
    p_0, q_0 = sol.x
    # Re-run the ODE with the found p_0, q_0
    _, t, y = f(np.array([p_0, q_0]))
    x, v, p, q = y

    return t, x, v, p, q

def get_args():
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("--l", type=float, default=1.0, help="Lambda value")
    parser.add_argument("--N", type=int, default=1000, help="Number of steps")
    parser.add_argument("--method", type=str, default="rde_implicit", choices=["rde_implicit", "
    rde_scipy", "hjb_shooting"], help="method to use")
    parser.add_argument("--alpha", type=str, default="sin10t", choices=["sin10t", "tsquare"],
    help="function to use for alpha")
    return parser.parse_args()

args = get_args()
l = args.l
N = args.N
if l <= 0:
    raise ValueError("l must be positive")
if N <= 0:
    raise ValueError("N must be positive")
if N < 2:
    raise ValueError("N must be at least 2")

solver = {
    "rde_implicit": rde_implicit_solver,
    "rde_scipy": rde_scipy_solver,
    "hjb_shooting": hjb_shooting_solver,
}[args.method]

if args.alpha == "sin10t":
    alpha = lambda t: float(np.sin(10 * t))
elif args.alpha == "tsquare":
    alpha = lambda t: t**2
else:
    raise ValueError("Invalid alpha function")


if args.method in ["rde_implicit", "rde_scipy"]:
    t, a, b, c = solver(
        N=N,
        l=l,
        alpha=alpha,
    )

```

```
    sort = np.argsort(t)
    t = t[sort]
    a = a[sort]
    b = b[sort]
    c = c[sort]

    print("optimal cost:", a[0])


    plt.plot(t, a, label="a")
    plt.plot(t, b, label="b")
    plt.plot(t, c, label="c")
    plt.legend()
    plt.xlabel("t")
    plt.ylabel("a,b,c")
    plt.title(f"lambda = {l} N = {N} alpha = {args.alpha} using {args.method} \n optimal cost = {a[0]:.6f}")
    plt.grid()
    plt.show()

if args.method in ["hjb_shooting"]:
    t, x, v, p, q = solver(
        N=N,
        l=l,
        alpha=alpha,
    )
    u = q / (2 * l)
    J = x[-1]**2 + np.trapezoid(l * u**2, t)
    print("optimal cost:", J)
    plt.plot(t, x, label="x")
    plt.plot(t, v, label="v")
    plt.plot(t, p, label="p")
    plt.plot(t, q, label="q")
    plt.legend()
    plt.xlabel("t")
    plt.ylabel("x,v,p,q")
    plt.title(f"lambda = {l} N = {N} alpha = {args.alpha} using {args.method} \n optimal cost = {J:.6f}")
    plt.grid()
    plt.show()
```