Control Theory Home Work 1

Utkarsh Kalra BS18-03

Varient f

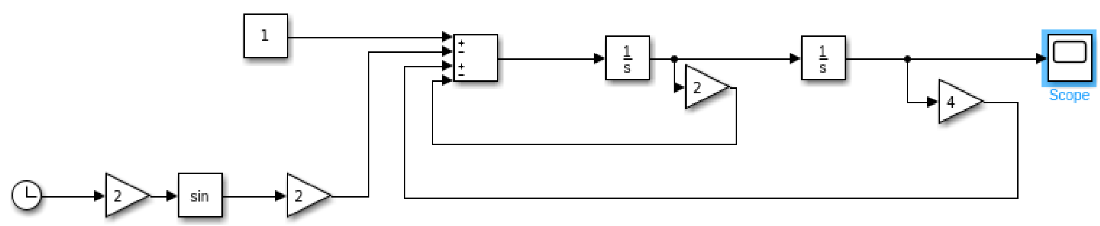
1 Git repo

https://github.com/kalraUtkarsh/Control-Theory-Utkarsh-Kalra

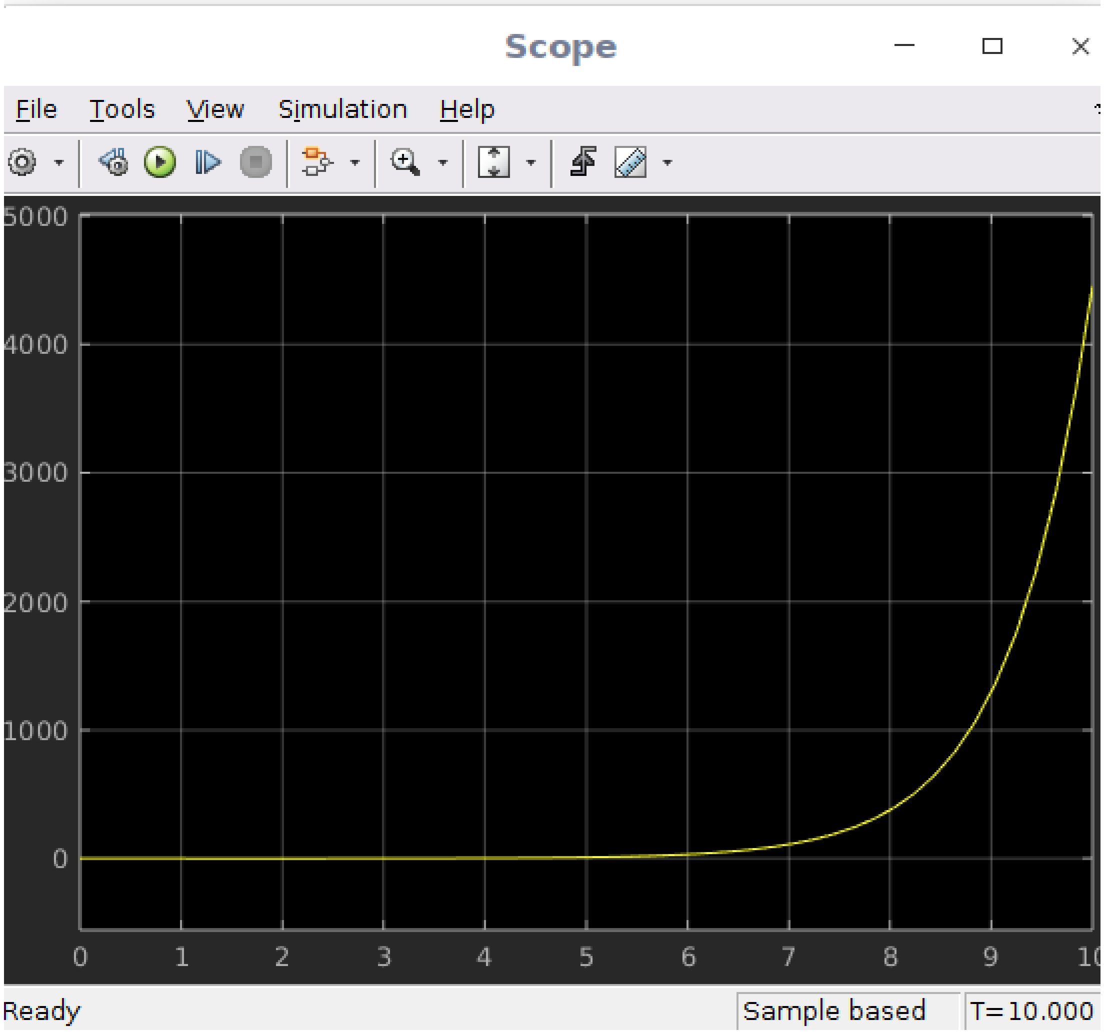
2 x" + 2 sin 2t + 2x0 = 4x + 1; x'(0) = 3; x(0) = 0

2.1 2(A)

2.1.1 Schema in Simulink without transfer funtion



2.1.2 Plot of the simulink without Transfer Function

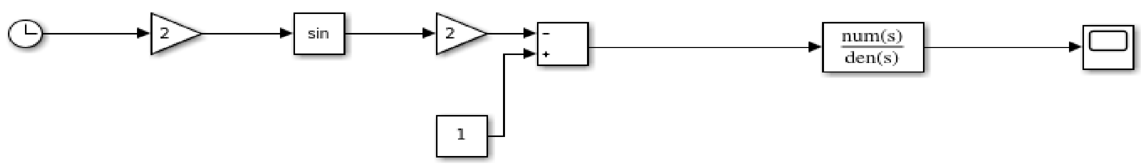


2.2 2(B)

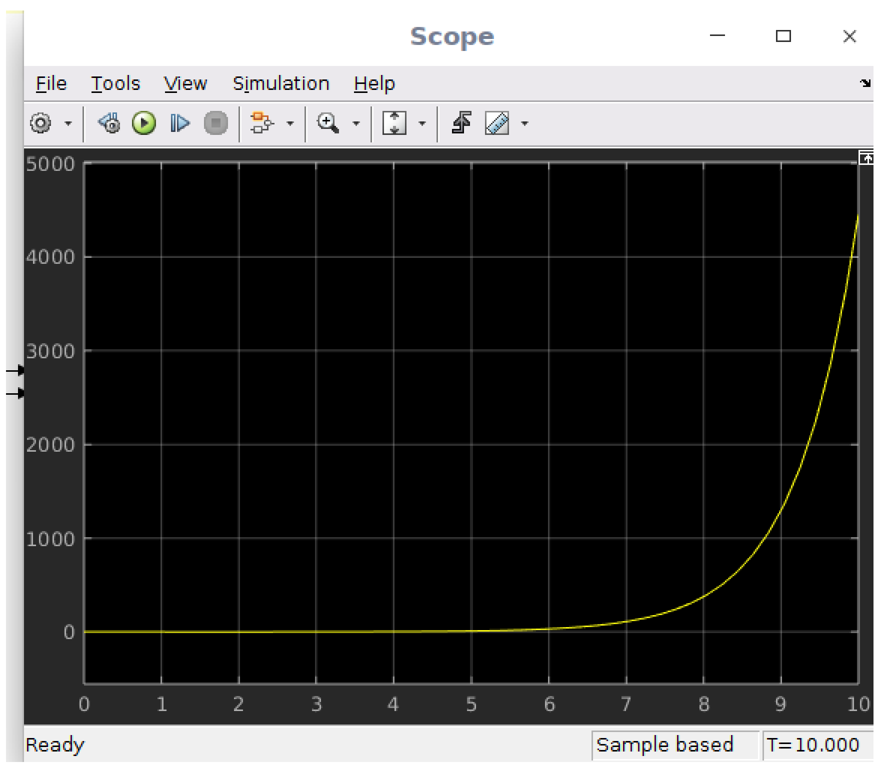
2.2.1 Calculation of Transfer Function

let d/dt = p so
$xp^2 - 4x + 2xp = 1 - 2sin(2t)$
$x(p^2 - 4 + 2p) = -sin(2t) + 1$
$x = [1 - 2sin(2t)]/[p^2 + 2p - 4]$
$input = 1 - 2sin(2t)so$
$TransferFunction = [1]/[p^2 + 2p - 4]$

2.2.2 Simulink Schema using Transfer Funtion block



2.2.3 Plot for Simulink using Transfer Function block

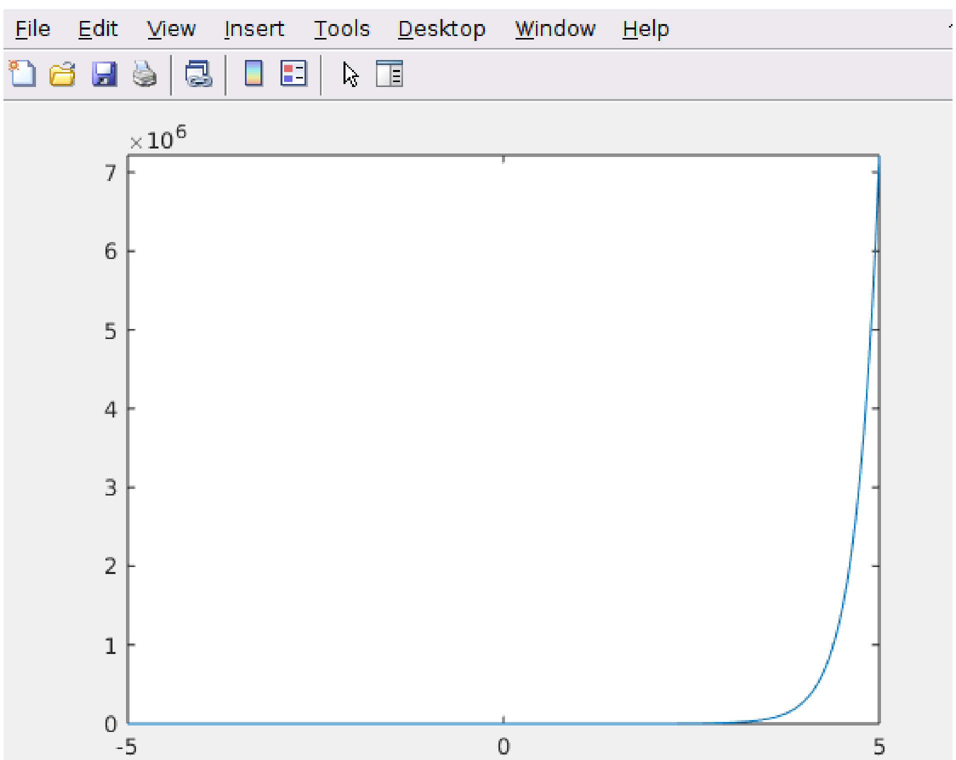


2.3 2(C)

2.3.1 Solving the Differential equation with Matlab

```
%solving the ode

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

ode = diff(x,t,2) == 4*x + 1 - 2*sin(2*t) + 2*diff(x,t) ;
cond1 = x(0) == 0;
cond2 = Dx(0) == 3;

conds = [cond1 cond2];
XSol(t) = dsolve(ode,conds);
XSol = simplify(XSol);
fplot(XSol)
```

2.3.2 The plot of the Ode solved using matlab



2.4 2(D)

2.4.1 Solving the Differential equation using Laplace in Matlab

```
%solving the differential equation with laplace

clc; clearvars;
syms x(t) t s X F;
Dx = diff(x,t);
D2x = diff(x,t,2);
ode = D2x == 4*x + 1 - 2*sin(2*t) + 2*Dx ;
F = laplace(ode, t, s);
F = subs(F, laplace(x,t,s), X);
F = subs(F,x(0), 0);
F = subs(F, subs(Dx, t, 0), 3);
X = solve(F, X);
x = ilaplace(X);
```

3 Converting into State Space Model

x”+2x’+2x=t+5, y=x’+2t
x”=t+5-2x’-2x

$$\begin{bmatrix} x' \\ x'' \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x \\ x' \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ x' \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix}$$

4 Converting into state space Model

$x'''' + 2x''' + 2x'' + 2x' - 6 = 2u_1 + 3u - 2, y = x' + u_1 + 2u_2$

$$\begin{bmatrix} x' \\ x'' \\ x''' \\ x'''' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & -2 & -2 \end{bmatrix} \begin{bmatrix} x \\ x' \\ x'' \\ x''' \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

$$\mathrm{y} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ x' \\ x'' \\ x''' \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

5 Function in Python to convert any ODE to State Space Representation

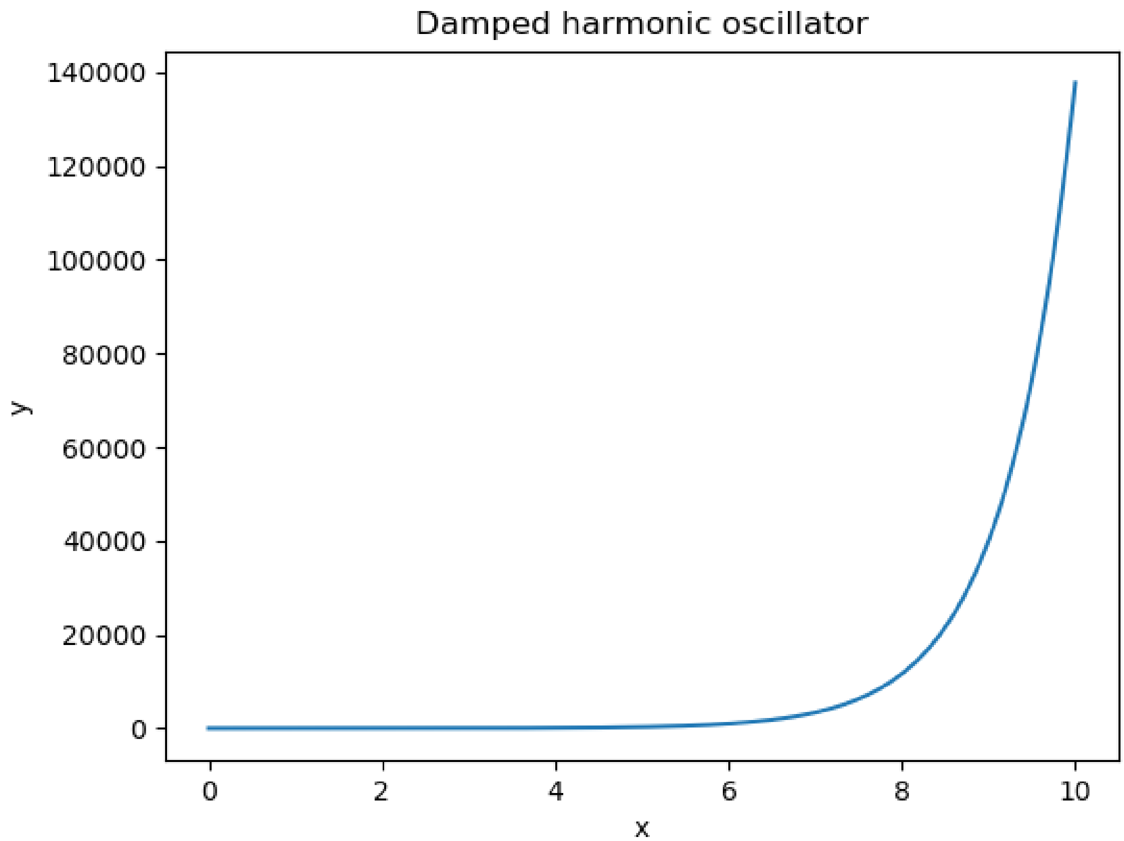
```python
import numpy as np

def odess(coff_a, b0):
    k = len(coff_a)-1;

    a = np.zeros(shape=(k,k))
    for i in range (k-1):
        a[i][i+1] = 1
    for i in range(k):
        a[-1][i] = -coff_a[i]/ coff_a[k]

    b = np.zeros(shape=(k,1))
    b[-1] = b0/coff_a[k]

    return a, b
```

6 Functions in python to solve ODE and the State Space Models

6.0.1 Code

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def dU_dx(U, x):
    # Here U is a vector such that y=U[0] and z=U[1]. This function should retur
    return [U[1], -2*U[1] + 4*U[0] - np.sin(2*x)]
U0 = [0, 3]
xs = np.linspace(0, 10, 200)
Us = odeint(dU_dx, U0, xs)
ys = Us[:,0]
plt.xlabel("x")
plt.ylabel("y")
plt.title("Damped harmonic oscillator")
plt.plot(xs,ys);
plt.show()
```

6.0.2 PLot


Damped harmonic oscillator
140000
120000
100000
80000
60000
40000
20000
0
y
0
2
4
6
8
10
x