Contol theory Homework #1 report

Anton Brisilin, BS18-02 Student

February 22, 2020

1 Task 1.

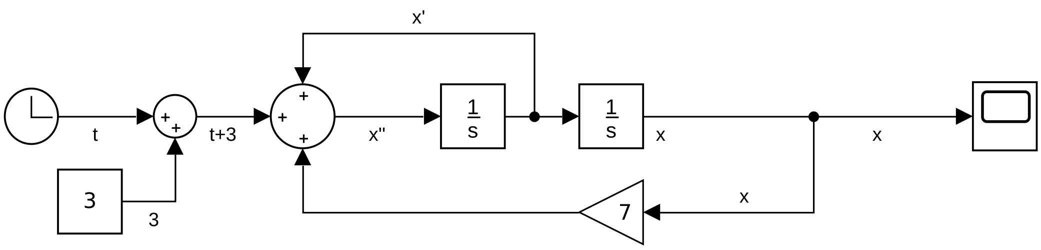
Given equation:

$$x'' - 5x = x' + t + 2x + 3, x'(0) = 4, x(0) = 3$$

Simplified DE:

$$x'' = x' + 7x + t + 3$$

Simulink schema (w/o transfer function block):



Calculation of transfer function:
Given equation:

$$x'' = x' + 7x + t + 3, x'(0) = 4, x(0) = 3$$

Introduce new operator: $p = \frac{d}{dt}$

$$p^2X = pX + 7X + t + 3$$

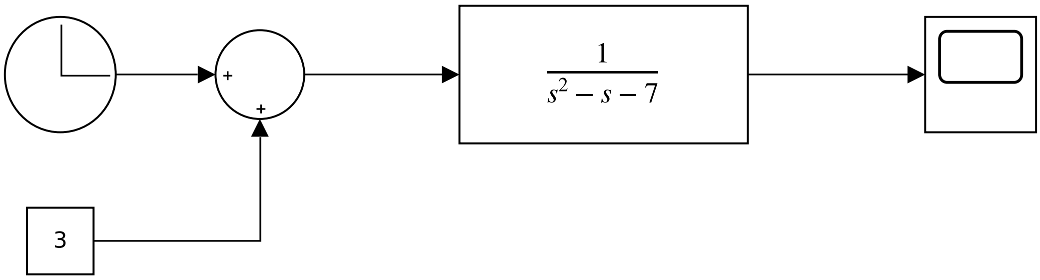
$$X(p^2 - p - 7) = t + 3$$

$$X = \frac{1}{(p^2 - p - 7)}(t + 3)$$

Therefore, our transfer function is

$$T = \frac{1}{(s^2 - s - 7)}$$

Now we can build Simulink schema with transfer function block to solve our DE. Needs to note, that transfer function is designed for very simple cases, when initial conditions are all zero.
Simulink schema (with transfer function block):



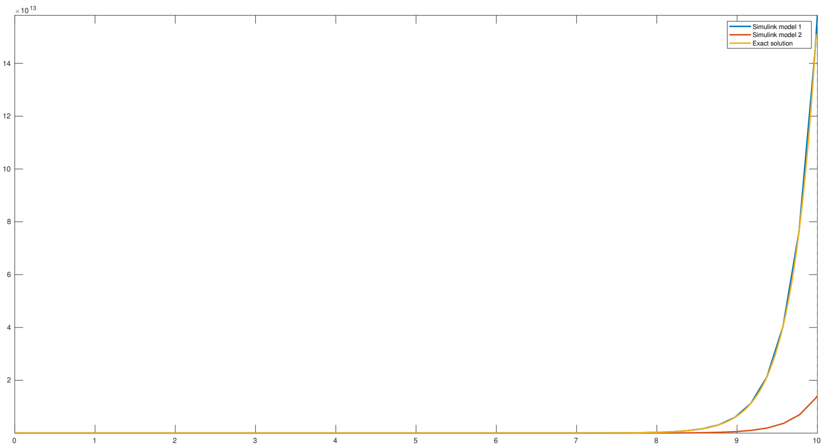
Matlab code with solution using Laplace Transform:

```
syms t x(t) s X;
% Given equation
dx = diff(x,t);
d2x = diff(x,t,2);
equation = d2x == dx + 7*x + t + 3;
% Laplace transform
trans = laplace(equation,t,s);
% Substituting initial conditions
trans = subs(trans, [laplace(x,t,s), dx(0), x(0)], [X,4,3]);
% Solving for X
sol = solve(trans,X);
% Inverse Laplace transform
sol =  ilaplace(sol, s, t);
% Plotting the solution
fplot(sol, [0 10])
```

Matlab code with solution of an equation with dsolve:

```
syms x(t);
equation = diff(x,t,2) == diff(x,t) + 7*x + t + 3;
% Adding conditions for given IVP
cond1 = x(0) == 3;
Dx=diff(x,t);
cond2 = Dx(0) == 4;
cond = [cond1,cond2];
% Using dsolve to obtain exact symbolic solution
sol = dsolve(equation,cond);
% Plotting solution
fplot(sol,[0 10]);
```

1.1 Plot of the solution



Plots of solutions obtained with different methods. Solution with symbolic Laplace transform was omited, because it differs from solution obtained by dsolve() less than by 10^{-12}

Plotting code. Here, $sim1$ and $sim2$ are solutions obtained with Simulink schemas

```
plot(sim1.time,sim1.data,sim2.time,sim2.data,'LineWidth',2);
hold on;
fplot(sol,'LineWidth',2);
axis([0 10 0 inf]);
legend('Simulink model 1','Simulink model 2',
        'Exact solution')
```

2 Task 2.

Given system:

$$3x'' + 3x' - 3 = 2t - 2, y = 3x'$$

After simplification:

$$x'' = -x' + \frac{2}{3}t + \frac{1}{3}$$

$$\begin{cases} x = x_1 \\ x_1' = x_2 \\ x_2' = -x_2 + \frac{2}{3}t + \frac{1}{3} \end{cases}$$

Hence, state-space representation of our system is:

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$

3 Task 3.

Given system:

$$\begin{cases} 3\ddot{\ddot{x}} + 2\dddot{x} - 3\ddot{x} + 2\dot{x} - 3 = u_1 + 5u_2 \\ y = \dot{x} + u2 \end{cases}$$

Convert to space-state:

$$\begin{cases} x = x_1 \\ \dot{x_1} = x_2 \\ \dot{x_2} = x_3 \\ \dot{x_3} = x_4 \\ \dot{x_4} = -\frac{2}{3}x_4 + x_3 + -\frac{2}{3}x_2 + \frac{1}{3}u_1 + \frac{5}{3}u_2 + 1 \\ y = x_2 + u_2 \end{cases}$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2}{3} & 1 & -\frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & \frac{1}{3} & \frac{5}{3} \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

4 Task 5.

Python code listing to convert ODE to state-space:

```
import numpy as np
#n - degree of the polynomial
n = 3
# a - array of coefficients: [ak ak-1 ... a0]
a = np.array([1,-1,-7,])

# normalization
a = a[1:] / a[0]
a=-a
# for convenience
a=np.flip(a)

# state matrix
A = np.zeros((n-1, n-1))
A[n-2 , 0:] = a
A[0:(n-2),1:] = np.eye(n-2)

print("Matrix A:")
print(A)
b=np.array([1])
#In this case we have only one input value - b_0
print("Matrix B:")
print(b)
```

Python code to solve DE with *scipy.integration.odeint*.
It uses vector of coeffitients and matrix from above model:

```
import scipy.integrate as sp_int
import numpy as np
import matplotlib.pyplot as plt
import convert

#derivative functions - default and in SS form
def derivativeDE(x,t):
    dx = x[1:].copy()
    dx = np.append(dx,convert.a.dot(x))
    return dx

def derivativeSS(x,t):
    dx=convert.A.dot(x)
    return dx

#initial conditions
initial = np.array([4,3])
#time samples
time = np.linspace(0, 10, 1000)
```

```
20 #solution and plots
21 sol1 = sp_int.odeint(derivativeDE,initial,time)
22 sol2 = sp_int.odeint(derivativeSS,initial,time)
23 plt.subplot(1,2,1)
24 plt.plot(time, sol1)
25 plt.xlabel('time')
26 plt.ylabel('x(t)')
27 
28 plt.subplot(1,2,2)
29 plt.plot(time, sol2)
30 plt.xlabel('time')
31 plt.ylabel('x(t)')
32 plt.show()
33 
34 e,v=np.linalg.eig(convert.A)
35 print(e)
```

According to eigenvalues, obtained on string 34 of code above, which are

$$[-2.1925824 \quad 3.1925824],$$

The ODE from task1 is unstable, and its solution diverges.

5 Used software.

- Python 3.8.1
- Matlab R2018b 9.5.0

All software was run under Manjaro Linux with 5.4.18-rt kernel