Control Theory: Assignment 1

Due on February 14, 2020 at 11:59pm

Mike Ivanov

Artem Bakhanov (B18-03)

Contents

1 Introduction

I created a GitHub repository here. All the problems are solved by me, Artem Bakhanov, a student of Innopolis University. My variant is **j**.

2 Problem 2 (SO ODE)

$$4x'' - 4x' + 5t - 2x = 3, \quad x'(0) = 0,\ x(0) = -3 \tag{1}$$

2.1 Simulink schema (without TF blocks)

Before transforming the equation to a Simuling schema I got the expression of x'' in terms of x', x, and t:

$$x'' = x' + \frac{1}{2}x - \frac{5}{4}t + \frac{3}{4} \tag{2}$$

Initial conditions are 0 and -3 (from left to right).

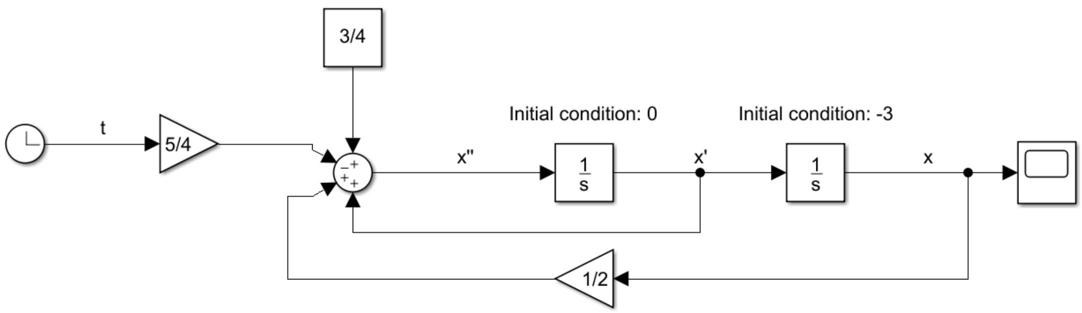


Figure 1: Simulink schema of the equation

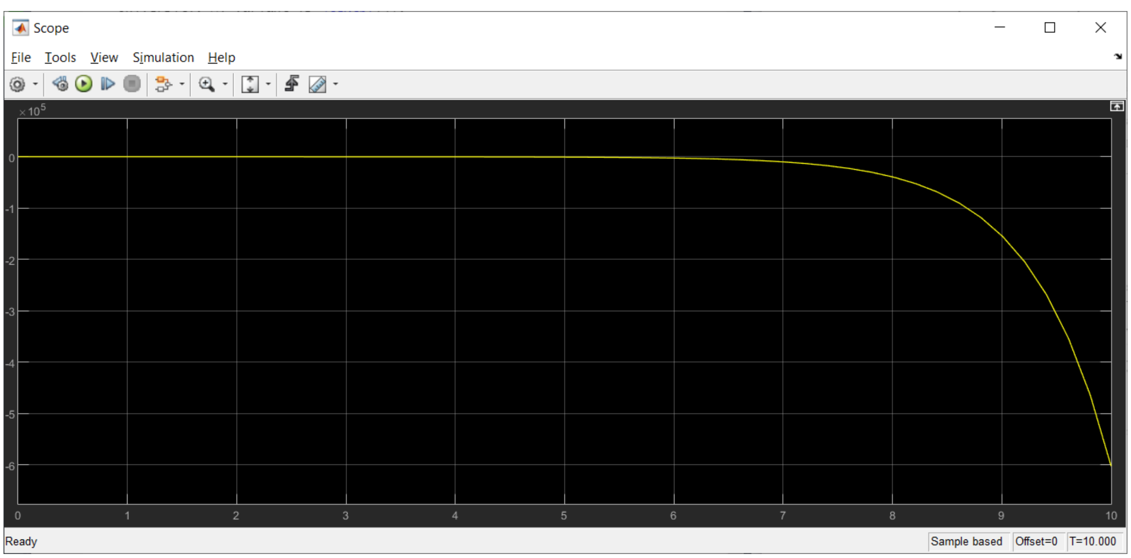


Figure 2: The plot of the solution

2.2 Simulink (with TF blocks)

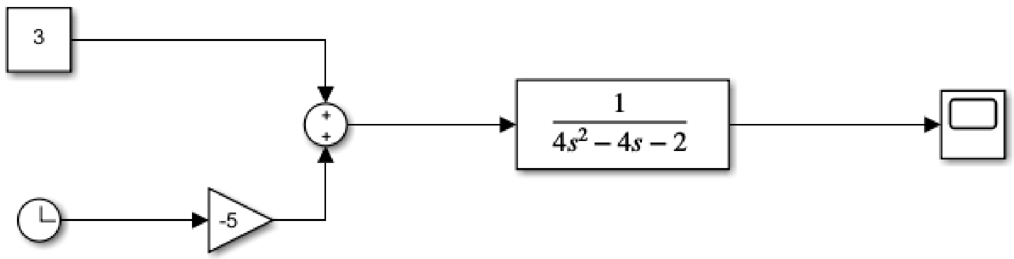


Figure 3: Simulink schema of the equation (TF)

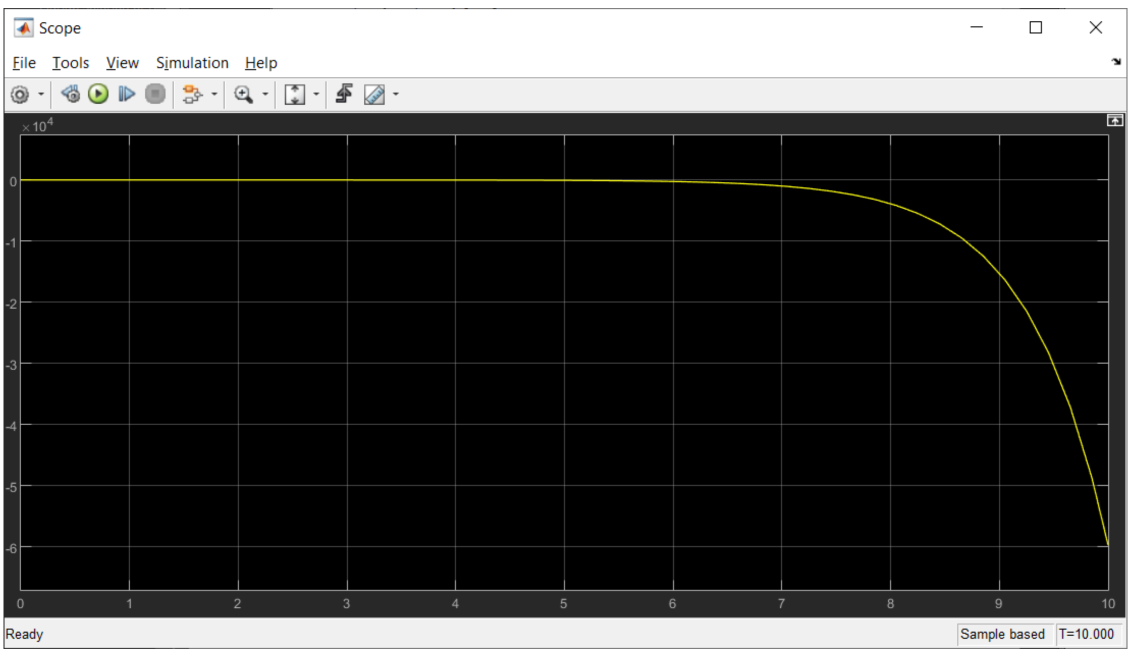


Figure 4: Simulink schema of the equation (TF)

This subsection is about Transforming the given equation to TF model.

$$4x'' - 4x' + 5t - 2x = 3, \quad x'(0) = 0,\ x(0) = -3 \tag{3}$$

Since it is impossible (at least with those tools we were given) to work with initial conditions I worked only with zero initial conditions. Let $u = 3 - 5t$:

$$4x'' - 4x' - 2x = u \tag{4}$$

It is easily transformed to TF:

$$4p^2X - 4pX - 2X = u \tag{5}$$

$$W(p) = \frac{1}{4p^2 - 4p - 2} \tag{6}$$

2.3 Matlab solution of the ODE

You can also find this code in task2c_j.m file.

Listing 1: Solution of the ODE in Matlab

```

% define the function
syms x(t)

% create symbolic functions for initial conditions
Dx = diff(x, t);

% define the ode
ode = 4 * diff(x, t, 2) - 4 * diff(x, t) + 5 * t - 2 * x == 3;

% define initial conditions
cond1 = x(0) == -3;
cond2 = Dx(0) == 0;
conds = [cond1 cond2];

xSol(t) = dsolve(ode, conds);
disp(xSol);

% draw a plot
figure
fplot(xSol, [0 10]);
```

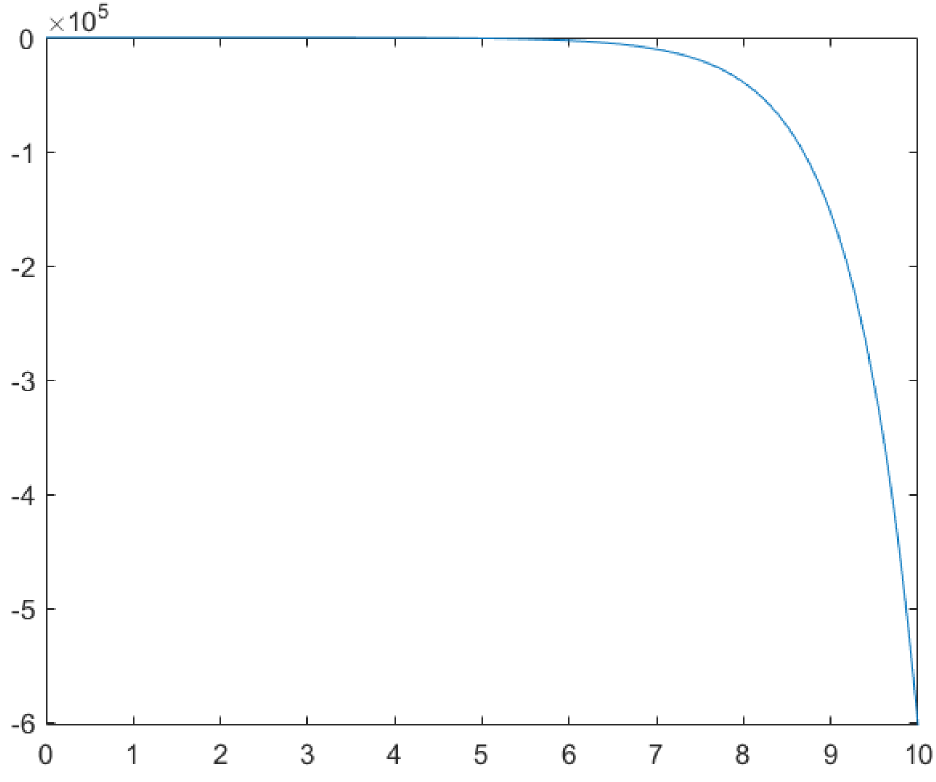


Figure 5: The plot of the Matlab solution

2.4 Laplace transform in Matlab

You can also find this code in task2d_j.m file.

Listing 2: Solution of the ODE using Laplace transform in Matlab

```
% define symbolic variables , X is Laplace transform of the solution
syms s t X;

% define x' and x''
X1 = s * X - (-3);
X2 = s * X1 - 0;

% define right-hand function and find its Laplace transform
f = 3 - 5 * t;
F = laplace(f);

% solve for X
Sol = solve(4*X2 - 4 * X1 - 2 * X - F, X);

% find the inverse Laplace transform of X
sol = ilaplace(Sol, s, t);

disp(sol);
% plot the solution graph
figure
fplot(sol, [0 10]);
```

3 ODE2SS (1)

Find the SS model of a system described by the following ODE:

$$x'' + 2x' - 3 = t + 5, \quad y = x' \tag{7}$$

The SS model will look like

$$\dot{x} = Ax + Bu \tag{8}$$

$$y = Cx + Du \tag{9}$$

Let $x = \begin{bmatrix} x \\ x' \end{bmatrix}$, $u = \begin{bmatrix} 1 \\ t \end{bmatrix}$. Let us express x'':

$$x'' = 0x - 2x' + t + 8 \tag{10}$$

Then $A = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 0 \\ 8 & 1 \end{bmatrix}$. It is easy to find matrices C and D. We are only interested in x'; thus, $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$ and $D = \begin{bmatrix} 0 & 0 \end{bmatrix}$. The final answer is:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 8 & 1 \end{bmatrix} u \tag{11}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \end{bmatrix} u \tag{12}$$

4 ODE2SS (2)

Find the SS model of a system described by the following ODE:

$$x'''' + 3x''' + 4x'' + 2x' - 6 = 2u_1 + 2u_2, \quad y = x' + u_1 + u_2 \tag{13}$$

The SS model will look like

$$\dot{x} = Ax + Bu \tag{14}$$

$$y = Cx + Du \tag{15}$$

Let $x = \begin{bmatrix} x \\ x' \\ x'' \\ x''' \end{bmatrix}$, $u = \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$. Then, obviously, $\dot{x} = \begin{bmatrix} x' \\ x'' \\ x''' \\ x'''' \end{bmatrix}$. Let us express x'''':

$$x'''' = 0x - 2x' - 4x'' - 3x''' + 6 + 2u_1 + 2u_2 \tag{16}$$

Then

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & -4 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 2 & 2 \end{bmatrix} \tag{17}$$

and

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \tag{18}$$

The final answer is:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & -4 & -3 \end{bmatrix} x + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 2 & 2 \end{bmatrix} u \tag{19}$$

$$y = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} u \tag{20}$$

5 ODE2SS (Python)

The code is available in the problem5.ipynb file. You need to have Jupyter Notebook installed on your computer to run the notebook, NumPy library is also required. Also you can read the code below.

problem5

February 12, 2020

1 Problem 5. (Assignment 1)

1.1 Introduction

In this solution I assume that $x = \begin{bmatrix} x \\ x' \\ \dots \\ x^{n-1} \end{bmatrix}$

1.2 Functions

Here I define a function that gets an array of coefficients as an input an returns matrix A for SS model.

[1]:
```
import numpy as np
```

[2]:
```
def ode2matrix(a):
    """
    Return A obtained from mult. a0...an
    Test1:
    >>> ode2matrix(np.array([0, 2, 4, 3, 1]))
    array([[ 0.,  1.,  0.,  0.],
           [ 0.,  0.,  1.,  0.],
           [ 0.,  0.,  0.,  1.],
           [ 0., -2., -4., -3.]])
    """
    size = len(a)
    # creating zeros for the first column
    zeros = np.zeros((size - 2, 1))
    # the last row of A. normalization happens here
    last_row = [-a[:-1] / a[-1]]
    return np.block([[zeros, np.eye(size - 2)], last_row])
```

A function that returns a vector b for a given ODE.

[3]:
```
def ode2b(a, b0):
    """
    Return vector b for state space from a0...an and b0
    Test1:
    >>> ode2b(np.array([0, 2, 4, 3, 4]), 2)
```

1

```
    array([[0. ],
           [0. ],
           [0. ],
           [0.5]])
    """
    return np.concatenate((np.zeros((len(a) - 2, 1)), np.array([[b0 / a[-1]]])))
```

1.3 ODE2SS

This is a function that returns a pair (A, b) for a given ODE. Note that a should be a numpy array that begins with a_0 and ends with a_n. b_0 is right-hand constant. ODE:

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

[4]:
```
def ode2ss(a, b0):
    return ode2matrix(a), ode2b(a, b0)
```

1.4 Example

$$-x''' + 5x'' + 3x' + 7x = 10$$

[5]:
```
#ode2ss(np.array([0, 2, 4, 3, 1]), 6)
ode2ss(np.array([7, 3, 5, -1]), 10)
```

[5]:
```
(array([[0., 1., 0.],
        [0., 0., 1.],
        [7., 3., 5.]]), array([[  0.],
        [  0.],
        [-10.]]))
```

1.5 Doctest

This is a test section. Run it if you want to check if the code is working properly.

[6]:
```
import doctest
doctest.testmod(verbose=True)
```

```
Trying:
    ode2b(np.array([0, 2, 4, 3, 4]), 2)
Expecting:
    array([[0. ],
           [0. ],
           [0. ],
           [0.5]])
ok
Trying:
    ode2matrix(np.array([0, 2, 4, 3, 1]))
Expecting:
```

2

```
    array([[ 0.,  1.,  0.,  0.],
           [ 0.,  0.,  1.,  0.],
           [ 0.,  0.,  0.,  1.],
           [ 0., -2., -4., -3.]])
ok
2 items had no tests:
    __main__
    __main__.ode2ss
2 items passed all tests:
   1 tests in __main__.ode2b
   1 tests in __main__.ode2matrix
2 tests in 4 items.
2 passed and 0 failed.
Test passed.
```

[6]:
```
TestResults(failed=0, attempted=2)
```

[7]:
```python
#ode2matrix(np.array([7, 3, 5, -1]))
#ode2matrix(np.array([0, 2, 4, 3, 1]))
```

[8]:
```python
ode2b(np.array([0, 2, 4, 3, 4]), 2)
```

[8]:
```
array([[0. ],
       [0. ],
       [0. ],
       [0.5]])
```

3

6 ODEs and SSs (Python)

problem6

February 13, 2020

1 Problem 6

In this notebook you can find my solutions of the problem 6 of the Assignment 1.

[1]:
```
from pylab import *
from scipy.integrate import *
```

1.1 Stability

Let us check if the given system is stable or not.

[2]:
```
a = np.array([[0, 1], [1/2, 1]]) # matrix A
b = np.array([[0, 0], [3/4, -5/4]])

np.linalg.eig(a)
```

[2]:
```
(array([-0.3660254,  1.3660254]), array([[-0.9390708 , -0.59069049],
        [ 0.34372377, -0.80689822]]))
```

As you can see the system has one eigenvalue with positive real part. That means the system is not stable.

1.2 ODE solution

This function solves the ODE and draws the plot of it.

[3]:
```
def solve_ode(f, init,  t = linspace(0, 5, 1000)):
    # solving the ode
    result = odeint(f, init, t)

    x0 = result[:, 0]

    # draw a plot
    plot(t,x0,lw=2)
    xlabel('t')
    ylabel('x')
    grid()

    return t, x0
```

1

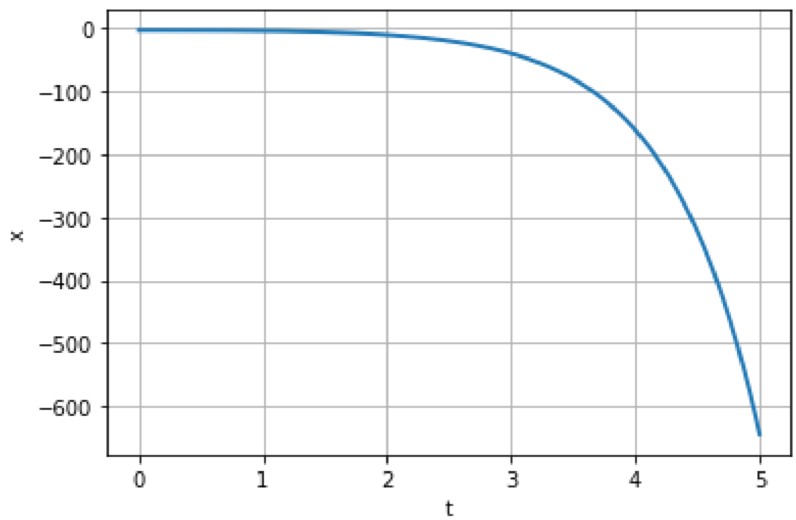
Solve the given ODE

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

Let us define the function that represents our ODE. Since pylab does not allow solving equations rather than in standard form. But x and y can be vectors, so we can easily solve it in vector form.

[4]:
```
# define the ODE function
def f(x, t):
    x0, x1 = x
    return [x1, x1 - 5/4 * t + 1/2 * x0 + 3/4]
```

[5]:
```
init = [-3, 0]
ode = solve_ode(f, init)
```



1.3 State Space solver

This function gets two matrices as input, solve the given SS and draws its plot.

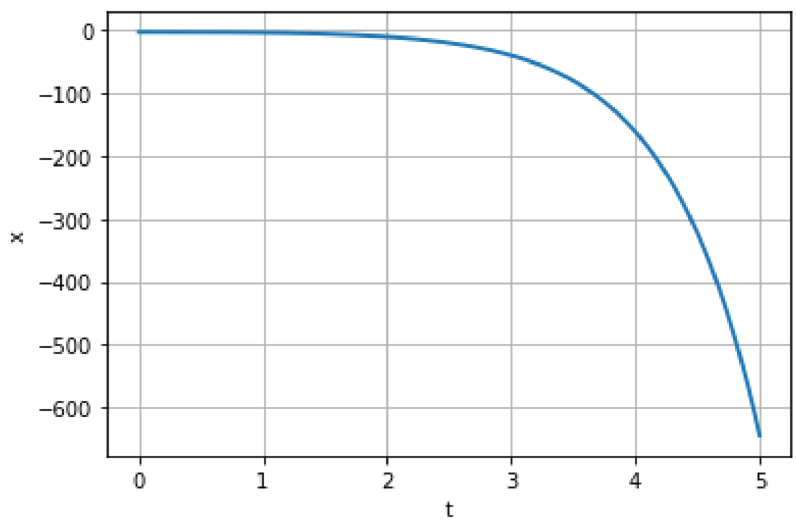
[6]:
```
def ss_solve(A, B, f, init):
    t = linspace(0, 5, 1000)

    # solving the ode
    result = odeint(f, init, t)
    x0 = result[:, 0]
```

2

```
    # draw a plot
    plot(t,x0,lw=2)
    xlabel('t')
    ylabel('x')
    grid()
    return (t, x0)
```

[7]:
```
def f_ss(x, t):
    x = np.reshape(x, (2, -1))
    return ((a.dot(x)) + b.dot([[1], [t]])).T.tolist()[0]
```

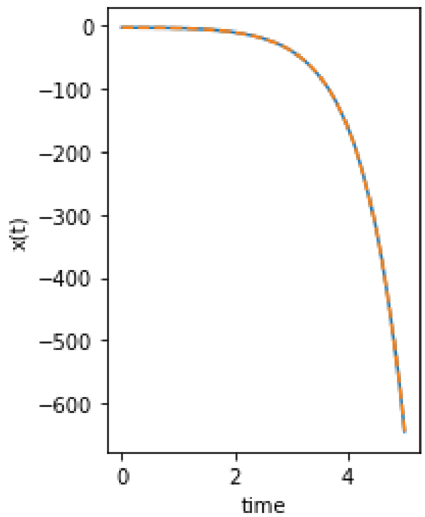
[8]:
```
ss = ss_solve(a, b, f_ss, [-3, 0]) # this is the same graph
```



[9]:
```
import matplotlib.pyplot as plt
# ode based model
plt.subplot(121)
plt.plot(*ode)
plt.plot(*ss, '--')
plt.xlabel('time')
plt.ylabel('x(t)')
```

[9]: `Text(0, 0.5, 'x(t)')`

3



As you can see the solutions are the same.

4