Chong Zhen Jie A0201613Y
MA5232: Optimal Control Theory

Project for Part 4

Q4 is implemented in `Python`. The code is in a separate folder, and is also attached to the end of this PDF document.

Question 1

(a) The corresponding Hamiltonian is

$$H(t, x_t, p_t, u_t) = p_t^\top \Big(A(t)x_t + B(t)u_t\Big) - \Big(x_t^\top Q(t)x_t + u_t^\top R(t)u_t\Big)$$

(b) We note that $M, Q(t)$ and $R(t)$ are symmetric matrices.

The state equation is

$$\dot{x}_t^* = \nabla_p H(t, x_t^*, p_t^*, u_t^*) = A(t)x_t^* + B(t)u_t^*, \qquad x^*(0) = x_0 \tag{1.0}$$

The co-state equation is

$$\dot{p}_t^* = -\nabla_x H(t, x_t^*, p_t^*, u_t^*) = -A^\top(t)p_t^* + 2Q(t)x_t^*\,, \tag{1.1}$$

$$p_T^* = -\nabla_x\Big(\Phi(x_T^*)\Big) = -2Mx_T^* \tag{1.2}$$

Expanding $H(t, x_t^*, p_t^*, u_t^*) \geq H(t, x_t^*, p_t^*, u)$, we get

$$p_t^{*\top} B(t)u_t^* - u_t^{*\top} R(t)u_t^* \geq p_t^{*\top} B(t)u - u^\top R(t)u$$

The Hamiltonian maximization condition is

$$u_t^* = \arg\max_u \Big\{p_t^{*\top} B(t)u - u^\top R(t)u\Big\}, \qquad u \in L^\infty\Big([0,T], \mathbb{R}^m\Big) \tag{1.3}$$

(c) By (1.3), we let $\nabla_u \Big(p_t^{*\top} B(t)u - u^\top R(t)u\Big) = 0$. Then, we get

$$\begin{aligned} B^\top(t)p_t^* - 2R(t)u &= 0 \\ 2R(t)u &= B^\top(t)p_t^* \\ u &= \frac{1}{2}R^{-1}(t)B^\top(t)p_t^* \end{aligned} \tag{1.4}$$

where $R^{-1}(t)$ exists because $R(t)$ is positive definite. This is a maximization because

$$\nabla_{uu}\Big(p_t^{*\top} B(t)u - u^\top R(t)u\Big) = -2R(t)$$

where the second partial derivative is negative definite since $R(t)$ is positive definite.

Since an optimal control $\boldsymbol{u}^*$ exists, with $\boldsymbol{x}^*$ being the corresponding optimally controlled trajectory, we have shown from (1.4) that

$$u_t^* = \frac{1}{2} R^{-1}(t) B^\top(t) p_t^* \tag{1.5}$$

where u_t^* is a linear function of p_t^*. Given that p_t^* is a linear function of x_t^*, we have shown that the optimal control u_t^* is a linear function of the state x_t^*.

Question 2

(a) Set the following

$$p_t^* = -2P(t)x_t^* \tag{2.0}$$

By differentiating (2.0) with respect to t,

$$\dot{p}_t^* = -2\dot{P}(t)x_t^* - 2P(t)\dot{x}_t^* \tag{2.1}$$

For LHS of (2.1), we substitute in (1.1), followed by (2.0) to get

$$\begin{aligned}\text{LHS} &= -A^\top(t)p_t^* + 2Q(t)x_t^* \\ &= 2A^\top(t)P(t)x_t^* + 2Q(t)x_t^* \\ &= 2\left(A^\top(t)P(t) + Q(t)\right)x_t^*\end{aligned}$$

For RHS of (2.1), we substitute in (1.0), followed by (1.5) and (2.0) to get

$$\begin{aligned}\text{RHS} &= -2\dot{P}(t)x_t^* - 2P(t)\Big(A(t)x_t^* + B(t)u_t^*\Big) \\ &= -2\dot{P}(t)x_t^* - 2P(t)A(t)x_t^* - 2P(t)B(t)u_t^* \\ &= -2\dot{P}(t)x_t^* - 2P(t)A(t)x_t^* - P(t)B(t)R^{-1}(t)B^\top(t)p_t^* \\ &= -2\dot{P}(t)x_t^* - 2P(t)A(t)x_t^* + 2P(t)B(t)R^{-1}(t)B^\top(t)P(t)x_t^* \\ &= 2\left(-\dot{P}(t) - P(t)A(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)\right)x_t^*\end{aligned}$$

For LHS=RHS of (2.1) to hold,

$$A^\top(t)P(t) + Q(t) = -\dot{P}(t) - P(t)A(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)$$

and by rearranging the terms, we see that the PMP is satisfied if $P(t)$ satisfies

$$\dot{P}(t) = -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t) \tag{2.2}$$

and at $t = T$, we observe from (1.2) and (2.0) that

$$-2Mx_T^* = p_T^* = -2P(T)x_T^*$$

and similarly for this to hold,

$$P(T) = M \tag{2.3}$$

which is the final value of P.

(b) We are given $\dot{X}(t)$ and $\dot{Y}(t)$ such that

$$\begin{pmatrix}\dot{X}(t)\\ \dot{Y}(t)\end{pmatrix} = \frac{d}{dt}\begin{pmatrix}X(t)\\ Y(t)\end{pmatrix} = \begin{pmatrix}A(t) & \frac{1}{2}B(t)R^{-1}(t)B^\top(t)\\ 2Q(t) & -A^\top(t)\end{pmatrix}\begin{pmatrix}X(t)\\ Y(t)\end{pmatrix}$$
$$= \begin{pmatrix}A(t)X(t) + \frac{1}{2}B(t)R^{-1}(t)B^\top(t)Y(t)\\ 2Q(t)X(t) - A^\top(t)Y(t)\end{pmatrix}$$

Define $P(t) \coloneqq -\frac{1}{2}Y(t)X^{-1}(t)$. Then we have

$$-2P(t)X(t) = Y(t) \tag{2.4}$$

By differentiating (2.4) with respect to t, we get

$$-2\dot{P}(t)X(t) - 2P(t)\dot{X}(t) = \dot{Y}(t) \tag{2.5}$$

For LHS of (2.5), we substitute in $\dot{X}(t)$, followed by (2.4) to get

$$\begin{aligned}\text{LHS} &= -2\dot{P}(t)X(t) - 2P(t)\left(A(t)X(t) + \frac{1}{2}B(t)R^{-1}(t)B^\top(t)Y(t)\right)\\ &= -2\dot{P}(t)X(t) - 2P(t)A(t)X(t) - P(t)B(t)R^{-1}(t)B^\top(t)Y(t)\\ &= -2\dot{P}(t)X(t) - 2P(t)A(t)X(t) + 2P(t)B(t)R^{-1}(t)B^\top(t)P(t)X(t)\\ &= 2\left(-\dot{P}(t) - P(t)A(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)\right)X(t)\end{aligned}$$

For RHS of (2.5), we substitute in $\dot{Y}(t)$, followed by (2.4) to get

$$\begin{aligned}\text{RHS} &= 2Q(t)X(t) - A^\top(t)Y(t)\\ &= 2Q(t)X(t) + 2A^\top(t)P(t)X(t)\\ &= 2\left(Q(t) + A^\top(t)P(t)\right)X(t)\end{aligned}$$

For LHS=RHS of (2.5) to hold,

$$-\dot{P}(t) - P(t)A(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t) = Q(t) + A^\top(t)P(t)$$

and by rearranging the terms, we have

$$\dot{P}(t) = -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)$$

showing that the given $P(t)$ satisfies the RDE, with the same final value of P since

$$P(T) = -\frac{1}{2}Y(T)X^{-1}(T) = -\frac{1}{2}(-2M)I = M$$

where $\begin{pmatrix}X(T)\\ Y(T)\end{pmatrix} = \begin{pmatrix}I\\ -2M\end{pmatrix}$ is given and so we have $X^{-1}(T) = I$.

Question 3

(a) The HJB equation is

$$
\begin{aligned}
&0 = \partial_t V(t,x) + \inf_u \left\{ x^\top Q(t) x + u^\top R(t) u + \Big[\partial_x V(t,x)\Big]^\top \Big(A(t)x + B(t)u\Big) \right\}, \\
&u \in L^\infty\Big([0,T], \mathbb{R}^m\Big), \qquad (t,x) \in (0,T) \times \mathbb{R}^d, \\
&V(T,x) = x^\top M x
\end{aligned}
$$

(b) By differentiating the HJB equation with respect to u,

$$
\begin{aligned}
2R(t)u + B^\top(t)\Big[\partial_x V(t,x)\Big] &= 0 \\
u = -\frac{1}{2} R^{-1}(t) B^\top(t) \Big[\partial_x V(t,x)\Big] \qquad &(3.1)
\end{aligned}
$$

This is a minimization because the second partial derivative of the HJB equation with respect to u has the form $2R(t)$, where $R(t)$ is defined to be positive definite. Then, we substitute (3.1) back into the HJB equation.

From the HJB equation, the term $u^\top R(t) u$ evaluates to

$$
\begin{aligned}
u^\top R(t) u &= \frac{1}{4}\Big(R^{-1}(t)B^\top(t)\Big[\partial_x V(t,x)\Big]\Big)^\top R(t) \Big(R^{-1}(t)B^\top(t)\Big[\partial_x V(t,x)\Big]\Big) \\
&= \frac{1}{4}\Big[\partial_x V(t,x)\Big]^\top B(t) R^{-1}(t) B^\top(t) \Big[\partial_x V(t,x)\Big]
\end{aligned}
$$

The other term $\Big[\partial_x V(t,x)\Big]^\top B(t) u$ evaluates to

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

Hence, the HJB equation evaluates to

$$
\begin{aligned}
0 =& \partial_t V(t,x) + x^\top Q(t) x + \Big[\partial_x V(t,x)\Big]^\top A(t) x \\
&+ (\frac{1}{4} - \frac{1}{2})\Big[\partial_x V(t,x)\Big]^\top B(t) R^{-1}(t) B^\top(t) \Big[\partial_x V(t,x)\Big] \\
=& \partial_t V(t,x) + x^\top Q(t) x + \Big[\partial_x V(t,x)\Big]^\top A(t) x \\
&- \frac{1}{4}\Big[\partial_x V(t,x)\Big]^\top B(t) R^{-1}(t) B^\top(t) \Big[\partial_x V(t,x)\Big]
\end{aligned}
$$

and by rearranging the terms, we have shown that the HJB equation simplifies to

$$\begin{aligned} -\partial_t V(t,x) =& x^\top Q(t)x + \Big[\partial_x V(t,x)\Big]^\top A(t)x \\ & - \frac{1}{4}\Big[\partial_x V(t,x)\Big]^\top B(t)R^{-1}(t)B^\top(t)\Big[\partial_x V(t,x)\Big] \\ V(T,x) =& x^\top Mx \end{aligned}$$

(c) Assume that

$$V(t,x) = x^\top P(t)x \tag{3.2}$$

If $P(t)$ is chosen to satisfy the RDE (2.2), then $P(t)$ must be symmetric. This is because if we take the tranpose of the RDE, then both $P(t)$ and $P^\top(t)$ are solutions to the same problem and also have the same final value from (2.3) where

$$P^\top(T) = M = P(T)$$

since M is symmetric.

Then, by differentiating (3.2) with respect to x and t separately, we have

$$\begin{aligned} \partial_x V(t,x) &= 2P(t)x, \\ \partial_t V(t,x) &= x^\top \dot{P}(t)x \end{aligned}$$

We substitute the above partial derivatives into the simplified HJB equation:

$$\begin{aligned} -x^\top \dot{P}(t)x &= x^\top Q(t)x + 2x^\top P(t)A(t)x - x^\top P(t)B(t)R^{-1}(t)B^\top(t)P(t)x \\ x^\top \dot{P}(t)x &= x^\top \Big(-2P(t)A(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)\Big)\, x \\ x^\top \dot{P}(t)x &= x^\top \Big(-P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)\Big)\, x \end{aligned} \tag{3.3}$$

where $x^\top P(t)A(t)x = x^\top A^\top(t)P(t)x$ because they are scalars. In (3.3), we have a quadratic form.

Claim: To show that for any symmetric matrices C and D such that $x^\top Cx = x^\top Dx$ for all x, we must have $C = D$.

Proof: For $x = e_i + e_j$, which is the sum of 2 different standard basis vectors,

$$c_{ii} + 2c_{ij} + c_{jj} = x^\top Cx = x^\top Dx = d_{ii} + 2d_{ij} + d_{jj}$$

Since $c_{kk} = e_k^\top Ce_k = e_k^\top De_k = d_{kk}$, the above evaluates to

$$c_{ij} = d_{ij} \Leftrightarrow C = D$$ ■

By the above claim, for (3.3) to hold,

$$\dot{P}(t) = -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)$$

which is the RDE (2.2). This is because the LHS and RHS of the RDE are symmetric from the fact that $P(t), Q(t)$ and $R(t)$ are symmetric. Also, from (3.2) and the HJB equation,

$$x^\top P(T)x = V(T, x) = x^\top Mx$$

and similarly, since $P(t)$ and M are symmetric, for the above to hold,

$$P(T) = M$$

which is the final value of P. Since the PMP is satisfied if $P(t)$ satisfies the RDE, the defined $V(t, x)$ solves the HJB equation and the control u^* is optimal.

Question 4

(a) Define the state variable $w_t \coloneqq \begin{pmatrix} x_t \\ v_t \end{pmatrix}$.

Comparing this problem to the general form of the LQR problem in Q1, we can identify the following quantities:

$$A(t) = \begin{pmatrix} 0 & 1 \\ 0 & -\alpha(t) \end{pmatrix}, \qquad B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad Q = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} \lambda \end{pmatrix},$$
$$P(1) = M = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad w_0 = \begin{pmatrix} x_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad t \in [0, 1]$$

where Q and M are symmetric positive semi-definite, and $\lambda > 0$ because R is symmetric positive definite. We will solve the problem at $\alpha(t) = \sin(10t)$ and $\alpha(t) = t^2$.

If $\lambda \gg 0$, that is the penalty cost for supplying acceleration is very large, the optimal control u_t would be driven towards zero. This causes the vehicle to travel a very short distance, far from our goal of reaching the origin. Hence, we adjust λ accordingly and finally fix it at $\lambda = 0.01$.

All matrix differential equations in both methods described below are solved using the `odeintw` function. Since Q is defined to be a zero matrix in this problem, we will simplify the equations used in the algorithms by removing the terms with Q.

i. **RICCATI DIFFERENTIAL EQUATION (RDE)**

With the quantities defined above, we can solve the problem using the RDE as follows:

Algorithm 1: Solve the problem using the RDE

Input: Matrices $A(t), B, R, M, w_0$ and $t \in [0, 1]$

$\boldsymbol{P} \leftarrow$ Solve $\dot{P}(t) = -P(t)A(t) - A^\top P(t) + P(t)BR^{-1}B^\top P(t), \quad P(1) = M,$
by integrating backward in time for $t \in [1, 0]$

$\boldsymbol{w} \leftarrow$ Solve $\dot{w}_t = A(t)w_t + Bu_t, \quad w(0) = w_0, \quad$ where $u_t = -R^{-1}B^\top P(t)w_t,$
by integrating forward in time for $t \in [0, 1]$

$\boldsymbol{u} \leftarrow$ Compute $u_t = -R^{-1}B^\top P(t)w_t$

return $\boldsymbol{P}, \boldsymbol{w}, \boldsymbol{u}$

---

ii. METHOD OF SUCCESSIVE APPROXIMATIONS (MSA)

For the second method, we will use the MSA, and also replace the maximization step by steepest ascent. The MSA algorithm is as follows:

Algorithm 2: Solve the problem using the MSA

Input: Matrices $A(t), B, R, M, w_0$ and $t \in [0,1]$

Initialize: $\boldsymbol{u} \in L^\infty\Big([0,1], \mathbb{R}\Big)$

$\texttt{val} \leftarrow \infty$

while 1 **do**

  $\boldsymbol{w} \leftarrow$ Solve $\dot{w}_t = A(t)w_t + Bu_t, \quad w(0) = w_0,$
  by integrating forward in time for $t \in [0,1]$

  $\boldsymbol{p} \leftarrow$ Solve $\dot{p}_t = -A^\top(t)p_t, \quad p_1 = -2Mx_1,$
  by integrating backward in time for $t \in [1,0]$

  for $t \in [0,1]$*;* // Update each entry of $\boldsymbol{u}$ by steepest ascent
  do
    $u_t \leftarrow u_t + \eta\Big(B^\top p_t - 2Ru_t\Big), \quad$ for small $\eta = 0.7$

  $\texttt{val}_{\texttt{new}} \leftarrow x_1^2 + \lambda\|\boldsymbol{u}\|_2^2$

  if $\texttt{val}_{\texttt{new}} < \texttt{val}$; // Stopping criterion
  then
    $\texttt{val} \leftarrow \texttt{val}_{\texttt{new}}$
  else
    break

return $\boldsymbol{w}, \boldsymbol{p}, \boldsymbol{u}$

We adjust the learning rate accordingly and fix it at $\eta = 0.7$ such that the algorithm still converges but at a relatively fast rate.

To initialize $\boldsymbol{u}$ for starting the algorithm, since our goal is for the vehicle to travel from $x_0 = 1$ to $x_1 \approx 0$, we need to supply acceleration in the negative direction. We make a guess and fix $\boldsymbol{u}$ to be a constant vector of the constant value -5.

For the stopping criterion, we use the discretized objective value

$$\texttt{val} = x_1^2 + \lambda \sum_t u_t^2 = x_1^2 + \lambda\|\boldsymbol{u}\|_2^2$$

Since this is a minimization problem, once the objective value increases in the current iteration, we end the algorithm. We will confirm that the algorithm converges around a single value in Figure (1) below.

We run the algorithm separately for 100 iterations to examine the changes in the objective value, `val`, to get the following:

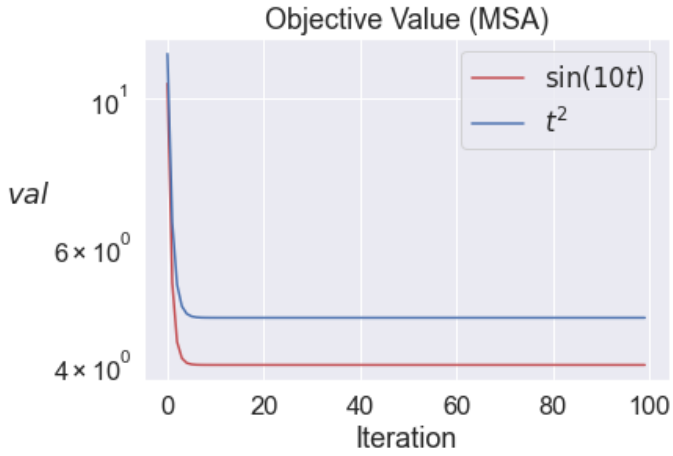


FIGURE 1. Convergence of objective value under the MSA method

From Figure (1), the graph shows the convergence of the MSA method at a single objective value for both $\alpha(t)$. We see that the algorithm converges fairly quickly from iteration 10 onwards.

Moreover, we can check that in the actual algorithm with the stopping criterion described above, for $\alpha(t) = \sin(10t)$, the solution is reached at iteration 27, whereas for $\alpha(t) = t^2$, the solution is reached at iteration 36.

(b) GRAPHING RESULTS

For the RDE method, we have the following results:

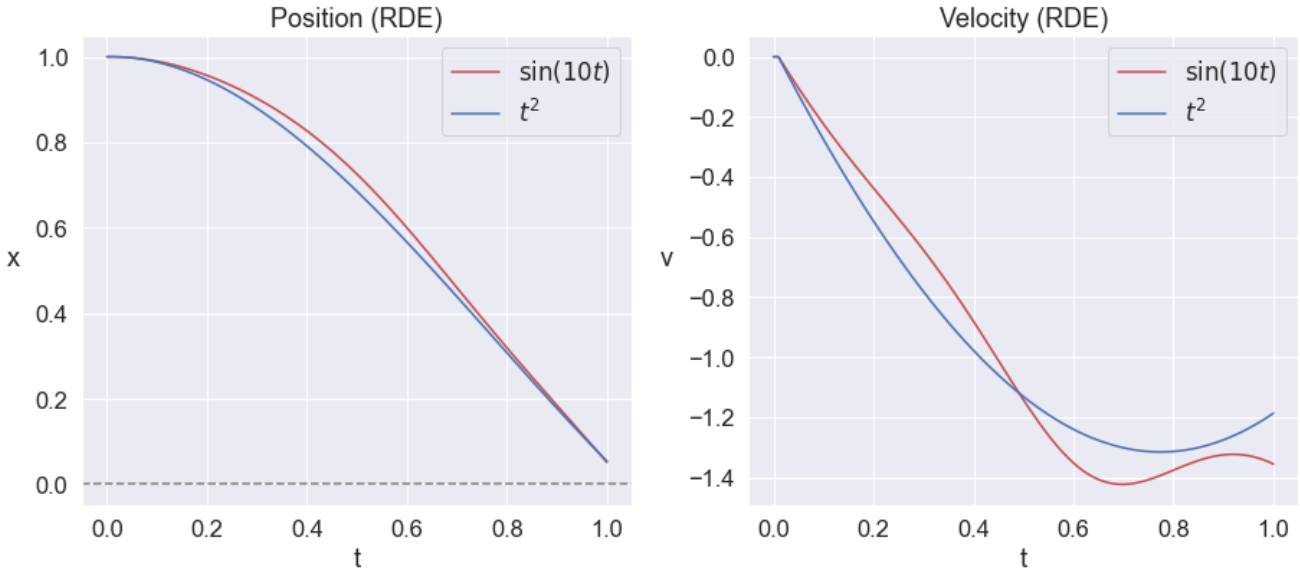


FIGURE 2. Position and velocity graphs of the RDE method

For the MSA method, we have the following results:

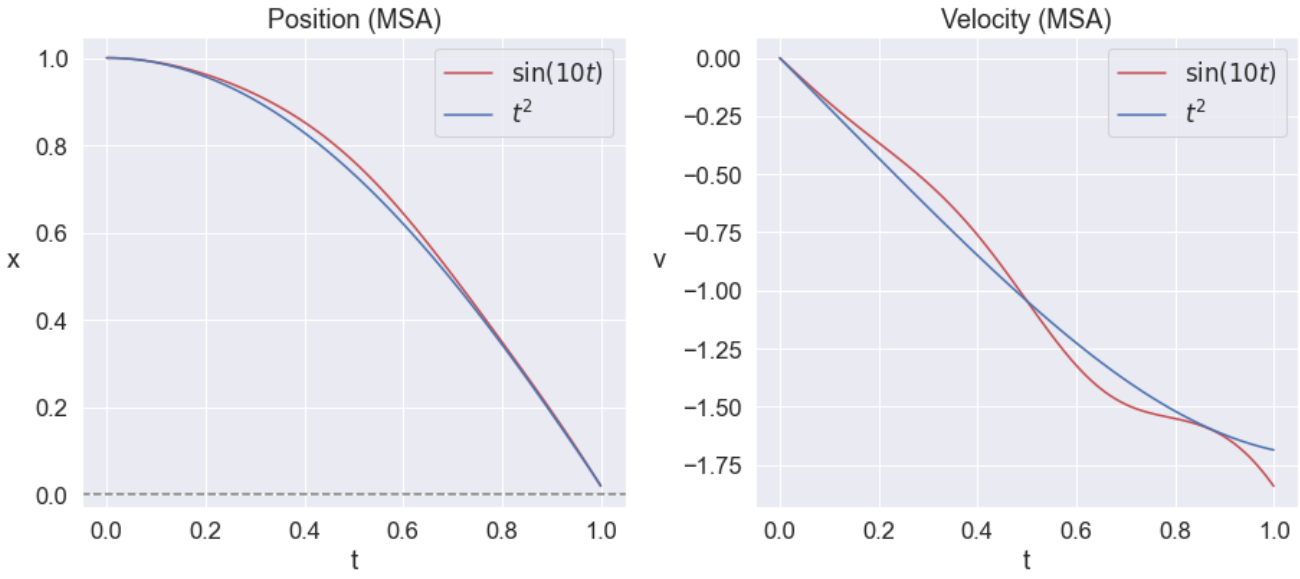


FIGURE 3. Position and velocity graphs of the MSA method

A horizontal grey dotted line is plotted at $x = 0$ for the position graphs as a reference. Some observations from Figures (2) and (3) are:

(i) From the position graphs of both methods, both decrease steadily towards 0 at around the same time, although the graph for t^2 is just slightly faster at the start. This can be explained by the velocity graph.

(ii) For the velocity graph, we see that the velocity increases in the negative direction, but at different rates for different $\alpha(t)$. This is makes sense because over time, the resistance $\alpha(t) = \sin(10t)$ fluctuates about the interval [0,1] while $\alpha(t) = t^2$ always increases.

However, for the RDE method, at around $t = 0.8$, the magnitudes of the velocities stop increasing at a value of about 1.3. As for the MSA method, the magnitudes continue increasing to a value of about 1.75. This may indicate a difference in the solution accuracy.

SOLUTION ACCURACY

Method, $\alpha(t)$	Estimated Solution
RDE, $\sin(10t)$	0.05137
RDE, t^2	0.05411
MSA, $\sin(10t)$	0.01986
MSA, t^2	0.02154

TABLE 1. Estimated solution for each method and $\alpha(t)$

From Table (1), we observe that the MSA method has a much better accuracy because the estimated solution is much closer to the origin 0 than that of the RDE method.

This corresponds to our findings from the graphs whereby under the RDE method, the magnitudes of the velocities stop increasing at around $t = 0.8$. As such, the lower speed causes the vehicle to be further away from the origin.

Also, under each method, the solution for $\alpha(t) = \sin(10t)$ is closer to 0 compared to the corresponding $\alpha(t) = t^2$. This is most likely due to the fact that at points approaching $t = 1$, the resistance $t^2 > \sin(10t)$.

RUNNING TIME

Method, $\alpha(t)$	Mean Running Time (milliseconds)
RDE, $\sin(10t)$	179
RDE, t^2	169
MSA, $\sin(10t)$	83.3
MSA, t^2	56.2

TABLE 2. Running time for each method and $\alpha(t)$

We use the magic command `%timeit` to calculate the mean running time. From Table (2), we see that the MSA method runs much faster than the RDE method. This shows that solving the non-linear RDE can be more computationally expensive than solving just iterations of linear differential equations under the MSA method. On a side note, we can also check that computing $\sin(10t)$ is slower than computing t^2. This results in the noticeable difference between each $\alpha(t)$ under the same method.

Under the MSA method, we need to take into account the magnitude of the learning rate η. Tuning the learning rate is important because it needs to be small enough for the algorithm to converge but also large enough such that the running time is relatively fast. We can check that lowering the learning rate results a longer running time and at a small enough value, the MSA method can be slower than the RDE method. Thus, optimizing the learning rate is important.

Besides the learning rate, initializing $\boldsymbol{u}$ is also important. We can check that if the values of $\boldsymbol{u}$ are too large in the negative direction, more iterations will be required, resulting in greater computation time. On the other hand, if the values are towards the positive direction, the solution may have larger errors or even not converge at all. Because we are given the scale of the problem where $x_0 = 1$ and $x_1 \approx 0$, we can make the initial guess that the values of $\boldsymbol{u}$ should not be far off.

MEMORY COST

Method	Peak Memory Usage (MiB)
RDE	126.57
MSA	125.05

TABLE 3. Peak memory usage for each method

We use the magic command `%memit` to compare the peak memory usage reached while running each algorithm. From Table (3), the peak memory usage for both methods

are rather similar. This may be because the same set of inputs are used for both methods. Moreover, the number of iterations under the MSA method does not affect the memory cost because we are always reassigning the same variables stored.

Nevertheless, the peak memory usage under the RDE method is slightly higher. The output variables that are different between the two methods are $\boldsymbol{P}$ and $\boldsymbol{p}$. As such, the difference in memory cost could probably be explained by the difference in the sizes of $\boldsymbol{P} \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{p} \in \mathbb{R}^2$ stored. If the size of the problem grows, the difference in memory cost should become more significant.

CONCLUSION

The RDE method seems to be less efficient due its non-linearity. Meanwhile, the MSA method requires us to properly optimize the learning rate and make an appropriate initial guess of $\boldsymbol{u}$ for the algorithm to be more efficient. To conclude Q4, the MSA method performs better than the RDE method in terms of solution accuracy, running time and memory cost.

Chong Zhen Jie A0201613Y

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from odeintw import odeintw
```

In [2]:
```
sns.set_theme(font_scale=1.5)
%load_ext memory_profiler
```

Project Part IV: Question 4 using the RDE

In [3]:
```
# Set parameters
x0 = 1
v0 = 0
w0 = np.array([[x0], [v0]])

t0, t1 = 0, 1
t_list = np.linspace(t0, t1, 101)
t_list_rev = np.flip(t_list)

param = 1e-2 # value of lambda
R = np.array([[param]])
B = np.array([[0],[1]])
M = np.array([[1,0],[0,0]])
```

```
def compute_A(fn):
    return lambda t: np.array([[0.,1], [0,-fn(t)]])
fn1 = lambda t: np.sin(10*t)
fn2 = lambda t: t**2
A1 = compute_A(fn1)
A2 = compute_A(fn2)

# Define functions to compute the derivatives
def dPdt(P, t, A):
    return -P@A(t) - A(t).T@P + P@B@np.linalg.inv(R)@B.T@P

def dwdt(w, t, A, P):
    # Retrieve corresponding P(t) by indexing from the back
    return A(t)@w + (B@(-np.linalg.inv(R)@B.T@P[-int(t*100)]@w))
```

In [4]:
```
# Define function to solve problem using RDE
def RDE(A):
    # Solve RDE for P, integrating backward in time for t in [1,0]
    P = odeintw(dPdt, M, t_list_rev, args=(A,))

    # Solve for x and v, integrating forward in time for t in [0,1]
    w = odeintw(dwdt, w0, t_list, args=(A, P))
    x, v = w[:, 0], w[:, 1]

    # Compute u; P is flipped to get back the same order of increasing time
    u = [(-np.linalg.inv(R)@B.T@P_t@w_t)[0] for P_t,w_t in zip(np.flip(P),w)]

    return P, x, v, u
```

In [5]:
```
# Run solver
P1, x1, v1, u1 = RDE(A1)
P2, x2, v2, u2 = RDE(A2)
```

In [6]:
```
# Check peak memory usage: around 126.5 MiB
# %memit RDE(A1)
# %memit RDE(A2)
```

In [7]:
```
# Check mean running time
# %timeit -r5 -n10 RDE(A1) # around 180 ms
# %timeit -r5 -n10 RDE(A2) # around 170 ms
```

In [8]:
```
print(f"Estimated final position for sin(10t): {x1[-1][0]}")
print(f"Estimated final position for t^2: {x2[-1][0]}")
```

```
Estimated final position for sin(10t): 0.05137190637309329
Estimated final position for t^2: 0.05411054944003505
```

In [9]:
```
# Plot results for x and v over t_list
plt.figure(figsize=(13,6), tight_layout=True)

def plot_custom(input1, input2, ylabel, title):
    plt.plot(t_list, input1, label="$\sin(10t)$", c="r")
    plt.plot(t_list, input2, label="$t^2$", c="b")
    plt.xlabel("t")
    plt.ylabel(ylabel, rotation="horizontal", labelpad=15)
    plt.title(title)
    plt.legend()

plt.subplot(1,2,1)
plot_custom(x1, x2, "x", "Position (RDE)")
# Reference line
plt.axhline(0, ls="--", c="grey")

plt.subplot(1,2,2)
plot_custom(v1, v2, "v", "Velocity (RDE)")

plt.show()
```

Chong Zhen Jie A0201613Y

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from odeintw import odeintw
```

In [2]:
```
sns.set_theme(font_scale=1.5)
%load_ext memory_profiler
```

Project Part IV: Question 4 using the MSA

In [3]:
```
# Set parameters
x0 = 1
v0 = 0
w0 = np.array([[x0], [v0]])

t0, t1 = 0, 1
t_list = np.linspace(t0, t1, 101)
t_list_rev = np.flip(t_list)

param = 1e-2 # value of lambda
R = np.array([[param]])
B = np.array([[0],[1]])
M = np.array([[1,0],[0,0]])

def compute_A(fn):
    return lambda t: np.array([[0.,1], [0,-fn(t)]])
fn1 = lambda t: np.sin(10*t)
fn2 = lambda t: t**2
A1 = compute_A(fn1)
A2 = compute_A(fn2)

# Define functions to compute the derivatives
def dwdt(w, t, A, u):
    # Retrieve corresponding u_t by indexing from the front
    return A(t)@w + B@u[int(t)*100]

def dpdt(p, t, A):
    return -A(t).T@p

def H_u(p, u):
    return B.T@p - 2*R*u
```

In [4]:
```
# Define function to run MSA algorithm
def MSA(A):
    # Initialize u; each entry u_t is in matrix form of size (1,1) for convenience
    u = np.array([[[-5]] for _ in range( len(t_list) )])

    # Fix learning rate
    eta = 7e-1

    # Initialize objective value
    val = np.inf

    # Record number of iterations
    count = 0

    # Main loop
    while 1:
        count += 1

        # Solve for w, integrating forward in time for t in [0,1]
        w = odeintw(dwdt, w0, t_list, args=(A, u))

        # Solve for p, integrating backward in time for t in [1,0]
        p = odeintw(dpdt, -2*M@w[-1], t_list_rev, args=(A,))

        # Update each entry by steepest ascent
        u = u + eta*(np.array([H_u(p_t,u_t) for p_t,u_t in zip(np.flip(p),u)]))

        # Stopping criterion
        val_new = w[-1][0]**2 + param*(np.linalg.norm(u)**2)
        if val_new < val:
            val = val_new
        else:
            break

    x, v = w[:, 0], w[:, 1]

    print(f"Solution reached at iteration {count}.")
    return x, v, p, u
```

In [5]:
```
# Run solver
x1, v1, p1, u1 = MSA(A1)
x2, v2, p2, u2 = MSA(A2)
```

```
Solution reached at iteration 27.
Solution reached at iteration 36.
```

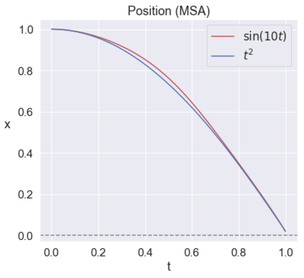
In [6]:
```
# Check peak memory usage; around 125 MiB
# %memit MSA(A1)
# %memit MSA(A2)
```

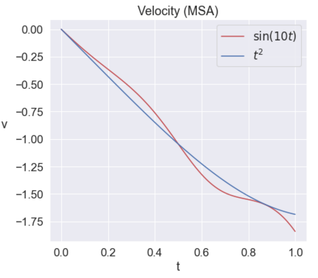
In [7]:
```
# Check mean running time
# %timeit -r5 -n10 MSA(A1) # slightly under 100 ms
# %timeit -r5 -n10 MSA(A2) # around 50 ms
```

In [8]:
```
print(f"Estimated final position for sin(10t): {x1[-1][0]}")
print(f"Estimated final position for t^2: {x2[-1][0]}")
```

```
Estimated final position for sin(10t): 0.019862804036121194
Estimated final position for t^2: 0.021544376741219257
```

In [9]:
```
# Plot results for x and v over t_list
plt.figure(figsize=(13,6), tight_layout=True)

def plot_custom(input1, input2, ylabel, title):
    plt.plot(t_list, input1, label="$\sin(10t)$", c="r")
    plt.plot(t_list, input2, label="$t^2$", c="b")
    plt.xlabel("t")
    plt.ylabel(ylabel, rotation="horizontal", labelpad=15)
    plt.title(title)
    plt.legend()

plt.subplot(1,2,1)
plot_custom(x1, x2, "x", "Position (MSA)")
# Reference line
plt.axhline(0, ls="--", c="grey")

plt.subplot(1,2,2)
plot_custom(v1, v2, "v", "Velocity (MSA)")

plt.show()
```





Convergence of the MSA method

In [10]:
```
# Modify function to run for 100 iterations
def MSA(A):
    u = np.array([[[-5]] for _ in range( len(t_list) )])
    eta = 7e-1
    val_list = []

    # Fixed at 100 iterations
    for _ in range(100):
        w = odeintw(dwdt, w0, t_list, args=(A, u))
        p = odeintw(dpdt, -2*M@w[-1], t_list_rev, args=(A,))
        u = u + eta*(np.array([H_u(p_t,u_t) for p_t,u_t in zip(np.flip(p),u)]))

        # Record changes in objective value over time
        val_new = w[-1][0]**2 + param*(np.linalg.norm(u)**2)
        val_list.append(val_new)

    return val_list
```

In [11]:
```
# Run solver for 100 iterations
val1 = MSA(A1)
val2 = MSA(A2)
```

In [12]:
```
# Plot changes in objective value
plt.plot(range(100), val1, label="$\sin(10t)$", c="r")
plt.plot(range(100), val2, label="$t^2$", c="b")
plt.xlabel("Iteration")
plt.ylabel("$val$", rotation="horizontal", labelpad=15)
plt.yscale("log")
plt.title("Objective Value (MSA)")
plt.legend()

plt.show()
```