Homework Problem: Matrix Exponential and System Dynamics

Introduction

In this problem, you will delve into the analytical and symbolic computation of the matrix exponential e^A, where A is a 2×2 matrix defined as:

$$A = \begin{bmatrix} 1 & 0 \\ \frac{k}{m} & \frac{b}{m} \end{bmatrix}$$

Objectives

- **Analytical and Symbolic Computation of e^A:** You will learn how to compute the matrix exponential e^A both analytically and symbolically. This involves leveraging techniques from linear algebra to handle the exponential of a matrix, which is essential in solving systems of differential equations.
- Additionally, you will learn about the Euler approximation, a numerical method for solving ordinary differential equations, and understand how it applies to approximating the system's response.

Overview

State-Space Representation

The state-space equation governing the system is:

$$\dot{x} = Ax + Bu$$

To solve this differential equation, we use the matrix exponential method. The general solution is given by:

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}Bu(s)\,ds$$

Note that since Case Study D in the controlbook is already linearized, there will be no nonlinear terms that can send the system predictor awry.

Simplifying with Zero Initial State

Given that the initial state x_0 is a zero vector, the solution simplifies to:

$$x(t) = \int_0^t e^{A(t-s)} Bu(s)\, ds$$

Matrix Exponential via Euler Approximation

While the analytical computation of $\int_0^t e^{A(t-s)} Bu(s)\, ds$, it can be difficult to discretize. Therefore, numerical methods like the **Euler approximation** are invaluable for approximating the matrix exponential and, consequently, the system's response.

Euler Approximation: This is a first-order numerical method for solving ordinary differential equations (ODEs) with a given initial value. It approximates the solution by taking small, discrete steps along the direction of the derivative. In the context of our state-space equation, the Euler method can be used to iteratively compute the state $x(t)$ over small time increments.

The Euler approximation for the state update can be expressed as:

$$x(t + \Delta t) \approx x(t) + \Delta t \cdot (\dot{x})$$

which becomes

$$x(t + \Delta t) \approx x(t) + \Delta t \cdot (e^{A(dt)} \cdot Bu(t) \cdot dt\,)$$

Where:

- Δt is a small time step.
- $x(t)$ is the current state.
- $(e^{A(dt)} \cdot Bu(t) \cdot dt)$ is the derivative of the integral of the state at time t (and is therefore simply the state at the next time interval).

This iterative process allows you to approximate the state $x(t)$ over time without explicitly computing the integral.

Fitting It All Together

Most of this assignment is set up for you. All of the code you need to develop is found in `todo.py`. In this assignment, you will:

1. **Compute e^{A*dt} Analytically**
2. **Compute e^{A*dt} Symbolically**
3. **Compare with Visualization:** Use the provided data plotter, which includes a dotted red line representing the predicted state based on your matrix exponential computation, to verify the accuracy of your analytical and numerical solutions.

Repository Overview

This homework problem utilizes controlbook_public repository, which serves as a public resource for the Controls class at BYU, with permission from one of its owners, Dr. Cammy Peterson. We will be working with **Case Study D**, which provides a practical scenario to apply the concepts learned in class.

Visualization

To aid your understanding, a dotted red line has been added to the data plotter. This line represents the **predicted state** of the system according to our computed matrix exponential $\int_0^t e^{A(t-s)} Bu(s)\, ds$. Comparing your analytical and symbolic results with this prediction will help you verify the accuracy of your computations.

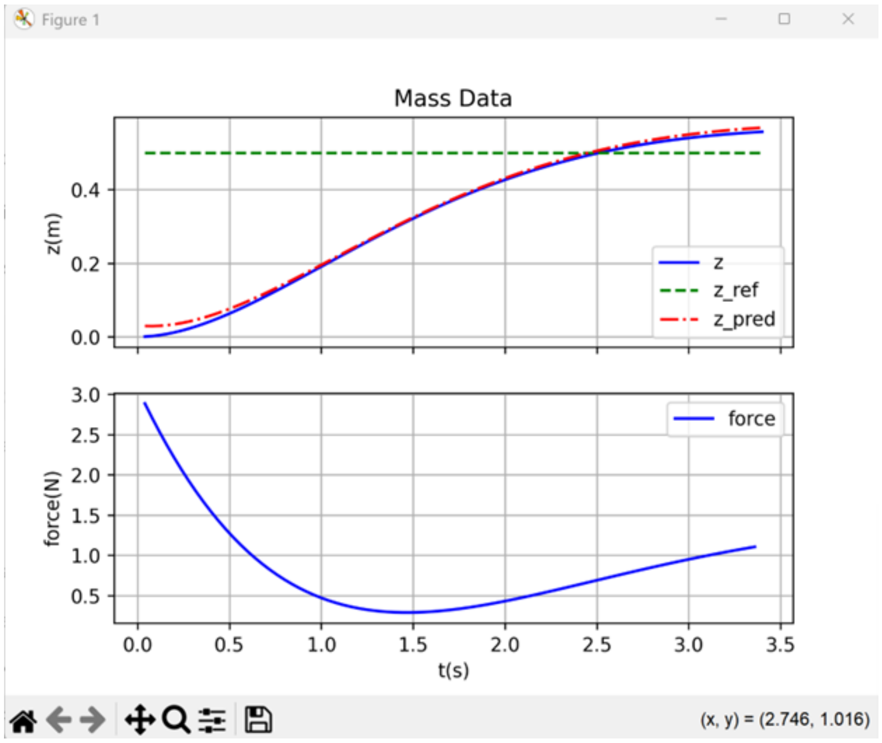

Figure 1: Labelled dataPlotter lines.

Note that these labels will not be seen in your version of the code, since it takes up too much computational time.

Your job, again, is to make sure that the predicted **z** position is accurate no matter which version of exponential matrix computation is used.

Getting Started

1. **Clone the Repository:**

`https://github.com/FredJones4/671_controlbook_D.git`

2. **Review the Provided Materials:** Familiarize yourself with the system parameters and existing codebase.
3. **Set up the enviroment:** The virtual environment is provided for you. See the `Code setup instructions.pdf`, as used in the Controls class, for more information about virtual environments and setup in Visual Studio Code.
4. **Test the animation code:** The code is setup so you can run the animation without needing to write any code.
   (a) Run the `run.py` file.
   (b) Go to `todo.py`.
   (c) Change the variable **"use self defined matrix"** to 0 or 1. This will change how $e^{A(dt)}$ is computed.

Conclusion

Happy computing!