Control Theory: HW1

Temurbek Hudzhaev

February 2020

Contents

1 Preparation

Variant is: **o**

2 Solve second order diff equation.

$$\ddot{x} + 2\dot{x} - 3x = sin(4t)$$

A) Draw a schema in Simulink

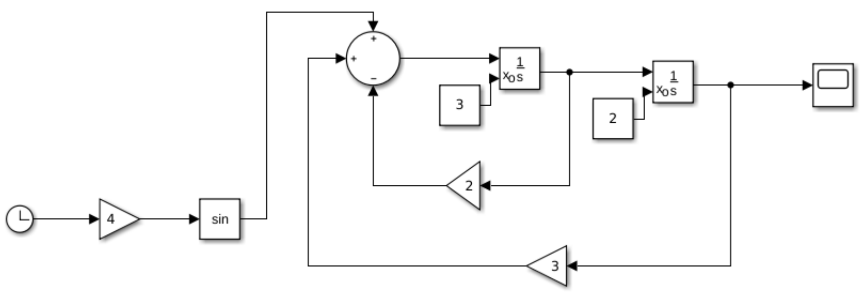

Figure 1: Simulink Model

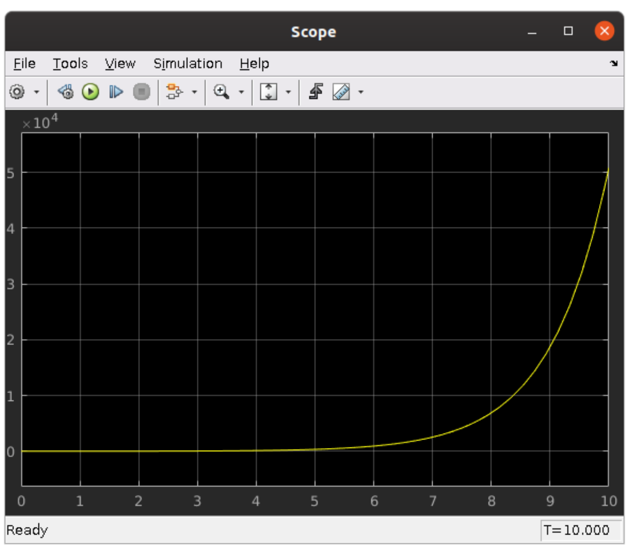

Figure 2: Plot

B) Draw a schema in Simulink (use transfer func block).

$$\frac{d}{dt} = p$$

$$p^2x + 2px - 3x = sin4t$$

$$(p^2 + 2p - 3)x = sin4t$$

$$(p^2 + 2p - 3)x = sin4t$$

$$x = \frac{1}{p^2 + 2p - 3} sin4t$$

$$W(p) = \frac{1}{p^2 + 2p - 3} sin4t$$

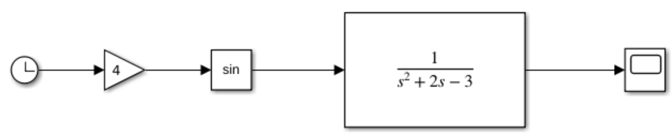

Figure 3: Simulink Model with Transfer Function

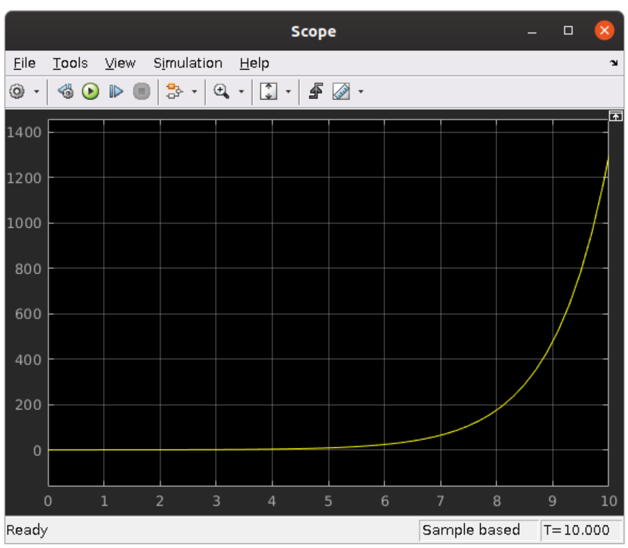

Figure 4: Plot

C) Solve diff equation with matlab function and draw a plot in matlab.

```
syms x(t)
Dx = diff(x);

ode = diff(x,t,2) == -2 * Dx + 3 * x + sin(4 * t);
cond1 = Dx(0) == 3;
cond2 = x(0) == 2;

conds = [cond1 cond2];
xSol(t) = dsolve(ode, conds);
xSol = simplify(xSol);

var = 0:0.05:15;
plot(var,xSol(var));
```

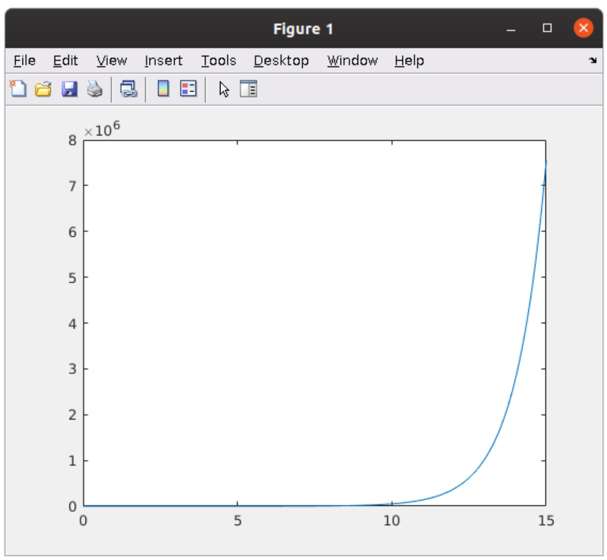

Figure 5: Plot

D) Solve diff equation with Laplace transform in matlab.

```
syms t s X

RHS = laplace(sin(4*t));

X1 = s * X - 2;
X2 = s * X1 - 3;

sols = solve(X2 + 2 * X1 - 3 * X - RHS, X);
solt = ilaplace(sols,s,t);


pretty(solt)

fplot(solt ,[0,8])
grid on
```

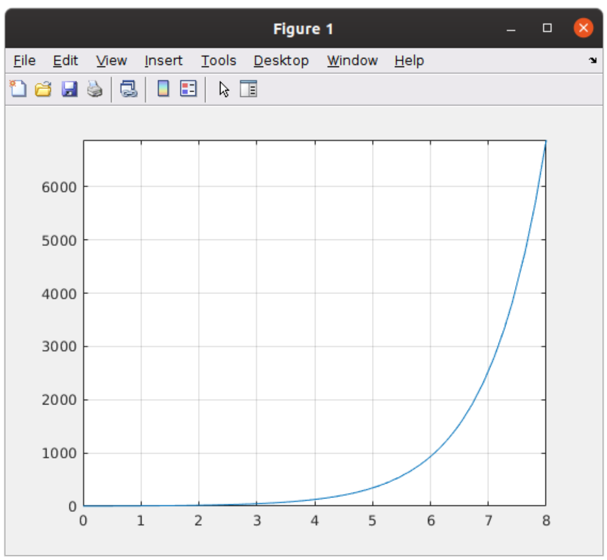

Figure 6: Plot

3 Find State Space Model of the system.

$$\begin{cases} \ddot{x} = t + 3 \\ y = x + 2\dot{x} \end{cases} \tag{1}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} (t + 3) \tag{2}$$

$$y = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{3}$$

4 Find State Space Model of the system.

$$\begin{cases} \ddddot{x} - 2\dddot{x} + \ddot{x} - \dot{x} + 5 = u_1 + u_2 \\ y = 2x + \dot{x} - u_1 \end{cases} \tag{4}$$

$$\ddddot{x} = \dot{x} - \ddot{x} + 2\dddot{x} + u_1 + u_2 - 5$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \\ \ddddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 5 \end{bmatrix} \tag{5}$$

$$y = \begin{bmatrix} 2 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} - u_1 \tag{6}$$

5 Write a function in python that converts any ODE

```python
import numpy as np

k = 5
def convert_to_SS(a, b0):
    b0 = b0/a[k]
    a = a[:k] / a[k] # a0 to ak-1 divided to a_k

    A = np.zeros((k,k))
    A[0: (k - 1), 1:k] = np.eye(k - 1) # 1..n-1 rows and 2..k columns are identity matrix
    A[k - 1, 0:] = -a #last row is a multiplied by -1, because of right hand side

    B = np.zeros(k)
    B[k - 1] = b0
    return A, B


a = np.random.rand(k + 1) # k + 1, because indexes are from 0 to k
b0 = np.random.rand(1)


```

```python
A,B = convert_to_SS(a,b0)

print("ODE with following random coefficients:", a, end = '')
print("and b0:",b0)
print("matrix A:", A)
print("matrix B:", B)
```

6 Write functions in python that solves ODE and its state space representation.

```python
import numpy as np

k = 2
def convert_to_SS(a, b0):

    A = np.zeros((k,k))
    A[0: (k - 1), 1:k] = np.eye(k - 1) # 1..n-1 rows and 2..k columns are identity matrix
    A[k - 1, 0:] = -a #last row is a multiplied by -1, because of right hand side

    B = np.zeros(k)
    B[k - 1] = b0
    return A, B


a = np.array([-3,2,1]) # k + 1, because indexes are from 0 to k
b0 = np.random.rand(1)
print("ODE with following random coefficients:", a, end = '')
print("and b0:",b0)


b0 = b0/a[k]
a = a[:k] / a[k] # a0 to ak-1 divided to a_k


A,B = convert_to_SS(a,b0)

print("matrix A:", A)
print("matrix B:", B)
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

```python
import math
def rhs(t):
    return math.sin(4*t)

def solveODE(x, t):
    dx = np.zeros(k)
    dx[0:(k-1)] = x[1:k]
    dx[k-1] = -a.dot(x) + rhs(t)
    return dx

def solveSS(x, t):
    return A.dot(x) + rhs(t)

t = np.linspace(0, 10, 5000)
x0 = np.array([3, 2])

sol1 = odeint(solveODE, x0, t)
sol2 = odeint(solveSS, x0, t)


plt.subplot(1,2,1)
plt.plot(t, sol1)
plt.xlabel('t')
plt.ylabel('x')


plt.subplot(1,2,2)
plt.plot(t, sol1)
plt.xlabel('t')
plt.ylabel('x')
e, v = np.linalg.eig(A)
print(e)
```

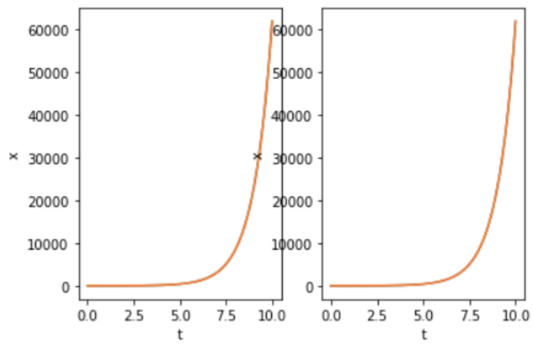

Figure 7: Plot of ODE and SS

We have eigenvalues $1 + 0 * i$ and $-3 + 0 * i$ which and because of eigenvalue with positive real part we can conclude that system is **not stable**. As the solution is exponential with the eigenvalues then the system will **diverge** as the power is positive