Homework 1

Variant: C

Question 1

Given equation:

$$x'' + 2x' = -2x + 3sin(t)$$

1. Figure 1 demonstrates a solution of the 2^{nd} order differential equation in Simulink without using transfer func block. Figure 2 is a plot of simulation made by the scheme mentioned above.

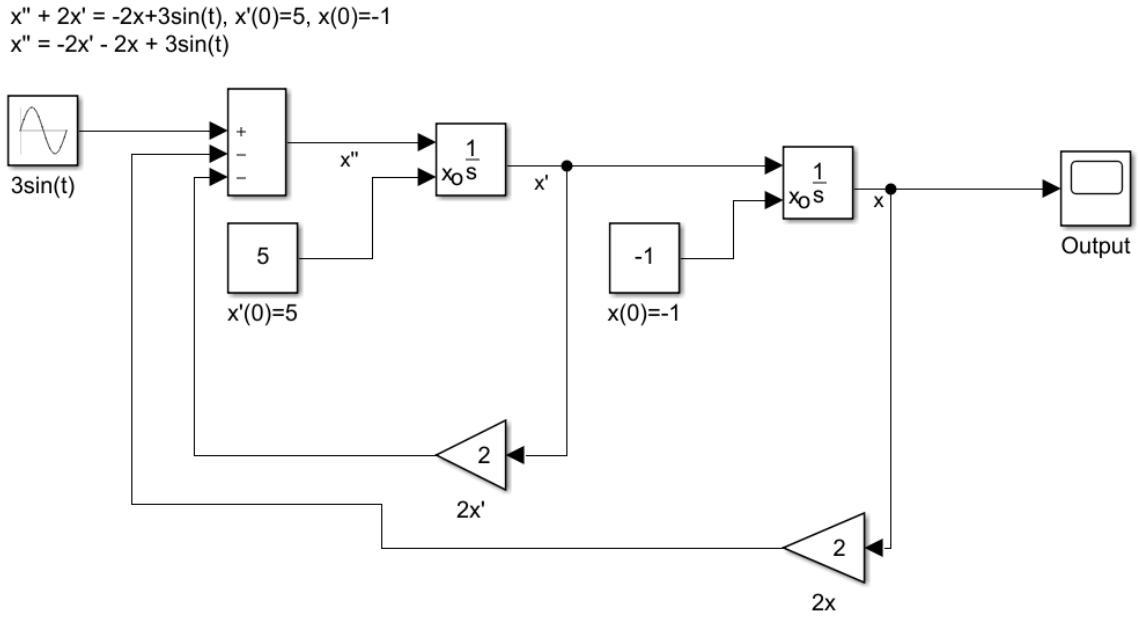

Figure 1: Simulink schema (without transfer func)

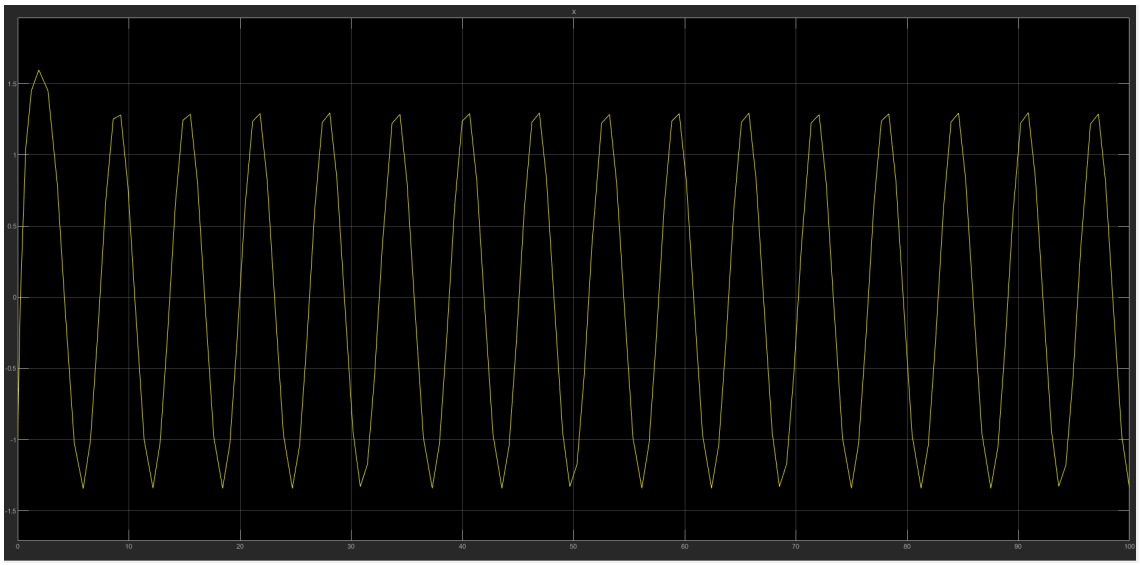

Figure 2: Simulink schema (without transfer func) output

2. Figures 3 and 4 shows the scheme and plot of the equation using transfer func block.The initial conditions were changed to zeros, since transfer function can't accept any (`Unfortunately, one can only solve problems with zero initial conditions` from 'Solving differential equations using simulink' by R. Herman, page 72)

$$x'' + 2x' = -2x+3\sin(t),\ x'(0)=0,\ x(0)=-0$$
$$x'' + 2x' + 2x = 3\sin(t)$$

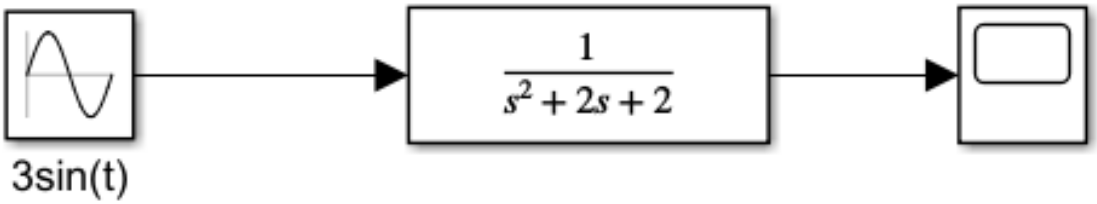

Figure 3: Simulink schema (with transfer func)

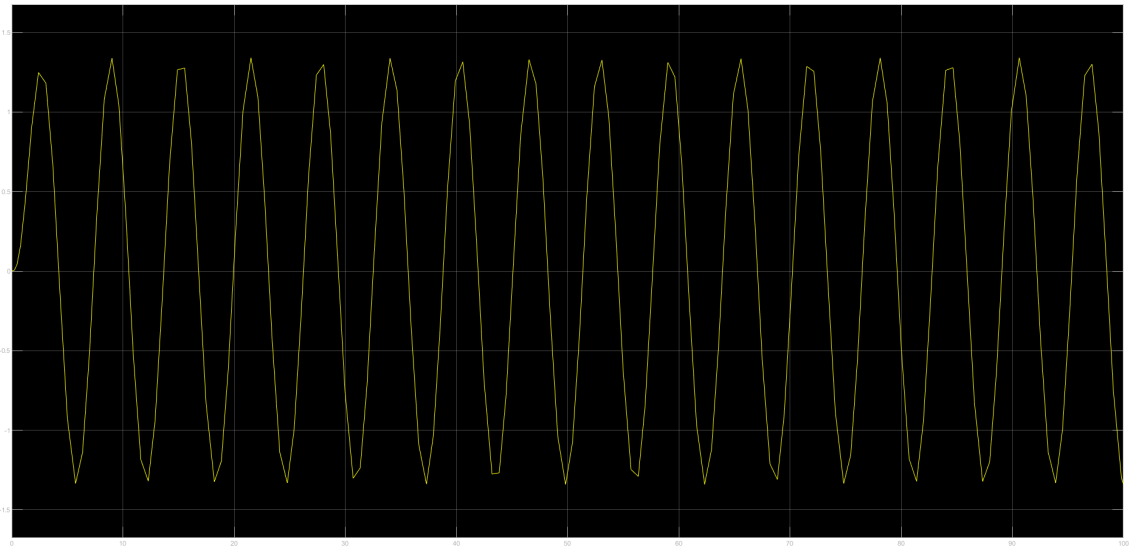

Figure 4: Simulink schema (with transfer func) output

3. Matlab code for solving the differential equation using Symbolic Toolbox is shown at listing 1. Figure 5 shows the graph of the solution.

Listing 1: Matlab solution via dsolve

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

ode = diff(x,t,2) == -2*diff(x,t,1) -2*x+3*sin(t);
cond1 = x(0) == -1;
cond2 = Dx(0) == 5;
conds = [cond1 cond2];
xSol(t) = dsolve(ode,conds);
xSol = simplify(xSol)

a = 0;
b = 100;

fplot(xSol, [a b])
```

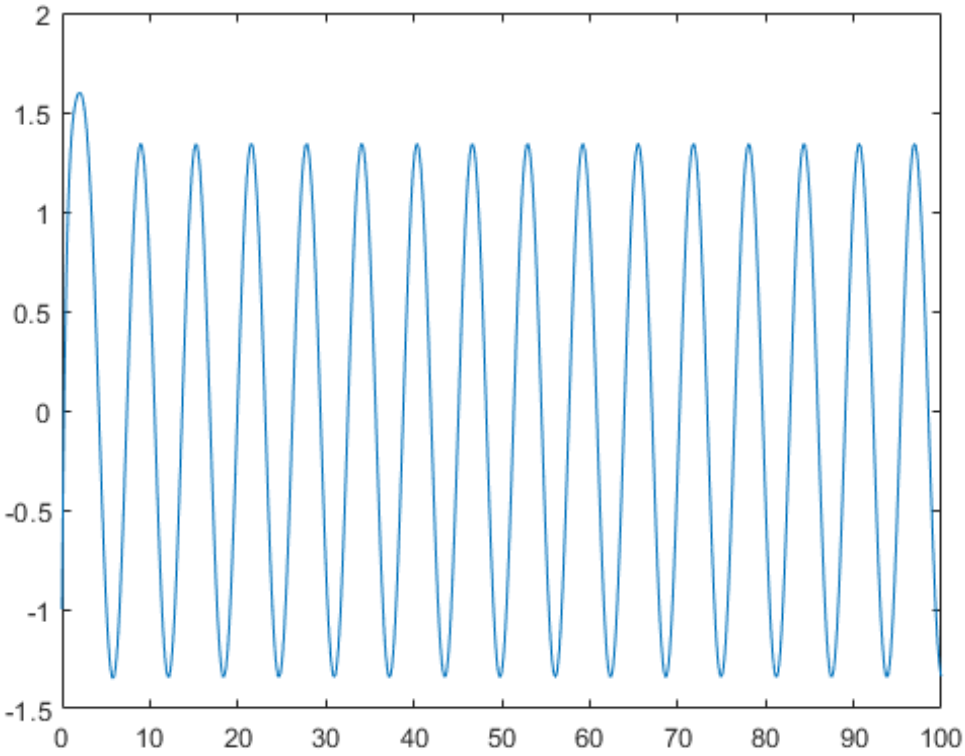

Figure 5: Matlab solution (dsolve)

4. Listing 2 shows a solution using Laplace transform and Symbolic Toolbox. The graph is present on the figure 6

Listing 2: Matlab solution via transfer func

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

ode = diff(x,t,2) == -2*diff(x,t,1) -2*x+3*sin(t);
odeLT = laplace(ode, t,s);

syms X_LT
odeLT = subs(odeLT, laplace(x,t,s), X_LT);

X_LT = solve(odeLT, X_LT);
xSol = ilaplace(X_LT, s, t);
xSol = simplify(xSol);
xSol = subs(xSol, [x(0) Dx(0)], [-1 5]);

```

```
15 fplot(xSol, [0 100])
```

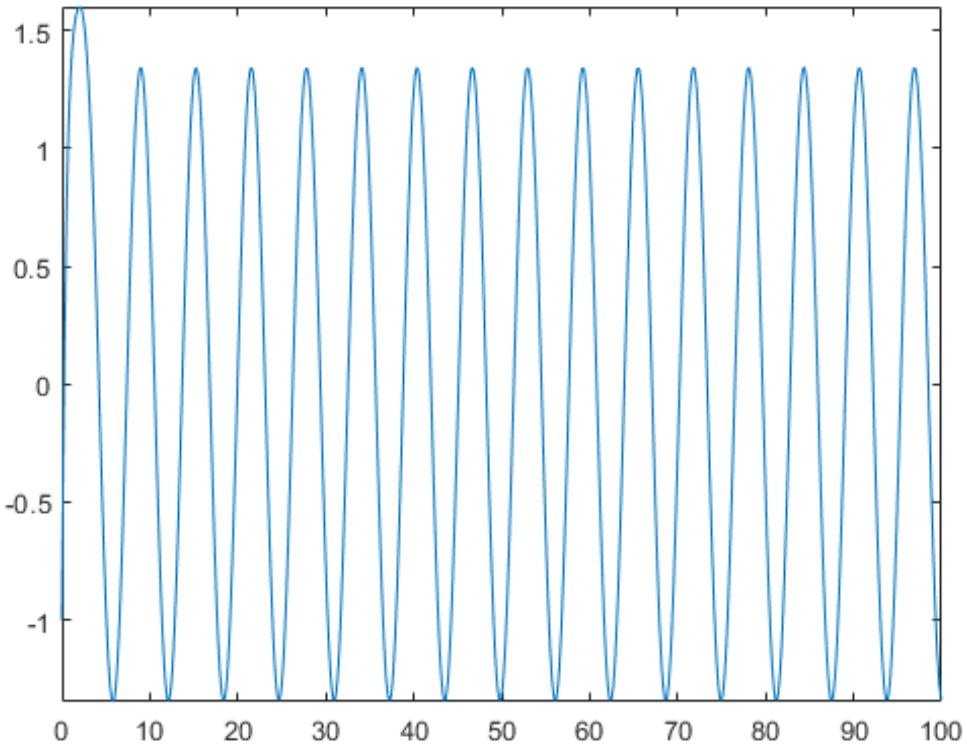

Figure 6: Matlab solution (dsolve)

Question 2

Find State Space Model of the system.

$$x'' - x' + 5 = t + 1,\ y = 2x + x'$$

$$x'' = x' - 4 + t$$

$$\begin{cases} \begin{bmatrix} x' \\ x'' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ x' \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot (t-4) \\ y = \begin{bmatrix} 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ x' \end{bmatrix} + 0 \cdot (t-4) \end{cases}$$

Question 3

Find State Space Model of the system.

$$2x^{(4)} + x^{(3)} - 3x'' + 4x' - 3 = u_1 - 2u_2,\ y = 4x' - u_1$$

$$x^{(4)} = -\frac{1}{2}x^{(3)} + \frac{3}{2}x'' - 2x' + \frac{3}{2} + \frac{1}{2}u_1 - u_2$$

$$x^{(4)} = -\frac{1}{2}x^{(3)} + \frac{3}{2}x'' - 2x' + \frac{1}{2}u_1 - (u_2 - \frac{3}{2})$$

$$\begin{cases} \begin{bmatrix} x' \\ x'' \\ x^{(3)} \\ x^{(4)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & \frac{3}{2} & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} x \\ x' \\ x'' \\ x^{(3)} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{2} & -1 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 - \frac{3}{2} \end{bmatrix} \\ y = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ x' \\ x'' \\ x^{(3)} \end{bmatrix} + \begin{bmatrix} -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 - \frac{3}{2} \end{bmatrix} \end{cases}$$

Question 4

Write a function in python that converts any ODE of power n to the state space representation.
ODE form:

$$a_k y^{(k)} + a_{k-1} y^{(k-1)} + \cdots + a_2 y'' + a_1 y' + a_0 y' = b_0$$

Desired output format:

$$\begin{bmatrix} y' \\ y'' \\ \vdots \\ y^{(k)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ -\frac{a_0}{a_k} & -\frac{a_1}{a_k} & -\frac{a_2}{a_k} & \dots & -\frac{a_{k-1}}{a_k} & -\frac{a_{k-1}}{a_k} \end{bmatrix} \cdot \begin{bmatrix} y \\ y' \\ y'' \\ \vdots \\ y^{(n-2)} \\ y^{(n-1)} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \frac{1}{a_k} \end{bmatrix} b_0$$

The solution is presented at listing 3. Usage: listing 4. Output example:

```
Coefficients:  [0.99886853 0.92023046 0.75979535 0.66250064 0.7541122  0.52864808
 0.73775728]
State:
[
 [ 0.          1.          0.          0.          0.          0.        ]
 [ 0.          0.          1.          0.          0.          0.        ]
 [ 0.          0.          0.          1.          0.          0.        ]
 [ 0.          0.          0.          0.          1.          0.        ]
 [ 0.          0.          0.          0.          0.          1.        ]
 [-0.73859298 -0.5292469  -0.75496643 -0.66325109 -0.76065601 -0.92127285]
]
input_matrix:
 [0.          0.          0.          0.          0.          1.00113276]
```

Listing 3: Converting an ODE of degree n to StateSpace format

```python
import numpy as np

def constructStateSpace(coefficients:np.ndarray) -> (np.ndarray, np.ndarray)
    :
    #Used comments instead of doctring due to latex formatting problem
    #degree = len(coefficients)
    #:param coefficients: 1 dimensional array in format [ak a(k-1) ... a0]
    #:return: state matrix ([degree-2]x[degree-2]) and input matrix [degree
        -2]x[1]

    if len(coefficients.shape) != 1:
        raise ValueError(coefficients are expected to be passed as 1
            dimensional array)
    deg = len(coefficients)

    input_matrix =np.zeros((deg-1))
    input_matrix[-1] = 1/coefficients[0] # normalize coefficient of input
```

```
15
16      coefficients = coefficients[1:]/coefficients[0] # normalize coefficients
17      coefficients = np.flip(coefficients) # flip coefficients before adding
            to matrix
18
19      state_matrix = np.zeros((deg-1, deg-1))
20      state_matrix[-1, 0:] = -coefficients
21      state_matrix[:-1, 1:] = np.eye(deg-2)
22
23
24      return state_matrix, input_matrix
```

Listing 4: Usage of constructStateSpace func.

```
1  degree = 7
2  coefs = np.random.rand(degree) #[ak a(k-1) ... a0]
3  b0 = np.random.rand()
4  state_m,input_m =  constructStateSpace(coefs)
5  def input_func(x,t):
6      return b0
```

Question 5

1. Write function in python that solves ODE. Since the ODE's order is n, solving technique is very similar to StateSpace, therefore some variables will be used from the state matrix. The code is on listing 5.

Listing 5: Solving linear ODE

```
1  from scipy.integrate import odeint
2  from typing import Tuple
3
4  def constructLabels(var:str, cnt:int) -> Tuple[str]:
5      # Utility function to make labels.
6      # Ex: var = x, cnt = 2. Return = (x, x')
7      # this function renders poorly in lstlisting, see the source code
8      # ...
9
10 time = np.linspace(0,100,10000)
11 x0 = np.asarray([-1, 5]) # x, x'
12 n = len(coefs)
13 def LinODE(x, t):
14     dx = np.zeros(n-1)
15     dx[0] = state_m[-1].dot(x) + input_func(x,t)
```

```
    dx[1:] = x[0:(n-2)]
    return dx

solution_ode = odeint(LinearODE, np.flip(x0), time)
lines=plt.plot(time, solution_ode)
plt.legend(lines, reversed(constructLabels(x, len(lines))))
plt.xlabel('time_ode')
plt.show()
```

2. Write functions in python that solves ODE's state space representation. The function is presented at listing 6

Listing 6: Usage of constructStateSpace func.

```
from scipy.integrate import odeint
from typing import Tuple

def constructLabels(var:str, cnt:int) -> Tuple[str]:
    # Utility function to make labels.
    # Ex: var = x, cnt = 2. Return = (x, x')
    # this function renders poorly in lstlisting, see the source code
    # ...
def StateSpace(x,t):
    return state_m.dot(x) + input_m * input_func(x,t)

time = np.linspace(0,100,10000)
x0 = np.asarray([-1, 5]) # x, x'

solution = odeint(StateSpace, x0, time)

lines=plt.plot(time, solution)
plt.legend(lines, constructLabels(x, len(lines)))
plt.xlabel('time')
plt.show()
```

3. Test functions on equations of task 2 (Question 1).
   (a) Output is presented at figure 7.

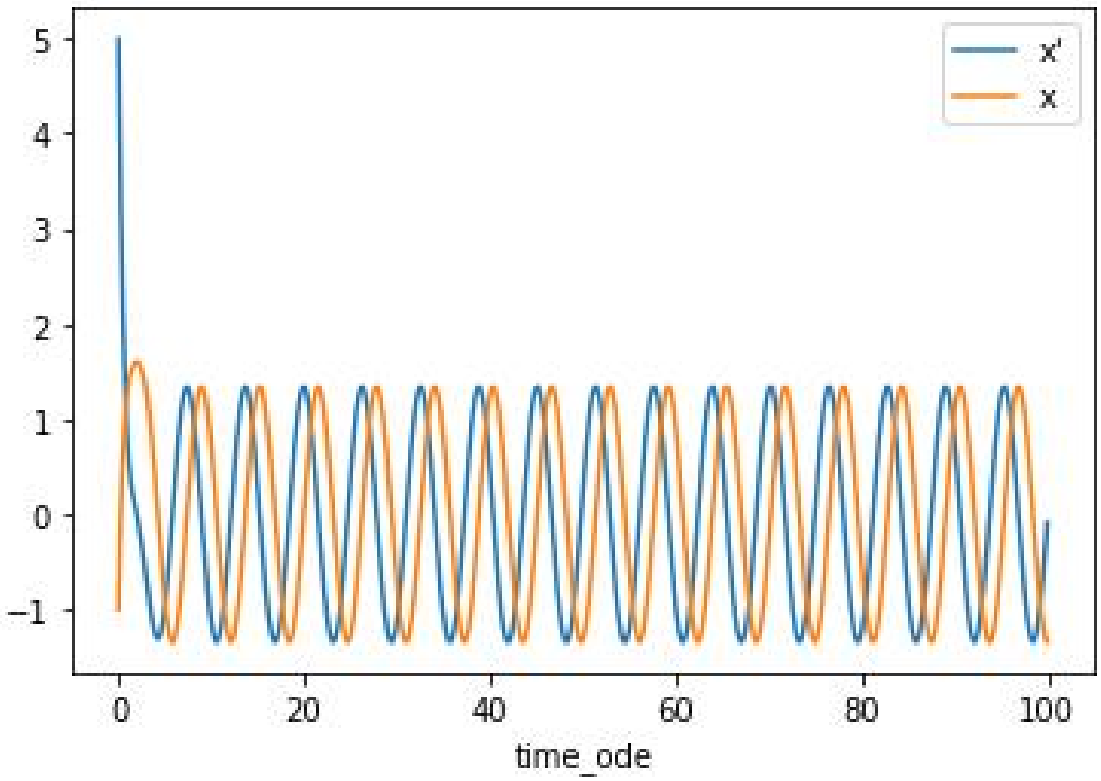

Figure 7: ODE solution of Question 1

(b) StateSpace representation output at figure 8

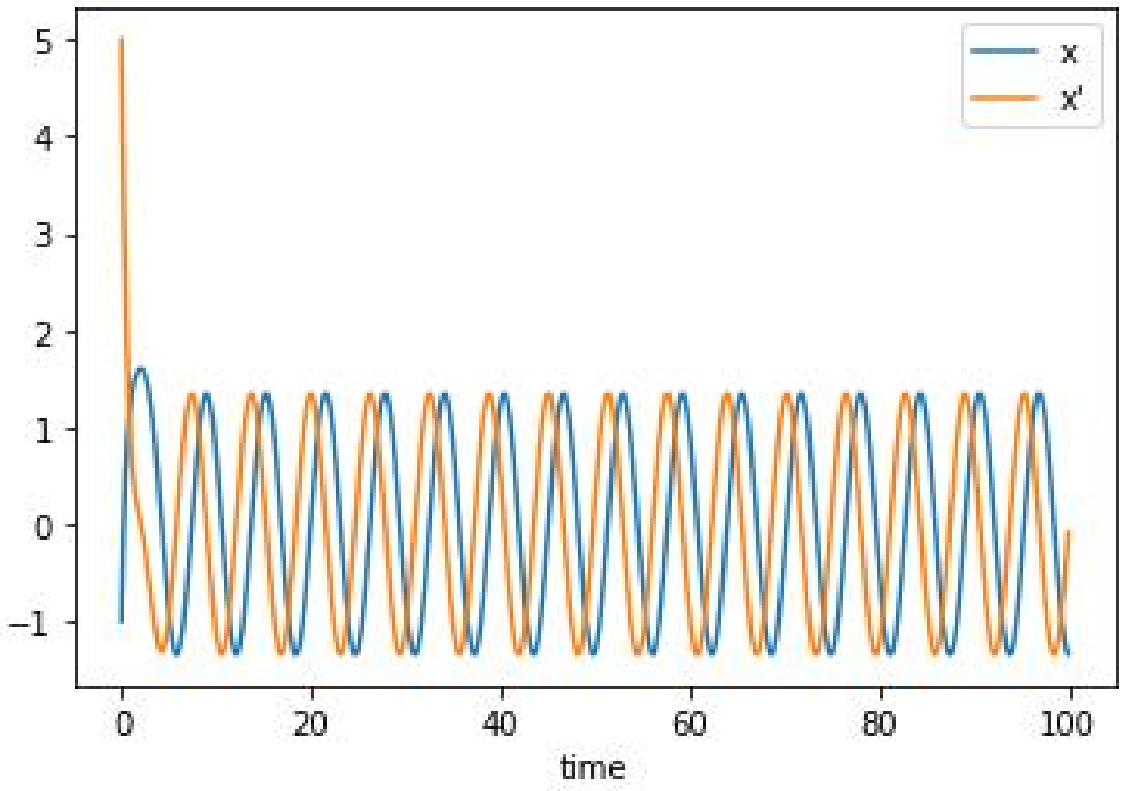

Figure 8: StateSpace representation solution of Question 1

Since the right-hand size contains the side the function of time, eigen-values can't tell anything about stability of a system. According to the plot 8, the solution is stable, but diverges.