ECE 450 - Exam #1

Collin Heist

September 18, 2019

1 Solution

I'll be using the following state variable assignments:

$$\begin{aligned} x_1 &= p(t) \\ x_2 &= q(t) \\ x_3 &= \frac{dq(t)}{dt} \\ x_4 &= r(t) \end{aligned}$$

Using these state variables, the provided set of differential equations can be rewritten as the following:

$$\dot{x}_1 + 20x_1 = f(t) \tag{1}$$
$$\dot{x}_3 + 10x_3 + 20\dot{x}_1 + 400x_2 = 0 \tag{2}$$
$$\dot{x}_4 + 5x_4 + 20x_2 = g(t) \tag{3}$$

With these state-variable-ized equations, a solution for $\dot{x}_1, \dot{x}_3,$ and $\dot{x}_4$ can be found by solving for them inside **Equations 1**, **2**, and **3**.

$$\dot{x}_1 = -20x_1 + f(t) \tag{4}$$
$$\dot{x}_3 = -20\dot{x}_1 - 400x_2 - 10x_3 \tag{5}$$
$$\dot{x}_4 = -20x_2 - 5x_4 + g(t) \tag{6}$$

The final form of these state space equations can then be found by substituting **Equation 4** into **Equation 5**, resulting in the following:

$$\begin{aligned} \dot{x}_1 &= -20x_1 + f(t) \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -20 \cdot (-20x_1 + f(t)) - 400x_2 - 10x_3 \\ \dot{x}_4 &= -20x_2 - 5x_4 + g(t) \end{aligned}$$

This can be simplified, and the aligned equations reveal the **A** and **B** matrices.

$$
\begin{aligned}
\dot{x}_1 &= -20x_1 && && +f(t) && (7)\\
\dot{x}_2 &= && x_3 && && (8)\\
\dot{x}_3 &= 400x_1 - 400x_2 && -10x_3 && -20f(t) && (9)\\
\dot{x}_4 &= -20x_2 && -5x_4 + && g(t) && (10)
\end{aligned}
$$

Thus, **Equations 7-10** can be used to find the state space formulation. The output is given as $r(t)$, which was chosen to be the state variable x_4, creating a very simple output equation.

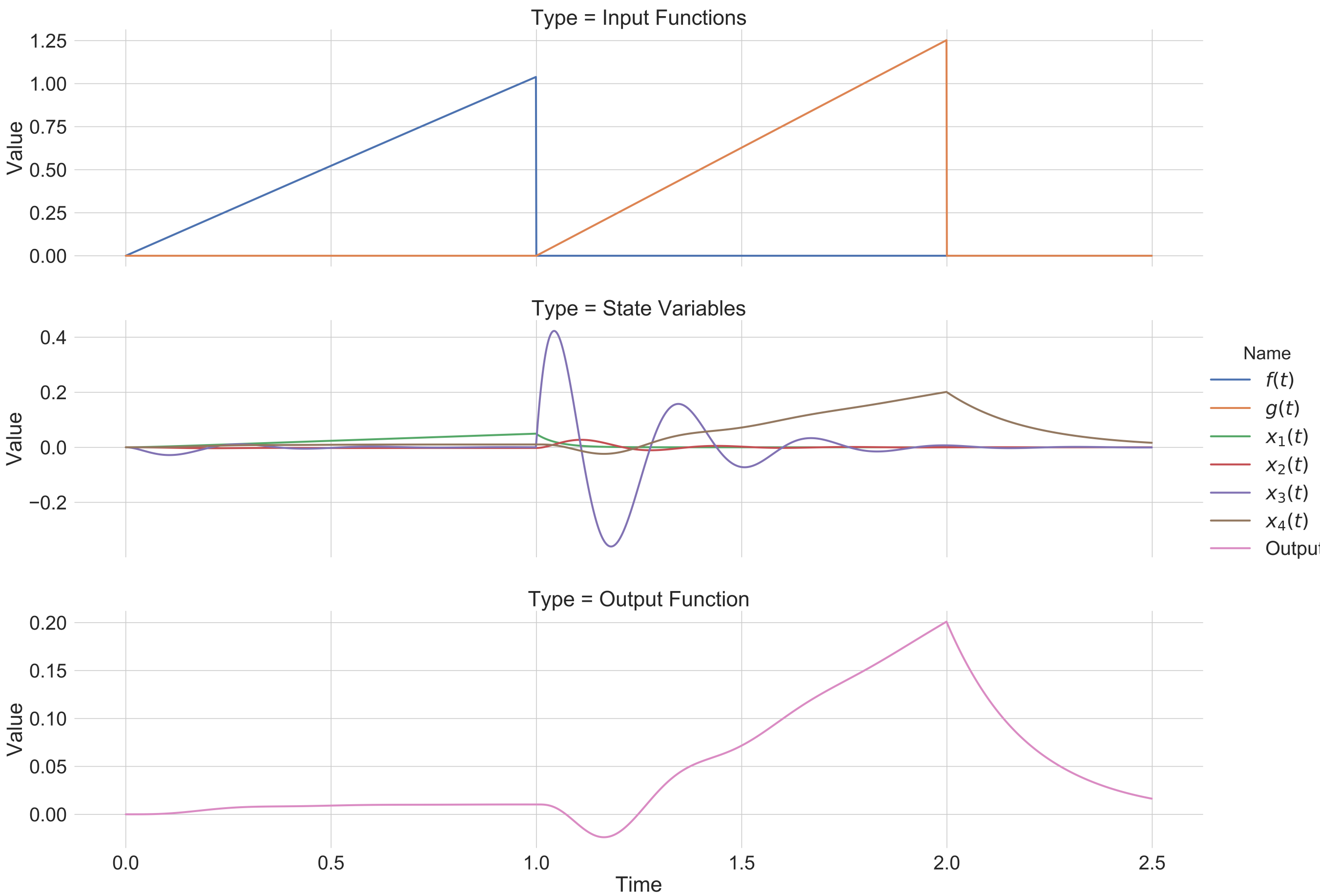
$$
\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix} = \begin{bmatrix}-20 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 400 & -400 & -10 & 0\\ 0 & -20 & 0 & -5\end{bmatrix} \cdot \begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 0\\ -20 & 0\\ 0 & 1\end{bmatrix} \cdot \begin{bmatrix}f(t)\\ g(t)\end{bmatrix} \tag{11}
$$

$$
y_{out}(t) = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + 0 \tag{12}
$$

From **Equation 11** and **12**, matrix **A**, **B**, **C**, and **D** can be stated as:

$$
A = \begin{bmatrix}-20 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 400 & -400 & -10 & 0\\ 0 & -20 & 0 & -5\end{bmatrix}, B = \begin{bmatrix}1 & 0\\ 0 & 0\\ -20 & 0\\ 0 & 1\end{bmatrix}, C = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}, D = 0
$$

Below is a plot of the given input functions, state variables (defined at the beginning of **Section 1**), and the output function – all as a function of time.



My code for generating the above plot is listed below, in **Section 2**.

2 Code Appendix

2.1 Package Imports

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

2.2 Generic Function for a State Variable Solver

```
In [2]: def state_solver(A_matrix, B_matrix, force_f, initial_conditions,
                         time_range=[0, 10], dt=0.01):
            time_values = np.arange(time_range[0], time_range[1], dt)
            x_vals = np.array(initial_conditions)
            state_variables = [[] for _ in initial_conditions]

            # Loop through each instance in time, calculate the state variable at that time
            for time in time_values:
                # For multiple input functions, do matrix operations on the 2nd term
                if isinstance(force_f, list):
                    force_vals = np.array([func(time) for func in force_f])
                    x_vals = x_vals + dt * (A_matrix @ x_vals) + dt * (B_matrix @ force_vals)
                else:
                    x_vals = x_vals + dt * (A_matrix @ x_vals) + dt * (B_matrix * force_f(time))
                # Add the value of the state variable at this time to the list
                for index, _ in enumerate(state_variables):
                    state_variables[index].append(x_vals[index])

            return state_variables, time_values
```

2.3 Generic Function for Creating a Combined `DataFrame` out of State Variables, Input Function(s), and an Output Function

```
In [3]: def create_df(time, state_variables, input_fs, output_f, state_names, input_names):
            df_list = []
            # If there is only one input, create an array out of it anyway (for looping)
            input_fs = input_fs if isinstance(input_fs, (list, np.ndarray)) else [input_fs]
            # For each input function, compute the time-values and add to the DF list
            for input_f, name in zip(input_fs, input_names):
                input_vals = [input_f(t) for t in time]
                df = pd.DataFrame(list(zip(time, input_vals)), columns=["Time", "Value"])
                df["Name"] = name             # Name the solution column
                df["Type"] = "Input Functions" if len(input_fs) > 1 else "Input Function"
                df_list.append(df)

            # Loop through every state variable, add its time and output to a list of DFs
            for state_var, name in zip(state_variables, state_names):
```

```
            df = pd.DataFrame(list(zip(time, state_var)), columns=["Time", "Value"])
            df["Name"] = name              # Name the solution column
            df["Type"] = "State Variables" # The 'type' of this DF
            df_list.append(df)

        # Add the output to the DF list
        output_vals = output_f(state_variables)
        df = pd.DataFrame(list(zip(time, output_vals)), columns=["Time", "Value"])
        df["Name"] = "Output"          # Name of the instance
        df["Type"] = "Output Function" # The 'type' of this DF
        df_list.append(df)

        return pd.concat(df_list, ignore_index=True, axis=0)
```

2.4 Generic Function to create the Three Necessary Plots

```
In [4]: def create_plots(df):
            sns.set(style="whitegrid", font_scale=2.25)
            g = sns.FacetGrid(df, hue="Name", row="Type", height=5.5, aspect=4, sharey=False)
            g.map(sns.lineplot, "Time", "Value", **dict(linewidth=2.5)).add_legend().despine(bot
```

2.5 My Solution

```
In [5]: A_matrix = np.array([[-20,    0,   0,  0],
                             [  0,    0,   1,  0],
                             [400, -400, -10,  0],
                             [  0,  -20,   0, -5]])
        B_matrix = np.array([[  1, 0],
                             [  0, 0],
                             [-20, 0],
                             [  0, 1]])
        init_state = [0, 0, 0, 0]

        step = lambda t: 0 if t < 0 else 1
        f = lambda t: 5 * np.sin((2 * np.pi * t)/30) * (step(t) - step(t - 1))
        g = lambda t: 10 * np.sin((2 * np.pi * (t - 1))/50) * (step(t - 1) - step(t - 2))
        input_funcs = [f, g]
        output_func = lambda st: [x4 for x1, x2, x3, x4 in zip(st[0], st[1], st[2], st[3])]

        state_vars, time = state_solver(A_matrix, B_matrix, input_funcs, init_state, [0, 2.5])
```

Plot the state variables over time

```
In [6]: state_names = ["$x_1(t)$", "$x_2(t)$", "$x_3(t)$", "$x_4(t)$"]
        input_names = ["$f(t)$", "$g(t)$"]
        df = create_df(time, state_vars, input_funcs, output_func, state_names, input_names)
```

Plot the results

```
In [7]: create_plots(df)
```